AVR276：AT90USBxxxマイクロ コントローラ用 USBソフトウェア ライブラリ

要点

- 低速(1.5Mbps)と全速(12Mbps)のデータ速度
- 制御、大量(バルク)、等時(アイソクロノス)、割り込みの転送形式
- 6つまでのデータ エンドポイント
- 単一または2重の緩衝
- 装置動作形態:
  - AVR USBソフトウェア ライブラリでの標準または独自のUSB装置クラス
- 縮小ホスト動作形態:
  - 装置記述子でのホスト パイプ自動形態設定
  - 複合装置(複数インターフェース)支援

1. 叙述

この資料はAT90USBxxx USBソフトウェア ライブラリを記述し、そしてこのライブラリを用いてUSB装置または縮小したホスト応用の開発方法を図解します。

この資料はAT90USBxxxのためのそれらの応用(装置または縮小されたホスト動作形態)の開発を手助けするため、ソフトウェア開発者用に書かれています。この資料の内容を理解するためにUSB仕様2.0(www.usb.org)の9章の知識も必要とされます。

1.1. 概要

AT90USBxxxソフトウェア ライブラリはソフトウェア開発者からUSB開発(と特に列挙(接続認識)段階)の複雑さを隠すために設計されています。

この資料の狙いはUSBファームウェアを記述してその基本設計の概要を与えることです。ファームウェアを独自化して自身の応用を構築する最も簡単な方法を使用者に与えるために主なファイルが記述されます。

AT90USBxxxソフトウェア ライブラリはこれらのライブラリの使用法を図解するために両方の役割(装置と縮小ホスト)の応用例も提供します。

1.2. 制限

・OTG(On The Go)準拠(SRP/HNP要求管理)の完全な支援は未だ統合されていません。

1.3. 用語

VID：USB供給者識別子
PID：USB製品識別子

2. コーディング様式

これより下で説明されるコーディング様式はファームウェアを理解するのに重要です。

・定義された定数は大文字を使用します。

```
#define FOSC 8000
```

・マクロ関数は先頭文字を大文字として使用します。

```
#define Is_usb_sof() ((UDINT & MSK_SOFI) ? TRUE: FALSE)
```

・使用者応用はusb_conf.hで以下のように定義された引っ掛け(フック)によって各USB事象で自身の特定命令を実行することができます。

```
#define Usb_sof_action() sof_action();
```

注: 引っ掛け(フック)関数は短時間を必要とする操作だけを実行すべきです。

・Usb_unicode()マクロ関数はUSB規約でユニコード文字が交換される処(文字列記述子など)毎に使用されるべきです。

8ビット AVR® マイクロ コントローラ

応用記述

Rev. 7675A-03/07, 7675AJ1-02/14

3. USBﾊﾞｽ

3.1. ﾊﾞｽ接続構成

USB仕様はﾎｽﾄと装置の2つの異なるUSBﾊﾞｽ上での節点の形式を定義します。

■ USBﾎｽﾄ:

・どのUSB系にも1つだけのﾎｽﾄがあり、USBﾊﾞｽの"主"として動きます。

・ﾎｽﾄ系に対するUSBｲﾝﾀｰﾌｪｰｽはﾎｽﾄ制御器として参照されます。

■ USB装置:

・USB装置はUSBﾊﾞｽの従節点(ﾉｰﾄﾞ)を動かします。

・(USB装置としても働く)USBﾊﾌﾞによってUSBﾊﾞｽに最大127の装置を接続できます。各装置は装置ｱﾄﾞﾚｽによって個別に識別されます。

AT90USBxxxﾃﾞﾊﾞｲｽはUSB装置またはUSBﾎｽﾄの両方として動作することができ、正確にはﾎｽﾄ動作形態でのAT90USBxxxは縮小されたﾎｽﾄ制御器として動作します。縮小ﾎｽﾄ制御器は独自のUSBﾎﾟｰﾄを持ち、ﾊﾌﾞでの完全なUSB木構造を扱いません。これは縮小ﾎｽﾄ制御器が固有のUSB装置とで独自の点対点接続を扱うように設計されていることを意味します。縮小ﾎｽﾄ応用は既知の目的対象装置一覧(VID/PID一覧)を支援します。この一覧に挙げられた装置だけが応用によって支援されます。加えてAT90USBxxx USBｿﾌﾄｳｪｱ ﾗｲﾌﾞﾗﾘはｸﾗｽ(CLASS)/補助ｸﾗｽ(SUBCLASS)/規約(PROTOCOL)の目的対象とされた一覧を支援することができます。

図3-1. USB標準接続構成

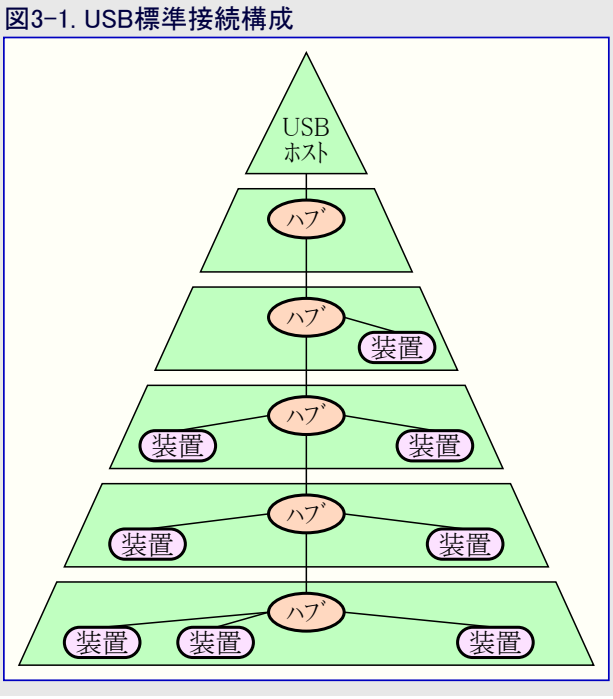
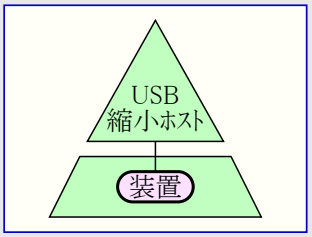


図3-2. 縮小ﾎｽﾄ接続構成



AT90USBxxxｿﾌﾄｳｪｱ ﾗｲﾌﾞﾗﾘは以下のUSB動作形態の1つを管理するように形態設定することができます。

■ USB装置

■ USB縮小ﾎｽﾄ

■ USB二重役割装置

・この形態は直前で定義された両形態の支援を許します。動作形態は外部のUSB IDﾋﾟﾝによって定義されます。GNDに繋がれた(ﾐﾆAﾌﾟﾗｸﾞに接続された)IDﾋﾟﾝはUSB縮小ﾎｽﾄ形態への移行をAT90USBxxxに許します。IDﾋﾟﾝが接続されない((ﾐﾆBﾌﾟﾗｸﾞに接続された)場合、AT90USBxxxは装置形態で動作します。

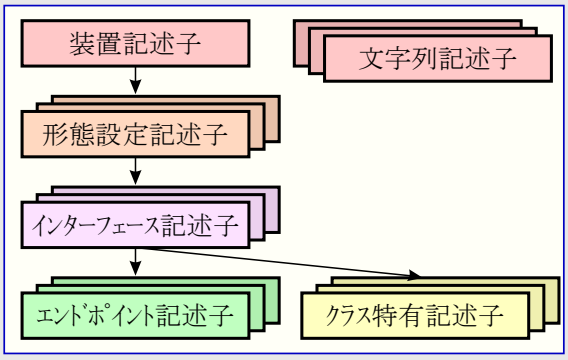
3.2. USB記述子

列挙(接続認識)処理中、ﾎｽﾄはそれを識別して正しいﾄﾞﾗｲﾊﾞを設定するために装置へ様々な記述子を問います。各USB装置はﾎｽﾄによって認証されるために最低、右図で示される記述子を持つべきです。

どのUSB応用でも最も難しい部分は何の装置記述子であるべきかを判断することです。全てのUSB装置は列挙(接続認識)と呼ばれる処理を通してその必要条件をﾎｽﾄと通信します。AT90USBxxxｿﾌﾄｳｪｱ ﾗｲﾌﾞﾗﾘは装置または縮小ﾎｽﾄの両形態に関して完全な列挙(接続認識)処理を提供します。

列挙(接続認識)中、装置記述子は装置へ固有のｱﾄﾞﾚｽを割り当てるﾎｽﾄへ転送されます。記述子はUSB2.0仕様の9章で詳細に記述されます。

図2-1. USB記述子



3.2.1. 装置記述子

USB装置は装置記述子を1つだけ持ちます。この記述子は装置全体を表します。それはUSB版、ｴﾝﾄﾞﾎﾟｲﾝﾄ0の最大ﾊﾟｹｯﾄ容量、供給者ID、製品ID、製品版、装置が持てる可能な形態設定数などについての情報を与えます。

これより下の表はこの記述子の形式を示します。

表3-1. 装置記述子

領域名	説明
bLength	記述子の大きさ
bDescriptorType	装置記述子
bcdUSB	USB版番号
bDeviceClass	ｸﾗｽ符号(0の場合はｸﾗｽが各ｲﾝﾀｰﾌｪｰｽによって指定されます、$FFの場合は供給者によって指定されます。)
bDeviceSubClass	(USB協会により割り当てられた)補助ｸﾗｽ符号
bDeviceProtocol	(USB協会により割り当てられた)規約符号
bMaxPacketSize	ｴﾝﾄﾞﾎﾟｲﾝﾄ0のﾊﾞｲﾄでの最大ﾊﾟｹｯﾄ容量。それは8(低速)、16,32,64(全速)でなければなりません。
idVendor	(USB協会により割り当てられた)供給者識別
idProduct	(製造(供給)者により割り当てられた)製品識別
bcdDevice	(製造(供給)者により割り当てられた)装置版番号
iManufacturer	製造(供給)者記述子の文字列配列への指標
iProduct	製品記述子の文字列配列への指標
iSerialNumber	通番記述子の文字列配列への指標
bNumConfigurations	形態設定数

3.2.2. 形態設定記述子

USB装置は1つよりも多くの形態設定記述子を持てますが、装置の大多数は単一の形態設定を使用します。この記述子は電力供給形態(自己給電またはﾊﾞｽ給電)、装置によって消費され得る最大電力、装置に付属するｲﾝﾀｰﾌｪｰｽ、全てのﾃﾞｰﾀ記述子の総容量などを指定します。

例えば1つの装置は、それがﾊﾞｽによって給電される時の一方と、それが自己給電にされる時の他方の、2つの形態設定を持つことができます。異なる転送形態を使用する形態設定も考えられます。

右の表はこの記述子の形式を示します。

表3-2. 形態設定記述子

領域名	説明
bLength	記述子の大きさ
bDescriptorType	形態設定記述子
wTotalLength	記述子の総容量
bNumInterfaces	ｲﾝﾀｰﾌｪｰｽ数
bConfigurationValue	形態設定番号
bmAttributes	ﾊﾞｽ給電または自己給電、遠隔起動
bMaxPower	2mA単位の最大消費電流

3.2.3. ｲﾝﾀｰﾌｪｰｽ記述子

1つの装置は1つよりも多くのｲﾝﾀｰﾌｪｰｽを持つことができます。この記述子によって与えられる主な情報はこのｲﾝﾀｰﾌｪｰｽによって使用されるｴﾝﾄﾞﾎﾟｲﾝﾄ数とUSBｸﾗｽ、補助ｸﾗｽです。

この下の表はこの記述子の形式を示します。

表3-3. ｲﾝﾀｰﾌｪｰｽ記述子

領域名	説明
bLength	記述子の大きさ
bDescriptorType	ｲﾝﾀｰﾌｪｰｽ記述子
bInterfaceNumber	ｲﾝﾀｰﾌｪｰｽ番号
bAlternateSetting	置き換えるｲﾝﾀｰﾌｪｰｽを選ぶのに使用
bNumEndpoints	(ｴﾝﾄﾞﾎﾟｲﾝﾄ0を除く)ｴﾝﾄﾞﾎﾟｲﾝﾄ数
bInterfaceClass	(USB協会により割り当てられた)ｸﾗｽ符号
bInterfaceSubClass	(USB協会により割り当てられた)補助ｸﾗｽ符号 (0:補助ｸﾗｽなし、1:ﾌﾞｰﾄ ｲﾝﾀｰﾌｪｰｽ補助ｸﾗｽ)
iInterface	使用するｲﾝﾀｰﾌｪｰｽを記述するための文字列配列への指標

3.2.4. エンドポイント記述子

この記述子は、方向(INまたはOUT)、支援する転送形式(割り込み、大量、等時)、エンドポイントの容量、割り込み転送形態の場合に於けるデータ転送の間隔などのようなエンドポイントのパラメータを記述するために用いられます。

この下の表はこの記述子の形式を示します。

表2-4. エンドポイント記述子

領域名	説明
bLength	記述子の大きさ
bDescriptorType	エンドポイント記述子
bEndpointAddress	エンドポイントのアドレス (ビット7:方向,0=出力,1=入力、ビット6～4:予約(0に設定)、ビット3～0:エンドポイント番号)
bmAttributes	ビット7～2:等時転送を除いて予約 ビット1,0:転送形式 (11:割り込み,10:大量,01:等時,00:制御) 低速(Loe-Speed)では制御と割り込みの動作形態だけが許されます。
wMaxPacketSize	このエンドポイントが支援する最大データ容量
bInterval	エンドポイントのデータ転送を要求するための時間間隔です。この値はフレーム数(ms)で与えられます。 大量と制御の転送では無視されます。 等時に対しては1～16(ms)間の値を設定してください。 全速での割り込み転送に対しては1～255(ms)の値を設定してください。 低速での割り込み転送に対しては10～255(ms)の値を設定してください。

4. ファームウェア基本設計

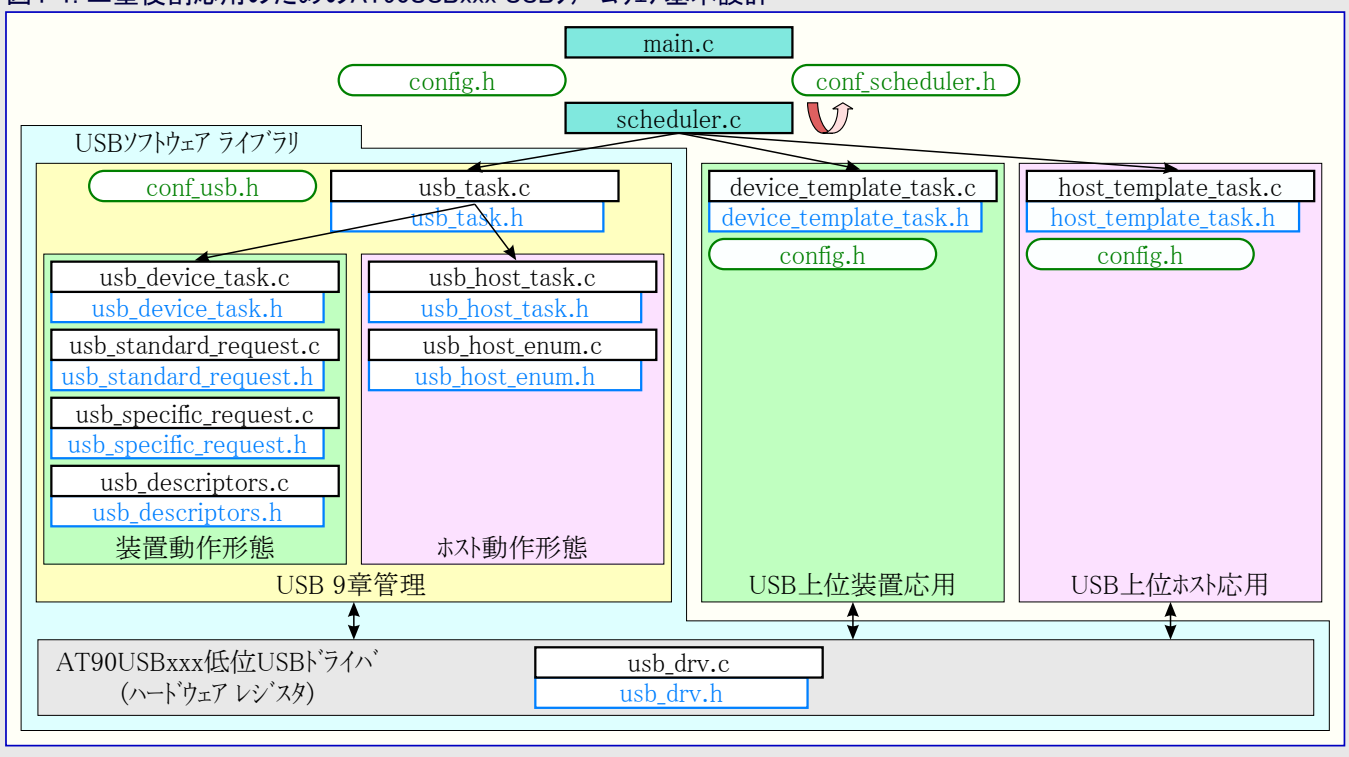
図4-1.で示されるように、USBファームウェアの基本設計はどんなハードウェア インターフェースをも避けるように設計されています(ドライバ層は使用者によって変更されるべきではありません)。USBソフトウェア ライブラリは装置とホストの両方のUSB 9章の列挙(接続認識)手順を管理することができます。

全体的なUSBファームウェア基本設計はUSB IDピンに依存して装置またはホストとしての動作をAT90USBxxxに許す二重役割見本応用"雛形"によって図解されます。

見本の二重役割応用は以下の3つの異なる作業に基きます。

- usb_task(usb_task.c)は装置またはホストに於けるUSB低位列挙(接続認識)手順を実行する作業です。USB接続が完全に動作することを一旦この作業が検知すると、上位の応用作業で検査することができる各種状態フラグを更新します。
- 装置雛形作業(device_template_task.c)は高位装置応用動作を実行します。この作業は一旦装置が列挙(接続認識)されると実行することができるUSB装置使用者応用を含みます。
- ホスト雛形作業(host_template_task.c)は一旦接続した装置が列挙(接続認識)されると、上位ホスト応用動作を実行します。

図4-1. 二重役割応用のためのAT90USBxxx USBファームウェア基本設計



4.1. 見本応用について

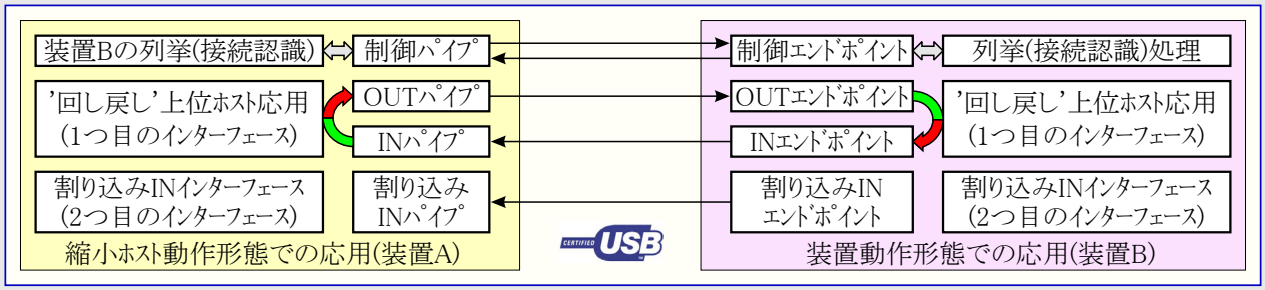
USBｿﾌﾄｳｪｱ ﾗｲﾌﾞﾗﾘを図解するのに使用した見本応用は両方のUSB動作形態(装置とﾎｽﾄ)で動くことができます。

装置動作形態は以下の2つのｲﾝﾀｰﾌｪｰｽを提供します。

- 1つ目の大量(ﾊﾞﾙｸ)IN/OUTｲﾝﾀｰﾌｪｰｽは回し戻しﾃﾞｰﾀ転送で動きます(OUTｴﾝﾄﾞﾎﾟｲﾝﾄで受信された全ﾃﾞｰﾀはINｴﾝﾄﾞﾎﾟｲﾝﾄに送り返されます)。
- 2つ目のｲﾝﾀｰﾌｪｰｽは割り込みINｴﾝﾄﾞﾎﾟｲﾝﾄでﾃﾞｰﾀを送信します。

見本応用のﾎｽﾄ動作形態は認証、列挙(接続認識)し、装置応用の両ｲﾝﾀｰﾌｪｰｽを使用します。

図4-2. 見本応用概要



注: この見本応用のために使用される装置Bの装置記述子は標準PCﾎｽﾄ ｼｽﾃﾑでの列挙(接続認識)のために直接使用可能ではありません。"実際の"装置応用例(HIDﾏｳｽ、HIDｷｰﾎﾞｰﾄﾞ、大容量記憶、CDCなど)についてはATMELのｳｪﾌﾞｻｲﾄを参照してください。

4.2. ソース ファイルの基本設計

図4-3. ソース ファイル構成

	要約説明
AT90USB128	
AT90USB128 / lib_mcu	
AT90USB128 / lib_mcu / usb	
AT90USB128 / lib_mcu / usb / usb_drv.c AT90USB128 / lib_mcu / usb / usb_drv.h	USBインターフェース下位ドライバ(レジスタ抽象化層)
AT90USB128 / demo	
AT90USB128 / demo / template	
AT90USB128 / demo / template / cont	
AT90USB128 / demo / template / cont / config.h	応用のための全体形態設定ファイル
AT90USB128 / demo / template / cont / conf_usb.h	USBソフトウェア ライブラリ用形態設定ファイル
AT90USB128 / demo / template / cont / conf_scheduler.h	計画機構形態設定(作業宣言)
AT90USB128 / demo / template / cont / main.c	主入口(計画機構初期化)
AT90USB128 / demo / template / cont / usb_specific_request.c AT90USB128 / demo / template / cont / usb_specific_request.h	使用者またはクラス特有装置列挙(接続認識)要求 (非標準列挙(接続認識)要求)
AT90USB128 / demo / template / cont / device_template_task.c AT90USB128 / demo / template / cont / device_template_task.h	USB装置動作形態用上位使用者応用 (見本装置応用)
AT90USB128 / demo / template / cont / host_template_task.c AT90USB128 / demo / template / cont / host_template_task.h	USBホスト動作形態用上位使用者応用 (見本装置応用)
AT90USB128 / demo / template / cont / usb_descriptors.c AT90USB128 / demo / template / cont / usb_descriptors.h	(装置列挙(接続認識)手順で使用される) 装置記述子構造体
AT90USB128 / modules	
AT90USB128 / modules / usb	
AT90USB128 / modules / usb / device_chap9	
AT90USB128 / modules / usb / device_chap9 / usb_device_task.c AT90USB128 / modules / usb / device_chap9 / usb_device_task.h	USB装置9章管理 (接続,切断,休止,再開,列挙(接続認識)手順呼び出し)
AT90USB128 / modules / usb / device_chap9 / usb_standard_request.c AT90USB128 / modules / usb / device_chap9 / usb_standard_request.h	装置列挙(接続認識)手順 (標準要求)
AT90USB128 / modules / usb / host_chap9	
AT90USB128 / modules / usb / host_chap9 / usb_host_task.c AT90USB128 / modules / usb / host_chap9 / usb_host_task.h	USBホスト9章管理 (装置接続,切断,休止,再開,上位列挙(接続認識)手順)
AT90USB128 / modules / usb / host_chap9 / usb_host_enum.c AT90USB128 / modules / usb / host_chap9 / usb_host_enum.h	下位列挙(接続認識)関数 (VID/PID検査,装置記述子に従ったホスト パイプ形態設定)
AT90USB128 / modules / usb / usb_task.c AT90USB128 / modules / usb / usb_task.h	USB作業管理用入口
AT90USB128 / lib_board	
AT90USB128 / lib_board / stk_525	
AT90USB128 / lib_board / stk_525 / stk_525.c AT90USB128 / lib_board / stk_525 / stk_525.h	
AT90USB128 / lib_board / usb_key	
AT90USB128 / lib_board / usb_key / usb_key.c AT90USB128 / lib_board / usb_key / usb_key.h	

5. USBソフトウェア ライブラリ形態設定

見本応用はホストと装置の両機能実装のために予め形態設定されています。これは装置または縮小ホストの形態だけで使用するように形態設定することもできます。選んだUSB動作形態に依存して、USB作業は9章要求を管理するためにusb_host_taskまたはusb_device_taskのどちらかを呼び出します。このような場合、対応するtemplate_device_taskまたはtemplate_host_taskは計画された作業から取り除くことができます。

5.1. 全体形態設定

応用に関する全ての共通形態設定パラメータは"config.h"ファイルで定義されます(XTAL周波数、CPU形式など)。単位部署特有パラメータはそれらの各々の形態設定ファイルで定義されます。

5.2. 計画機構形態設定

見本応用は全体の応用基本設計と構成を変更せずに応用的な作業の作成と追加を使用者に許す、簡単な作業計画機構を提供します。この計画機構はどんな優先権もなしに予め定義された順に予め定義された全ての作業を呼びます。作業はそれが終了されるまで実行され、そして次の作業を呼びます。

計画機構の作業は"conf_scheduler.h"で定義され、ここで使用者は自身の作業を宣言することができます。

見本USB二重役割応用に関して、以下の計画機構形態設定ﾊﾟﾗﾒｰﾀが使用されます。

```
#define Scheduler_task_1_init    usb_task_init
#define Scheduler_task_1         usb_task
#define Scheduler_task_2_init    device_template_task_init
#define Scheduler_task_2         device_template_task
#define Scheduler_task_4_init    host_template_task_init
#define Scheduler_task_4         host_template_task
```

scheduler_task_n_init関数は計画機構の始動で一度だけ実行され、ところがscheduler_task_n関数は永遠の繰り返しで実行されます。

5.3. USBﾗｲﾌﾞﾗﾘ形態設定

USBﾗｲﾌﾞﾗﾘは"conf_usb.h"によって形態設定することができます。このﾌｧｲﾙは装置とﾎｽﾄの両方のUSB動作形態設定ﾊﾟﾗﾒｰﾀを含みます。この形態設定ﾌｧｲﾙは全体、装置、ﾎｽﾄの形態設定ﾊﾟﾗﾒｰﾀに関して3つの領域に分かれます。

全体形態設定領域はﾗｲﾌﾞﾗﾘが装置やﾎｽﾄのUSB形態を使用するのかどうかと、(応用電源範囲に依存して)内部USBﾊﾟｯﾄﾞ(電圧)調整器が応用に対して許可されるべきかどうかの選択を許します。

5.4. 装置形態設定

USBﾗｲﾌﾞﾗﾘは装置Bに対してUSB 9章を管理します。

- 接続/切断 (VBUS監視)
- 休止
- 再開
- 列挙(接続認識)要求

非同期USB事象(接続、休止、再開、ﾘｾｯﾄ)は"usb_task.c"ﾌｧｲﾙ内に配置されたUSB割り込みｻﾌﾞﾙｰﾁﾝ内で管理されます。使用者応用は"conf_usb.h"ﾌｧｲﾙの使用者定義活動によってそれらの事象で特定関数を実行することができます。

ﾎｽﾄからの列挙(接続認識)要求は"usb_device_task.c"と"usb_standard_request.c"のﾌｧｲﾙでのﾎﾟｰﾘﾝｸﾞ形態で管理されます。計画機構に属するusb_taskは新しい制御要求に関してﾎｽﾄから周期的に検査されます。

5.4.1. USBﾗｲﾌﾞﾗﾘの形態設定

ﾗｲﾌﾞﾗﾘのUSB装置動作形態を許可するにはUSB_DEVICE_FEATUREがENABLEDとして定義されるべきです。

"conf_usb.h"ﾌｧｲﾙの装置特有形態設定領域は装置応用によって使用される物理的なｴﾝﾄﾞﾎﾟｲﾝﾄ番号定義と、USB通信中に特別な事象で実行することができる使用者指定活動の組を含みます。

見本応用に関しては以下の通りです。

```
#define NB_ENDPOINTS 4           // ｴﾝﾄﾞﾎﾟｲﾝﾄ0を含めた応用でのｴﾝﾄﾞﾎﾟｲﾝﾄ数
#define EP_TEMP_IN 1
#define EP_TEMP_OUT 2
#define EP_TEMP_INT_IN 3
// 各USB事象と連携する活動をここに書いてください。
// 機能の邪魔をせず順番で時間を浪費しないように注意してください。
#define Usb_sof_action()         sof_action();
#define Usb_wake_up_action()
#define Usb_resume_action()
#define Usb_suspend_action()
#define Usb_reset_action()
#define Usb_vbus_on_action()
#define Usb_vbus_off_action()
#define Usb_set_configuration_action()
```

使用者活動は特定関数を実行するための使用者上位応用を定義します。例えば使用者はUSBﾌﾚｰﾑ開始またはUSBﾊﾞｽ ﾘｾｯﾄの各々で実行される関数を割り付けることができます。

5.4.2. 複合装置について

複合装置はUSBバス上の複数周辺機能として検出されることをUSB装置に許します。従って複合装置はその形態設定記述子内に複数のインターフェースを宣言します。各インターフェースはそれ自身のクラス/補助クラスと連携する上位応用の動きを持ちます(例えば複合装置は大容量記憶応用との連携でHIDインターフェースとして動作することができます)。

USBソフトウェア ライブラリで図解される見本複合装置は以下の異なる2つのインターフェースを宣言します。

- 2つの大量(バルク)IN/大量OUTエンドポイントを持つ1つ目のインターフェース
- 単一の割り込みINエンドポイントを持つ2つ目のインターフェース

5.4.3. 装置記述子の形態設定

装置列挙(接続認識)手順に用いられる装置記述子は"usb_descriptors.c"と"usb_descriptors.h"のファイルに格納します。

"usb_descriptors.h"ファイルで記述子構造体型が宣言され、使用者は自身の装置形態設定に関する全ての列挙(接続認識)パラメータをここで宣言すべきです。

形態設定記述子型は"usb_descriptors.h"ファイルの最後で以下のように定義されます。

```
// 形態設定記述子雛形
// 装置は2つのインターフェースを持ちます。
// - 1つ目のインターフェースは2つの大量(バルク)エンドポイントを持ちます。
// - 2つ目のインターフェースは1つの割り込みINエンドポイントを持ちます。
typedef struct
{
    S_usb_configuration_descriptor      cfg_temp;
    S_usb_interface_descriptor          ifc_temp;
    S_usb_endpoint_descriptor           ep1_temp;
    S_usb_endpoint_descriptor           ep2_temp;
    S_usb_interface_descriptor          ifc_second_temp;
    S_usb_endpoint_descriptor           ep3_temp;
} S_usb_user_configuration_descriptor;
```

連携するインターフェースとエンドポイントのパラメータは"usb_descriptors.h"ファイルの始めで宣言されます。

```
// USB装置記述子
#define USB_SPECIFICATION               0x0200
#define DEVICE_CLASS                    0       //! 各インターフェースはそれ自身のクラスを持ちます。
#define DEVICE_SUB_CLASS                0       //! 各インターフェースはそれ自身の補助クラスを持ちます。
#define DEVICE_PROTOCOL                 0       //! 各インターフェースはそれ自身の規約を持ちます。
#define EP_CONTROL_LENGTH               64
#define VENDOR_ID                       0x03EB  // ATMELの供給者ID=$03EB
#define PRODUCT_ID                      0x0000
#define RELEASE_NUMBER                  0x1000
#define MAN_INDEX                       0x01
#define PROD_INDEX                      0x02
#define SN_INDEX                        0x03
#define NB_CONFIGURATION                1
  // 形態設定
#define NB_INTERFACE                    2       //! この形態設定に関するインターフェース数
#define CONF_NB                         1       //! この形態設定の番号
#define CONF_INDEX                      0
#define CONF_ATTRIBUTES                 USB_CONFIG_SELFPOWERED
#define MAX_POWER                       50      // 100mA (2mA単位!)
  // USBインターフェース記述子(gen)
#define INTERFACE_NB_TEMP               0       //! このインターフェースの番号
#define ALTERNATE_TEMP                  0       //! このインターフェースの代替設定番号
#define NB_ENDPOINT_TEMP                2       //! このインターフェースが持つエンドポイント数
#define INTERFACE_CLASS_TEMP            0x00    //! クラス
#define INTERFACE_SUB_CLASS_TEMP        0x00    //! 補助クラス
#define INTERFACE_PROTOCOL_TEMP         0x00    //! 規約
#define INTERFACE_INDEX_TEMP            0
```

[次頁へ続く]

```
  // USBｴﾝﾄﾞﾎﾟｲﾝﾄ1記述子(FS)
#define ENDPOINT_NB_TEMP1                   (EP_TEMP_IN | 0x80)
#define EP_ATTRIBUTES_TEMP1                 0x02        // 大量(ﾊﾞﾙｸ)=$02,割り込み=$03
#define EP_IN_LENGTH_TEMP1                  64
#define EP_SIZE_TEMP1                       EP_IN_LENGTH_TEMP1
#define EP_INTERVAL_TEMP1                   0x00        // ﾎｽﾄからの割り込みﾎﾟｰﾘﾝｸﾞ間隔
 // USBｴﾝﾄﾞﾎﾟｲﾝﾄ2記述子(FS)
#define ENDPOINT_NB_TEMP2                   EP_TEMP_OUT
#define EP_ATTRIBUTES_TEMP2                 0x02        // 大量(ﾊﾞﾙｸ)=$02,割り込み=$03
#define EP_IN_LENGTH_TEMP2                  64
#define EP_SIZE_TEMP2                       EP_IN_LENGTH_TEMP2
#define EP_INTERVAL_TEMP2                   0x00        // ﾎｽﾄからの割り込みﾎﾟｰﾘﾝｸﾞ間隔
  // USB第2ｲﾝﾀｰﾌｪｰｽ記述子(gen)
#define INTERFACE_NB_SECOND_TEMP            1           //! このｲﾝﾀｰﾌｪｰｽの番号
#define ALTERNATE_SECOND_TEMP               0           //! このｲﾝﾀｰﾌｪｰｽの代替設定番号
#define NB_ENDPOINT_SECOND_TEMP             1           //! このｲﾝﾀｰﾌｪｰｽが持つｴﾝﾄﾞﾎﾟｲﾝﾄ数
#define INTERFACE_CLASS_SECOND_TEMP         0x00        //! ｸﾗｽ
#define INTERFACE_SUB_CLASS_SECOND_TEMP     0x55        //! 補助ｸﾗｽ
#define INTERFACE_PROTOCOL_SECOND_TEMP      0xAA        //! 規約
#define INTERFACE_INDEX_SECOND_TEMP         0
  // USBｴﾝﾄﾞﾎﾟｲﾝﾄ2記述子(FS)
#define ENDPOINT_NB_TEMP3                   (EP_TEMP_INT_IN | 0x80)
#define EP_ATTRIBUTES_TEMP3                 0x03        // 大量(ﾊﾞﾙｸ)=$02,割り込み=$03
#define EP_IN_LENGTH_TEMP3                  64
#define EP_SIZE_TEMP3                       EP_IN_LENGTH_TEMP2
#define EP_INTERVAL_TEMP3                   20          // ﾎｽﾄからの割り込みﾎﾟｰﾘﾝｸﾞ間隔
```

記述子領域を満たすのに使用されるこれら全ての列挙(接続認識)ﾊﾟﾗﾒｰﾀは"usb_descriptors.c"ﾌｧｲﾙで宣言されます。ﾎｽﾄ制御器が列挙(接続認識)手順を実行する時に、その要求は"standard_request.c"の列挙(接続認識)関数によって復号され、予め定義された記述子がﾎｽﾄ制御器へ送られます。

5.5. 縮小ﾎｽﾄ形態設定

USBﾗｲﾌﾞﾗﾘは縮小ﾎｽﾄ制御器用のUSB 9章を管理します。

- VBUSの生成と管理
- 周辺機能の接続
- 周辺機能の切断
- 接続した周辺機能の列挙(接続認識)と識別
- 接続した周辺機能の装置記述子に従ったﾎｽﾄ制御器ﾊﾟｲﾌﾟの形態設定
- USB活動の休止
- 再開と遠隔起動管理

ﾗｲﾌﾞﾗﾘの装置動作形態と同様に、非同期USB事象(接続、切断、遠隔起動の検出)は"usb_task_c"ﾌｧｲﾙに配置されたUSB割り込みｻﾌﾞﾙｰﾁﾝ内で直接管理されます。"conf_usb.h"ﾌｧｲﾙの使用者定義活動によって使用者応用はこれらの事象で特定の関数を実行することができます。

接続した装置の状態とそれの列挙(接続認識)手順は、"usb_host_task.c"と"usb_host_enum.c"のﾌｧｲﾙでのﾎﾟｰﾘﾝｸﾞ形態で管理されます。装置切断と遠隔起動の検出のような重要且つ非同期の装置事象だけが("usb_task_c"ﾌｧｲﾙの)割り込み下で管理されます。

ﾗｲﾌﾞﾗﾘのUSBﾎｽﾄ動作形態を許可するには、USB_HOST_FEATUREがENABLEDとして定義されるべきです。

"conf_usb.h"ﾌｧｲﾙのﾎｽﾄ固有形態設定領域は以下の主なﾊﾟﾗﾒｰﾀの選択を許します。

- ﾎｽﾄ応用によって支援される既知の装置のVID/PIDの表
- ﾎｽﾄ応用によって支援されるｲﾝﾀｰﾌｪｰｽのｸﾗｽ/補助ｸﾗｽ/規約の表
- 接続する複合装置に対して支援する最大ｲﾝﾀｰﾌｪｰｽ数
- ｲﾝﾀｰﾌｪｰｽと連携する最大ｴﾝﾄﾞﾎﾟｲﾝﾄ数
- USB転送に関する時間制限ﾊﾟﾗﾒｰﾀ (NAK数または遅延時間)

見本応用に対する例:

```
//!  この表は縮小ﾎｽﾄ応用によって支援されるVID/PIDを含みます。
//!  VID_PID_TABLEの形式定義:
//!
//!  #define VID_PID_TABLE    {VID1, number_of_pid_for_this_VID1, PID11_value,..., PID1X_Value ¥n
//!                           ...¥n
//!                           ,VIDz, number_of_pid_for_this_VIDz, PIDz1_value,..., PIDzX_Value}
#define VID_PID_TABLE                       {0x03EB, 2, 0x201C, 0x2014 ¥
                                            ,0x0123, 3, 0x2000, 0x2100, 0x1258}

//!  @brief 支援されるｸﾗｽ/補助ｸﾗｽ/規約表一覧
//!
//!  この表は縮小ﾎｽﾄ応用によって支援されるｸﾗｽ/補助ｸﾗｽ/規約を含みます。
//!  この表定義は簡単なVID/PID表一覧の代わりにｸﾗｽ/補助ｸﾗｽ/規約の全体に対して減少された応用装置支援の
//!  拡張を許します。
//!
//!  CLASS_SUBCLASS_PROTOCOLの形式定義: ¥n
//!  #define CLASS_SUBCLASS_PROTOCOL      {CLASS1, SUB_CLASS1,PROTOCOL1, ¥n
//!                                       ...¥n
//!                                       CLASSz, SUB_CLASSz,PROTOCOLz}
#define CLASS_SUBCLASS_PROTOCOL            {¥
                                           0x00, 0x00, 0x00,¥
                                           0x00,0x55,0xAA}

//! 記述子操作の予約されたRAM緩衝部容量
#define SIZEOF_DATA_STAGE                  250
//! 接続した装置に割り当てられるｱﾄﾞﾚｽ
#define DEVICE_ADDRESS                     0x05

//! 支援することができる最大ｲﾝﾀｰﾌｪｰｽ数 (複合装置)
#define MAX_INTERFACE_SUPPORTED            0x02

//! 支援するｲﾝﾀｰﾌｪｰｽ当たりの最大ｴﾝﾄﾞﾎﾟｲﾝﾄ数
#define MAX_EP_PER_INTERFACE               3

//! ﾎｽﾄ制御器は厳密なVID/PID一覧に制限されます。
//! 許可時、装置のVID/PIDが支援一覧に属さなければ、ﾎｽﾄ制御器ﾗｲﾌﾞﾗﾘはより深い形態設定へ行かず、
//! 異常状態になります。
#define HOST_STRICT_VID_PID_TABLE          DISABLE

//! 受信した装置記述子に従ってﾎｽﾄ ﾊﾟｲﾌﾟの形態設定を試みます。
#define HOST_AUTO_CFG_ENDPOINT             ENABLE

//! ﾎｽﾄのﾌﾚｰﾑ開始割り込みを常時許可
#define HOST_CONTINUOUS_SOF_INTERRUPT      DISABLE

//! ﾎｽﾄの異常状態検出時、無接続状態になります。
#define HOST_ERROR_RESTART                 ENABLE

//! USBﾎｽﾄ ﾊﾟｲﾌﾟ転送はUSB通信割り込みを使用します(非塊化関数の使用を許します)。
#define USB_HOST_PIPE_INTERRUPT_TRANSFER   ENABLE

//! IDﾋﾟﾝ変化でWDTﾘｾｯﾄを強制
#define ID_PIN_CHANGE_GENERATE_RESET       ENABLE

//! ﾎｽﾄ転送に対して時間制限遅延(時間)を許可
#define TIMEOUT_DELAY_ENABLE               ENABLE
```

[次頁へ続く]

```
//! 時間制限値の前に1/4秒(250ms)遅延
#define TIMEOUT_DELAY                       1

//! ホスト転送に対してNAK捕獲時間制限を許可
#define NAK_TIMEOUT_ENABLE                  ENABLE

//! 送信関数用の時間制限前のNAKハンドシェーク数 (最大$FFFF)
#define NAK_SEND_TIMEOUT                    0x0010

//! 受信関数用の時間制限前のNAKハンドシェークのNAK数 (最大$FFFF)
#define NAK_RECEIVE_TIMEOUT                 0x0010

//! 縮小ホストに対して、エンドポイント7でのVBUS生成器の制御だけを許します。
#define SOFTWARE_VBUS_CTRL                  ENABLE
#if (HOST_AUTO_CFG_ENDPOINT==FALSE)
    //! エンドポイント自動形態設定なしの場合、ここで使用者関数を割り当てます。
    #define                                 User_configure_endpoint()
#endif

//! @defgroup host_cst_actions USBホスト独自活動
//!
//! @{
// 各USBホスト事象と連携する活動をここに書いてください。
// 関数を邪魔しないで順番に時間を無駄にしないよう注意してください。
#define Usb_id_transition_action()
#define Host_device_disconnection_action()
#define Host_device_connection_action()
#define Host_sof_action()
#define Host_suspend_action()
#define Host_hwup_action()
#define Host_device_not_supported_action()
#define Host_device_class_not_supported_action()
#define Host_device_supported_action()
#define Host_device_error_action()
//! @}
```

6. 上位USB応用とのUSBソフトウェア ライブラリ使用

6.1. 装置応用

装置使用者応用は一旦ホストからのSET_CONFIGURATION要求が受信されるとTRUEを返す"Is_device_enumrated()"関数によって装置が正しく列挙(接続認識)されたことを知ります。

```
void device_template_task(void)
{
   //.. 装置列挙(接続認識)状況初回検査
   if (Is_device_enumerated())
   {
      //.. ここからUSB装置応用コード開始
   }
}
```

見本応用コードに含まれる"device_template_task.c"ファイルは形態設定記述子で宣言されたインターフェースと連携する大量(バルク)IN/OUTと割り込みの両方のエンドポイントの使用方法を図解します。

6.2. ホスト応用

ホスト使用者応用は以下を許す事象特有関数によってホスト9章ライブラリで通信します。

- 装置の接続/切断の検出
- 主な装置特性(VID、PID、クラス、補助クラス、規約、インターフェース数、最大電力供給など)の取得
- USBバスの休止/再開

USB 9章管理に加えて、ホスト ライブラリは大量(バルク)のINとOUTのデータの流れ管理を許す標準関数の組(ポーリング(塊化)形態または割り込み下の非塊化形態の両方での送信と受信の関数)も提供します。

6.2.1. 装置検出

6.2.1.1. 装置接続

新しい装置が列挙(接続認識)されてUSBホスト9章ライブラリで形態設定される(形態設定要求送信)時に、"Is_new_device_connection_event()"関数はTRUEを返します。

6.2.1.2. 装置切断

最後に形態設定された装置がUSBホスト ポートから正に切断される時に、"Is_device_disconnection_ event()"関数はTRUEを返します。

6.2.2. 形態設定された装置特性

ホスト ライブラリは形態設定された装置のUSB特性を得るための1組の関数を提供します。

6.2.2.1. 標準情報

- "Get_VID()":形態設定された装置のVIDを返します。
- "Get_PID()":形態設定された装置のPIDを返します。
- "Get_maxpower()":形態設定された装置に必要とされる最大電力(2mA単位)を返します。
- "Is_device_self_powered()":形態設定された装置が自己給電される時にTRUEを返します。
- "Is_device_supports_remote_wakeup()":形態設定された装置が遠隔起動機能を支援する時にTRUEを返します。
- "Is_host_full_speed()":形態設定された装置が全速(Full-speed)形態で接続された時にTRUEを返します。低速(Low-speed)装置への接続で、この関数はFALSEを返します。

6.2.2.2. インターフェース、エンドポイント

ホスト ライブラリは複数インターフェース宣言を持つ装置(複合装置)を支援し、受信する装置記述子に従ってホスト パイプを形態設定することができます。

形態設定された装置のインターフェース パラメータはインターフェース特性を含む構造体の配列に格納されます。

```
S_interface interface_supported[MAX_INTERFACE_SUPPORTED]
// 以下とで、
typedef struct
{
   U8  interface_nb;
   U8  altset_nb;
   U16 class;
   U16 subclass;
   U16 protocol;
   U8  nb_ep;
   U8  ep_addr[MAX_EP_PER_INTERFACE];
} S_interface;
```

ホスト ライブラリはこれらのパラメータをアクセスする1組の関数を提供します。

- "Get_nb_supported_interface()":接続した装置に関して形態設定された支援インターフェース数を返します。
- "Get_class(interface)":指定したインターフェースのクラスを返します。
- "Get_subclass(interface)":指定したインターフェースの補助クラスを返します。
- "Get_protocol(interface)":指定したインターフェースの規約を返します。
- "Get_nb_ep(s_interface)":指定したインターフェース番号と連携するエンドポイント数を返します。
- "Get_interface_number(s_interface)":指定した支援されるインターフェースに関する装置記述子インターフェース番号を返します。
- "Get_alts_s(s_interface)":指定したインターフェースに関する代替設定値の番号を返します。
- "Get_ep_addr(s_interface,n_ep)":指定したインターフェース番号とエンドポイント番号と連携する論理エンドポイント アドレスを返します。

ホスト ライブラリは"進行中の"ホスト パイプ形態設定を許します。受信した装置記述子に従って、ホスト ライブラリは各装置エンドポイントと連携するパイプを割り当てます。参照表("ep_table")は物理的なホスト パイプ番号を論理的なエンドポイント アドレスに繋ぎます。

```
U8 ep_table[MAX_EP_NB];
```

以下の関数は相互参照を得るのに使用することができます。

・"Get_ep_addr(s_interface,n_ep)"：指定したインターフェース番号とエンドポイント番号と連携する論理エンドポイント アドレスを返します。

・"host_get_hwd_pipe_hb(ep_addr)"：指定した論理エンドポイント アドレスと連携する物理的なパイプ番号を返します。

6.2.3. バス活動管理

6.2.3.1. USBバスの休止

使用者ホスト応用から呼び出されると、"Host_request_suspend()"関数はUSBバスを休止形態へ移行します。形態設定された装置が遠隔起動を支援するなら、USBホスト ライブラリは休止形態へ移行する前に"遠隔起動機能許可設定(SET Feature Enable Remote Wake Up)"要求を送ります。

"Is_host_supended()"関数はUSBが休止状態で且つUSB状態を検出するためにホスト上位応用から使用されるかもしれない時にTRUEを返します。

6.2.3.2. USBバスの再開

ホスト応用は"Host_request_resume()"関数を呼ぶことでUSB活動を再開することができます。

6.2.3.3. 遠隔起動

形態設定された装置が遠隔起動を支援するなら、この関数はUSBホストの活動再開を許します。

6.2.4. データ転送関数

ホスト ライブラリはポーリングまたは割り込みのどちらの形態でも大量(バルク)のINとOUTのパイプでデータの送受信を許す2つの形式の標準関数を提供します。

6.2.4.1. ポーリング転送関数

・送信関数

```
U8 host_send_data(U8 pipe, U16 nb_data, U8 *buf);
```

この関数は*bufで指示されたnb_dataを指定されたパイプ番号で送ります。

・受信関数

```
U8 host_get_data(U8 pipe, U16 *nb_data, U8 *buf);
```

この関数は*bufで指示されたnb_dataへ指定されたパイプ番号で受け取ります。処理のためのバイト数はパラメータとして渡され、従って関数が戻る時に、それは結果的に関数によって受信されたデータ数を含みます。

6.2.4.2. 非塊化転送関数

これらの関数を使用するには、ホスト形態設定領域で"USB_HOST_PIPE_INTERRUPT_TRANSFER"がENABLEとして定義されるべきです。これらの関数は直前の塊化のものと同様ですが、それらはデータ転送の最後を待つ活動なしで直ちに戻ります。データ転送が完了または異常発生時、ポインタ パラメータとして渡される呼び戻し関数が呼ばれ、呼び出し関数のパラメータとして状態と処理したバイト数を返します。

・送信関数

```
U8 host_send_data_interrupt(U8 pipe, U16 nb_data, U8 *buf, void  (*handle)(U8 status, U16 nb_byte));
```

この関数は*bufで指示されたnb_dataを指定された物理的なパイプ番号で送ります。

・受信関数

```
U8 host_get_data_interrupt(U8 pipe, U16 nb_data, U8 *buf, void  (*handle)(U8 status, U16 nb_byte));
```

この関数は*bufで指示されたnb_dataに指定されたパイプ番号で受け取ります。

6.2.4.3. 戻り値

通信関数は以下の状態値を返します。

```
#define PIPE_GOOD               0       // 転送OK
#define PIPE_DATA_TOGGLE        0x01    // データ交互誤り
#define PIPE_DATA_PID           0x02    // PID誤り
#define PIPE_PID                0x04
#define PIPE_TIMEOUT            0x08    // 時間超過異常 (ハンドシェーク受信なし)
#define PIPE_CRC16              0x10    // CRC誤り
#define PIPE_STALL              0x20    // STALLハンドシェーク受信
#define PIPE_NAK_TIMEOUT        0x40    // NAKハンドシェーク最大数受信
#define PIPE_DELAY_TIMEOUT      0x80    // 時間超過異常
```

6.2.5. 見本上位ホスト応用

ホスト応用作業から抽出された以下のコードはUSBホスト ソフトウェアを用いる代表的な上位ホスト応用管理を図解します。

```
void host_template_task(void)
{
   U16 *ptr_nb;
   U16 nb;
   U8 i;
   ptr_nb=&nb;
   // ホスト制御器が接続され列挙(接続認識)された装置Bでの全動作形態かの初回検査
   if(Is_host_ready())
   {
      // (装置接続後に1回だけ実行される)新規装置接続
      if(Is_new_device_connection_event())
      {
         for(i=0;i<Get_nb_supported_interface();i++)
         {
            // 2つの大量(バルク)IN/OUTパイプを持つ1つ目のインターフェース
            if(Get_class(i)==0x00 && Get_subclass(i)==0x00 && Get_protocol(i)==0x00)
            {
               // IN/OUTエンドポイントと連携する正しい物理的なパイプを取得
               if(Is_ep_addr_in(Get_ep_addr(i,0)))
               {  // IN/OUT連携パイプ一致
                  pipe_in=host_get_hwd_pipe_nb(Get_ep_addr(i,0));
                  pipe_out=host_get_hwd_pipe_nb(Get_ep_addr(i,1));
               }
               else
               {  // IN/OUT連携パイプ不一致、逆転
                  pipe_in=host_get_hwd_pipe_nb(Get_ep_addr(i,1));
                  pipe_out=host_get_hwd_pipe_nb(Get_ep_addr(i,0));
               }
            }
            // 割り込みINパイプPを持つ2つ目のインターフェース
            if(Get_class(i)==0x00 && Get_subclass(i)==0x55 && Get_protocol(i)==0xAA)
            {
               pipe_interrupt_in=host_get_hwd_pipe_nb(Get_ep_addr(i,0));
               Host_select_pipe(pipe_interrupt_in);
               Host_continuous_in_mode();
               Host_unfreeze_pipe();
            }
         }
      }
```

[次頁へ続く]

```
        // 1つ目のｲﾝﾀｰﾌｪｰｽ(大量IN/OUT)をﾎﾟｰﾘﾝｸﾞ形態で管理
        // 見本作業はﾊﾟｲﾌﾟnb2を通して64ﾊﾞｲﾄを送ります。
        sta=host_send_data(pipe_out,64,tab);
        // そしてﾊﾟｲﾌﾟnb1から64ﾊﾞｲﾄを受け取ります。
        *ptr_nb=64;
        sta=host_get_data(pipe_in,ptr_nb,tab);
        // 2つ目のｲﾝﾀｰﾌｪｰｽ管理(USB割り込みINﾊﾟｲﾌﾟ)
        Host_select_pipe(pipe_interrupt_in);
        if(Is_host_in_received())
        {
            if(Is_host_stall()==FALSE)
            {
                i=Host_read_byte();
                Host_read_byte();
            }
            Host_ack_in_received(); Host_send_in();
        }
        // ここはUSB休止にする応用の要求の例
        if(Is_hwb())
        {
            Host_request_suspend();
        }
    }
        // ここは再開要求の応用例
        if(Is_host_suspended() && Is_joy_select())
        {
            Host_request_resume();
        }

            // 装置切断
            if(Is_device_disconnection_event())
            {
                // 装置切断で実行されるｺｰﾄﾞをここに置いてください。
            }
        }
```

7. よくある質問

7.1. 装置動作形態

7.1.1. VIDとPIDを変更する方法は?

VID(供給者ID)とPID(製品ID)はﾎｽﾄによる装置の識別を許します。各製造業者は自身のVIDを持つべきで、これは全ての製品に対して同じです(これはUSB協会によって割り当てられます)。各製品は自身のPIDを持つべきです(これは製造業者によって割り当てられます)。

VIDとPIDの値はusb_descriptor.hで定義されます。それらを変更するには、その値を以下のように変更しなければなりません。

```
// USB装置記述子 (USB Device descriptor)
#define VENDOR_ID       0x03EB          // ATMELの供給者ID = $03EB
#define PRODUCT_ID      0x201C
```

7.1.2. どうしたら文字列記述子の値を変更できますか?

文字列記述子の値はusb_descriptor.hで定義されます。例えば製品名の値を変更するには、以下のように値を変更しなければなりません。

文字列値の長さ:

```
#define USB_PN_LENGTH          18
```

文字列値:

```
#define USB_PRODUCT_NAME ¥
{ Usb_unicode('A') ¥
 ,Usb_unicode('V') ¥
 ,Usb_unicode('R') ¥
 ,Usb_unicode(' ') ¥
 ,Usb_unicode('U') ¥
 ,Usb_unicode('S') ¥
 ,Usb_unicode('B') ¥
 ,Usb_unicode(' ') ¥
 ,Usb_unicode('M') ¥
 ,Usb_unicode('O') ¥
 ,Usb_unicode('U') ¥
 ,Usb_unicode('S') ¥
 ,Usb_unicode('E') ¥
 ,Usb_unicode(' ') ¥
 ,Usb_unicode('D') ¥
 ,Usb_unicode('E') ¥
 ,Usb_unicode('M') ¥
 ,Usb_unicode('O') ¥
}
```

7.1.3. どうすれば私の装置を自己給電またはﾊﾞｽ給電形態で形態設定できますか?

装置を自己給電またはﾊﾞｽ給電の形態に形態設定するためのﾊﾟﾗﾒｰﾀはusb_descriptor.hﾌｧｲﾙで定義されます。これより下は各形態の定義です。

ﾊﾞｽ給電形態:

```
// USB形態設定記述子 (USB Configuration descriptor)
#define CONF_ATTRIBUTES     USB_CONFIG_BUSPOWERED
```

自己給電形態:

```
// USB形態設定記述子 (USB Configuration descriptor)
#define CONF_ATTRIBUTES     USB_CONFIG_SELFPOWERED
```

7.1.4. どうしたら新しい記述子を追加できますか?

あなたの応用に新しい記述子を追加するには、以下の段階に従わなければなりません。

1. usb_descriptor.hファイルに於いて記述子パラメータの値とそれの構造体型を定義してください。
   例えばHID記述子とそれの構造体はusb_descriptor.hファイルで以下で示されるように定義されるべきです。

```
/* ____HID descriptor_____________________________________________*/
   #define HID                  0x21
   #define REPORT               0x22
   #define SET_REPORT           0x02
   #define HID_DESCRIPTOR       0x21
   #define HID_BDC              0x1001
   #define HID_COUNTRY_CODE     0x00
   #define HID_CLASS_DESC_NB    0x01
   #define HID_DESCRIPTOR_TYPE  0x22

/*_____ U S B   H I D   D E S C R I P T O R _____________________________*/
   typedef struct {
    U8  bLength;                /* この記述子のﾊﾞｲﾄでの量 */
    U8  bDescriptorType;        /* HID記述子形式 */
    U16 bscHID;                 /* 2進化10進仕様、公開 */
    U8  bCountryCode;           /* ﾊｰﾄﾞｳｪｱ目的対象国 */
    U8  bNumDescriptors;        /* 伴うHIDｸﾗｽ記述子数 */
    U8  bRDescriptorType;       /* 報告記述子形式 */
    U16 wDescriptorLength;      /* 報告記述費の合計長 */
   } S_usb_hid_descriptor;
```

2. usb_descriptor.hファイルに於いてs_usb_user_configuration_descriptor構造体のための記述子を追加してください。

```
typedef struct
{
 S_usb_configuration_descriptor  cfg_mouse;
 S_usb_interface_descriptor      ifc_mouse;
 S_usb_hid_descriptor            hid_mouse;
 S_usb_endpoint_descriptor       ep1_mouse;
} S_usb_user_configuration_descriptor;
```

3. usb_descriptor.Cファイルに於いて形態設定記述子のwTotalLengthパラメータに新しい記述子の量を追加し、記述子の値を追加してください(下の例をご覧ください)。

```
code S_usb_user_configuration_descriptor usb_conf_desc = {
 { sizeof(S_usb_configuration_descriptor)
 , CONFIGURATION_DESCRIPTOR
 , Usb_write_word_enum_struc(sizeof(S_usb_configuration_descriptor)¥
   +sizeof(S_usb_interface_descriptor)      ¥
   +sizeof(S_usb_hid_descriptor)            ¥
   +sizeof(S_usb_endpoint_descriptor)       ¥
   )
 , NB_INTERFACE
 , CONF_NB
 , CONF_INDEX
 , CONF_ATTRIBUTES
 , MAX_POWER
 }
 ,
 { sizeof(S_usb_interface_descriptor)
 , INTERFACE_DESCRIPTOR
 , INTERFACE_NB_MOUSE
 , ALTERNATE_MOUSE
 , NB_ENDPOINT_MOUSE
 , INTERFACE_CLASS_MOUSE
 , INTERFACE_SUB_CLASS_MOUSE
 , INTERFACE_PROTOCOL_MOUSE
 , INTERFACE_INDEX_MOUSE
 }
```

```
  }
  ,
  { sizeof(S_usb_hid_descriptor)
  , HID_DESCRIPTOR
  , HID_BDC
  , HID_COUNTRY_CODE
  , HID_CLASS_DESC_NB
  , HID_DESCRIPTOR_TYPE
  , Usb_write_word_enum_struc(sizeof(S_usb_hid_report_descriptor_mouse))
  }
  ,
  { sizeof(S_usb_endpoint_descriptor)
  , ENDPOINT_DESCRIPTOR
  , ENDPOINT_NB_1
  , EP_ATTRIBUTES_1
  , Usb_write_word_enum_struc(EP_SIZE_1)
  , EP_INTERVAL_1
  }
};
```

4. 新しい記述子の管理に関連する全ての関数の追加を忘れないでください。

7.1.5. どうしたら新しいエンドポイントを追加できますか?

新しいエンドポイントを追加するには、以下の段階に従ってください。

1. USB記述子項で説明されたように、エンドポイントはインターフェースに属します。新しいエンドポイントを追加するために最初に行うことはNB_ENDPOINTパラメータ値を1つ増やすことです。この値はusb_descriptor.hファイルで定義され、インターフェース記述子パラメータに属します。

```
// USBインターフェース記述子(Interface descriptor)
#define INTERFACE_NB          xx
#define ALTERNATE             xx
#define NB_ENDPOINT           xx                    // このパラメータ=インターフェースのエンドポイント数
#define INTERFACE_CLASS       xx
#define INTERFACE_SUB_CLASS   xx
#define INTERFACE_PROTOCOL    xx
#define INTERFACE_INDEX       xx
```

2. 次の段階はエンドポイント記述子の値を定義することです。これらの値はusb_descriptor.hで定義されなければなりません。

```
// USBエンドポイント1記述子(Endpoint 1 descriptor) FS
#define ENDPOINT_NB_1         (EP_MOUSE_IN | 0x80)
#define EP_ATTRIBUTES_1       0x03                  // 大量(バルク)=0x02,割り込み=0x03
#define EP_IN_LENGTH_1        8
#define EP_SIZE_1             EP_IN_LENGTH_1
#define EP_INTERVAL_1         0x02                  // ホストからの割り込みポーリング間隔
```

応用によって使用されるエンドポイント番号を指定するために、conf_usb.hでEP_MOUSE_INが定義されます。

3. 形態設定記述子にエンドポイント記述子を追加してください(直前のよくある質問点で説明されるように進めてください)。
4. usb_specific_request.cにハードウェア初期化呼び出しを追加してください。

```
void usb_user_endpoint_init(U8 conf_nb)
{
 usb_configure_endpoint(EP_MOUSE_IN,  ¥
    TYPE_INTERRUPT,                   ¥
    DIRECTION_IN,                     ¥
    SIZE_8,                           ¥
    ONE_BANK,                         ¥
    NYET_ENABLED);
} qsd
```

本社

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines
Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan
104-0033 東京都中央区
新川1-24-8
東熱新川ﾋﾞﾙ 9F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製造拠点

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3
France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR
Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn
Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-76-58-47-50
FAX (33) 4-76-58-47-60

文献請求
www.atmel.com/literature

本応用記述はATMELのAVR276応用記述(doc7675.pdf Rev.7675A-03/07)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。