第1章

キーボードやマウス，HDD，ハブなどを
標準周辺機器として使うための

クラス・ドライバとその基本動作

岡野 彰文

Windows や Linux では，USB にハブやキーボード，HDD を接続すればすぐに使用することができる．しかも OS に標準で用意されているドライバが読み込まれるため，別途ドライバを用意する必要がない．OS に標準でドライバが用意されているのは，これらのデバイスの仕様が USB 規格で標準化されているためである．　（編集部）

1 クラス・ドライバの役割

USB は，PC の世界では「標準インターフェース」と呼んでもよいほど普及しました．これまで周辺機器側の USB といえば，「ペリフェラル（スレーブとして機能する）側」の実装が多く紹介されてきましたが，最近では組み込み機器でも，その処理能力や使用できるリソースの増大と，組み込みシステムに特化した専用チップを用いることにより，USB ホスト機能を比較的容易に実装することが可能になってきました．

USB ペリフェラルとして供給されている機器は実に多様で，しかも低価格化が進んでいます．これらの機器は基本的には PC 用として供給されているものがほとんどですが，組み込みアプリケーションでもこのようなデバイス群を活用しない手はありません．「ホスト機能」を実装すれば，それらの USB 周辺機器が組み込みシステムからも簡単に使えるのです．

さらに進んで「USB は組み込み機器に適したインターフェースである」と言うこともできるでしょう．これは次のような例を USB 以前のインターフェースで考えると明らかです．

キーボード，マウス，ストレージを接続する機器を考えてみましょう．USB 以前ではこれを実現するには，それぞれの物理ポート，それをサポートするインターフェース・チップ，さらにそのハードウェアをドライブするドライバがそれぞれ必要でした（**図 1**）．使用者の側から見れば，これらの周辺機器を使用する際にはシステムを立ち上げる前に接続しておき，稼働中の取り外しや追加はできませんでした．

これを USB で実現すると，非常に簡単になります（**図 2**）．機器のパネルには，より単純な（1 種類だけの）物理ポートを用意

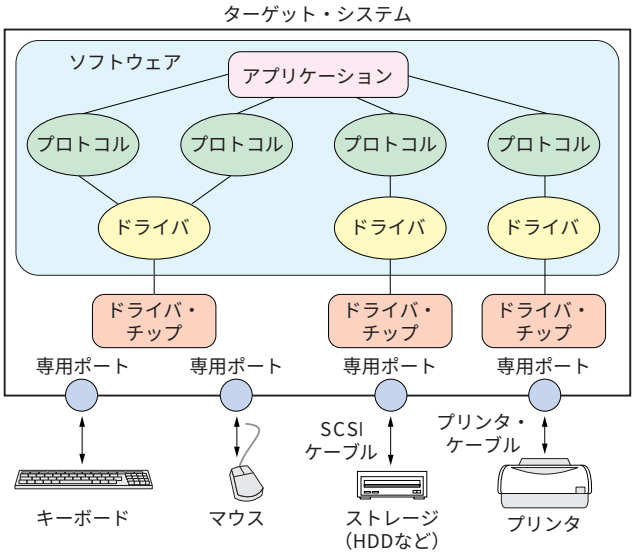


図 1　USB 以前の周辺機器接続

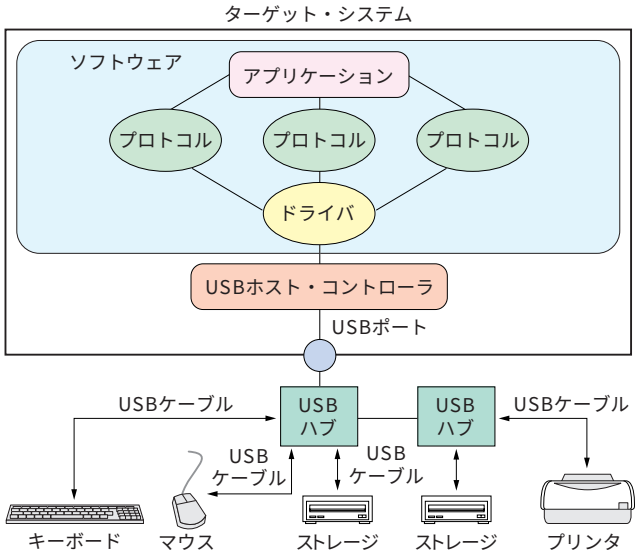
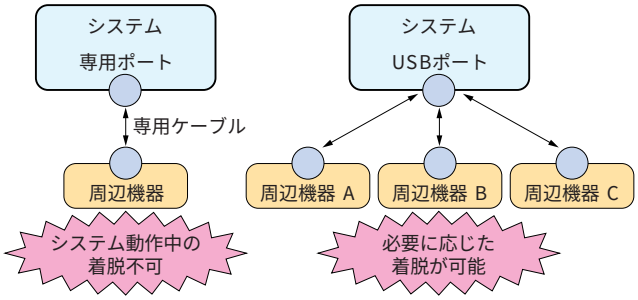


図 2　USB を用いた周辺機器接続

KEYWORD ―USB，USB ホスト機能，ホスト・ドライバ・スタック，標準ディスクリプタ，WASABI



(a) USB以前の周辺機器接続　(b) USBを使った周辺機器接続

図3　ホット・プラグ機能で必要なときだけ接続して使う

するだけで済み，システムが稼働している最中でも必要なときだけ接続して使うこともできます(**図3**).

もしこのような機器がUSBハブもサポートできれば，一度に接続できる周辺機器の数はポートの数に制限を受けることもなくなります.

USBホスト機能付き組み込みシステムの普及が急ペースで進んでいるのは，このような「低価格」，「物理的な制約が少ない」，「周辺機器の多様性」，「使いやすさ」という利点を考えると，当然の流れでしょう.

● USBを使うためのハードウェアとソフトウェア

USB登場当初は組み込みシステムでUSBホストを実装するのはたいへん困難でした．当時はPC用のホスト・チップしかないため，そのチップを使える環境が用意できなければどうにもなりませんでした.

90年代半ばからは組み込みシステム向けのUSBホスト・チップが供給され始め，状況は改善されてきました．現在では多くのUSBホスト・チップから，システムに求められるパフォーマンスやコストに応じてチップを選択できるようになっています.

もちろん，USBホスト機能を実現するにはハードウェアだけでなく，ソフトウェアも必要になります.

このソフトウェアは一般に「USBホスト・ドライバ・スタック(以下ホスト・スタック)」と呼ばれ，いくつかの層をなす構造を通してUSBホスト機能が実現されるように作られています.

● ホスト・スタックの構造

図4にホスト・スタックの一般的な構造を示します．大きく分けて三つの層からなっており，最下層から「Host Controller Driver(HCD)」，「Bus Driver(BD)」，「Class Driver(CD)」のように作られています.

▶一般的な実装と独自の実装を吸収するHCD(Host Controller Driver)

最下層のHCDは，ハードウェアで実現される「ホスト・コントローラ(チップ)」のドライバです．この層は上位の「バス・ドライバ」からハードウェアを抽象化するために設けられています.

実際にホスト・コントローラはいくつかの一般的な実装と，多数の独自の実装が存在します．これらの違いを吸収する層がHCDの目的です.

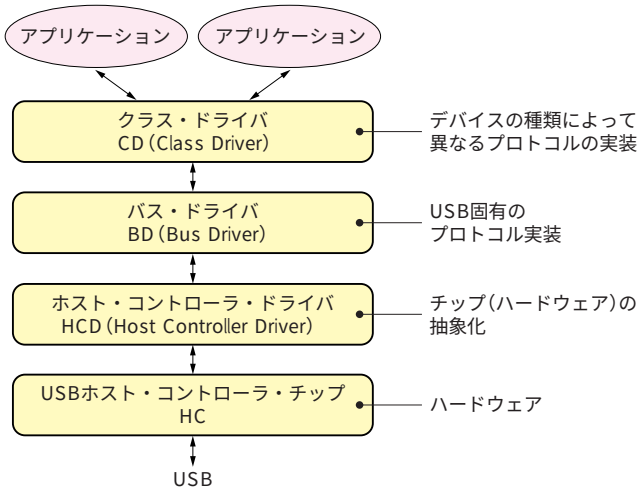


図4　ホスト・スタックの一般的な構造

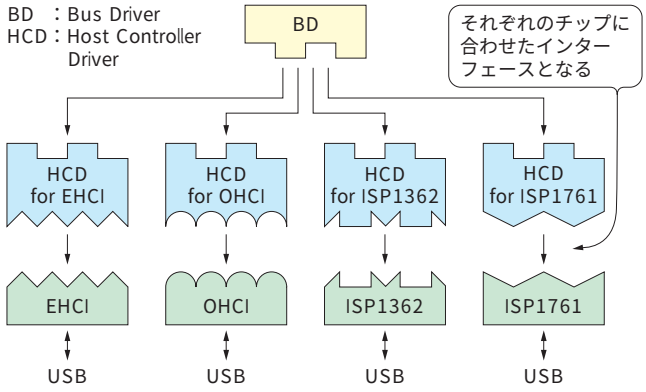


図5　HC(USBホスト・コントローラ・チップ)**とHCD**(Host Controller Driver)**の関係**

ホスト・コントローラの一般的なものにはUHCI(Universal Host Controller Interface)，OHCI(Open Host Controller Interface)，EHCI(Enhanced Host Controller Interface)と呼ばれる規格に則った実装があり，これらそれぞれの規格に準拠したドライバであれば，互換性が保証されます．PC用の汎用ホスト・コントローラのほとんどは，これらの規格に準拠したものが使われています(**図5**).

一方，これら以外の実装は各ベンダ独自のものであり，組み込み用途のチップがこれにあたります．組み込み用途のホスト・コントローラは実に多様で，OHCIやEHCIのようなフル機能のコントローラを内蔵したものから，機能を制限してハードウェアを簡略化した実装のものまで，それぞれ特徴のあるチップが供給されています.

もちろん，これらの独自の実装のチップにおいては，レジスタ・セットやバッファ管理，転送管理の方法は統一されていません.

HCDはこれらの違いを隠ぺいするために用意されます.

▶BD（Bus Driver）── USBバス管理の実装を上位層に対して隠ぺいする

BDはUSBとして基本的な動作を実装している部分です．デバイスの接続と転送一般の管理はすべてここが行います．具体的には各デバイスのUSBアドレスを管理し，その各デバイスのエンドポイントの状態を監視し，接続状態に変化があった場合にはそのときに必要な処理（Enumeration ；エニュメレーション）を提供します．

この層はUSBのバス管理の実装を上位層に対して隠ぺいする役割りをもっています．

▶CD（Class Driver）──エンドポイントとそれを用いるプロトコルの標準化

さて，その上位にクラス・ドライバが位置しています．これにはどのような役割りがあるのでしょうか．

先に説明したバス・ドライバは，上位層に向けてUSBに接続された各デバイスに対する通信を，USBアドレス，エンドポイント番号で抽象化してくれます（**図6**）．

バス・ドライバよりも上位のソフトウェアは，これらのUSBアドレス，エンドポイント番号を指定すればUSB上の通信が実現できます．

本来，USBの提供する機能としてはこれで十分なはずです．実際にUSBの規格で定義されるUSB自体はこのレベル止まりです（ただしハブは除く）．

しかし，このレベルでの定義のみで，その使い方の標準化がされていないとなると，少し不便です．USBの周辺機器はその機器ごとに独自のエンドポイントとプロトコルを使用することになり，それぞれに対応したドライバを用意しなくてはなりません．

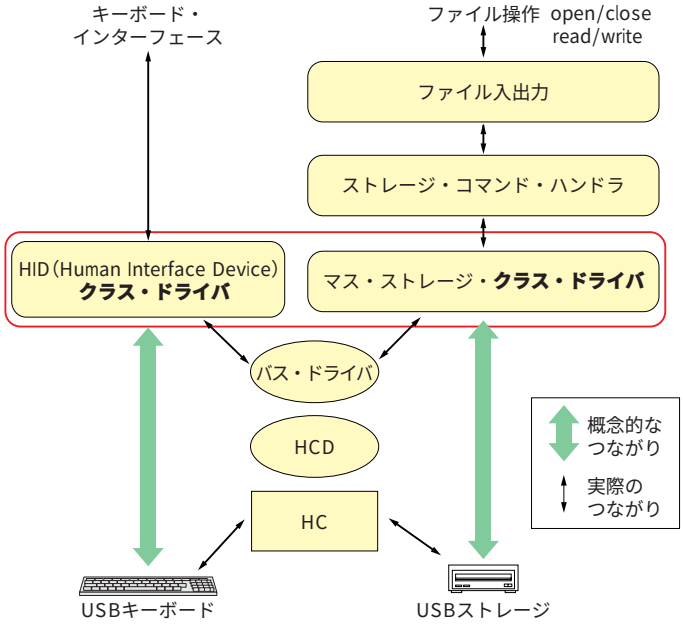


図6　クラス・ドライバの提供するもの

クラス・ドライバはこの「エンドポイントとそれを用いるプロトコルの標準化」を提供する層です．ハブやHIDやストレージといったUSBデバイスの種類をそれぞれまとめて「クラス」とし，その基本的な部分が定義されます．

このクラス・ドライバの仕様で定義されるものは強制ではなく，あくまでそのデバイスに汎用性をもたせるための選択肢の一つでしかありません．もしこのクラスの定義で不足があれば，ベンダ独自のクラス（ベンダ・スペシフィック・クラス）を実装することができます．

実際のアプリケーションでは，このクラス・ドライバの上にアプリケーション層が置かれることになります．あるいはアプリケーションとの間で仲立ちを行う，さらにほかのドライバやミドルウェアが存在するかもしれません．

たとえば，プリンタであれば個々のイメージをプリンタのコマンドに変換処理するためのドライバ層であったり，ストレージであればメディアに対するコマンドやファイル・システムを処理する層などがこれにあたります．

2 クラス・ドライバ共通の基本動作

● 標準ディスクリプタによるクラスの扱い

USBを扱うシステム（ホストを搭載する側）では，USBデバイスが接続されると自動的にそれを検知し，必要な設定を行います．このときホストは，接続されたデバイスからディスクリプタと呼ばれる情報を取得します．このディスクリプタを解釈することで，システムはそのデバイスに必要な動作を知ることができるようになっています．

デバイスの接続からUSBアドレスの割り当て，一連のディスクリプタ取得，デバイスの初期化までをエニュメレーション処理と呼び，クラス独自の処理もこの中で行われます〔USB規格の厳密な定義によるとUSBデバイスの取り外しに行われる処理もエニュメレーションとされている（USB規格4.6.3節）〕．

▶ホスト側システムの「クラス」の扱い

ここからは具体的に，ホスト側システムがどのように「クラス」を扱うのかを見ていきます．

デバイスがもつクラスは，標準ディスクリプタ「Standard DeviceDescriptor」（**表1**）のフィールド「bDeviceClass」によって識別できます（USB2.0規格，9.6.1節参照）．この1バイトのフィールドは0x01～0xFEの値をもつときUSB-IFにより定義されたクラスとして扱われます．もしこのフィールドが0xFFであれば，ベンダ・スペシフィック・クラスとして扱われます．ベンダ・スペシフィック・クラスとは，その名のとおり標準的なクラスではなく，ベンダによって独自に定義されたクラスです．これをドライブするには独自に実装されたクラス・ドライバが必要になります．

あるいは，もしこのフィールドが0x00であった場合，クラスを検出するにはもう一つのディスクリプタを取得しなければ

New Products ──松下電器，2GバイトのSDメモリーカード「RP-SDQ02GJ1A」を発売
松下電器産業（株）は，2GバイトのSDメモリーカード「RP-SDQ02GJ1A」を発売した．データ転送速度は最大で5Mバイト/s．オープン価格．月産10,000枚の予定．

表1　標準ディスクリプタ StandardDeviceDescriptor

オフセット（バイト）	フィールド	サイズ（バイト）	値	内　容
0	bLength	1	数値	このディスクリプタのサイズ「18」
1	bDescriptorType	1	定数	デバイス・ディスクリプタ型「1」
2	bcdUSB	2	BCD	準拠する USB 規格の Revision
4	bDeviceClass	1	Class	USB-IF が定義するクラス・コード
5	bDeviceSubClass	1	SubClass	USB-IF が定義するサブクラス・コード
6	bDeviceProtocol	1	Protocol	USB-IF が定義するプロトコル・コード
7	bMaxPacketSize0	1	数値	エンドポイント 0 のバッファ・サイズ
8	idVendor	2	ID	USB-IF が定義するベンダ ID
10	idProduct	2	ID	製造者が定義する製品 ID
12	bcdDevice	2	BCD	デバイス・リリース番号
14	iManufacturer	1	Index	ストリング・ディスクリプタ（製造者）のインデックス
15	iProduct	1	Index	ストリング・ディスクリプタ（製品）のインデックス
16	iSerialNumber	1	Index	ストリング・ディスクリプタ（シリアル番号）のインデックス
17	bNumConfigurations	1	数値	このデバイスのもつコンフィグレーションの数

表2　「インターフェース」を定義するディスクリプタ StandardInterfaceDescriptor

オフセット（バイト）	フィールド	サイズ（バイト）	値	内　容
0	bLength	1	数値	このディスクリプタのサイズ「9」
1	bDescriptorType	1	定数	インターフェース・ディスクリプタ型「4」
2	bInterfaceNumber	1	数値	インターフェース番号
3	bAlternateSetting	1	数値	オルタネート設定番号
4	bNumEndpoints	1	数値	エンドポイントの数
5	bInterfaceClass	1	Class	USB-IF が定義するクラス・コード
6	bInterfaceSubClass	1	SubClass	USB-IF が定義するサブクラス・コード
7	bInterfaceProtocol	1	Protocol	USB-IF が定義するプロトコル・コード
8	iInterface	1	Index	ストリング・ディスクリプタ（このインターフェース）のインデックス

なりません．これは「このデバイスはデバイスとしてクラスが定義はされてない」ことを示しており，さらにその先の「インターフェース」の定義を参照しなければなりません．

この「インターフェース」とは，デバイスを多義的に実装するためのしくみです．通常，デバイスはホストとの通信を行うエンドポイントを複数用意し，そのエンドポイントのセットを標準クラスの定義に従って使うように作ります．このエンドポイントのセットを定義するのが「インターフェース」で，インターフェースごとに各セットを用意することができます．各インターフェースには，それぞれをクラスとして定義しておくことができます．

▶ クラス検出手順

「インターフェース」を定義するディスクリプタは「Standard InterfaceDescriptor」（**表2**）です．このディスクリプタには「bInterfaceClass」という1バイトのフィールドによってクラスが示されます（USB2.0 規格，9.6.5 節参照）．この値も 0x01 ～ 0xFE の値をもつとき USB-IF により定義されたクラスとして，0xFF であればベンダ・スペシフィック・クラスとして扱われます．0x00 は予約されており，使うことはできません（**図7**）．

「インターフェース」は参照番号が与えられており，ホストが発行するコマンド（標準デバイス・リクエスト）「SET_INTERFACE」によってインターフェースが決定されます．このときホストは自分が選択した「インターフェース」に対応したクラス・ドライバをロードしておくことになります（**リスト1**，**表3**）．

さて，デバイスのディスクリプタで指定されたクラスに対応

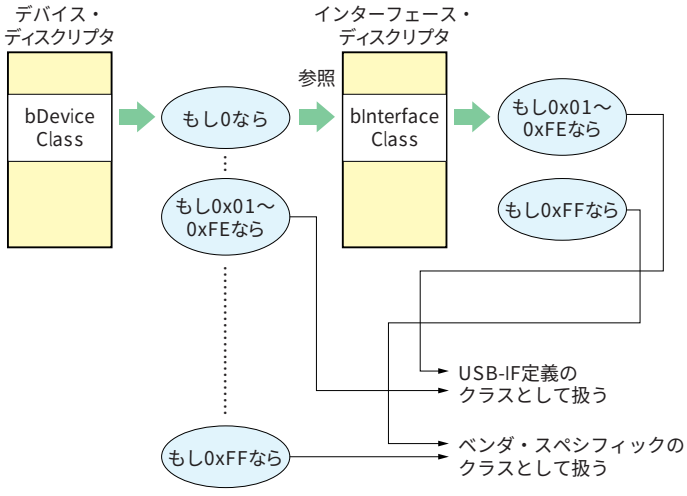


図7　クラス検出手順

リスト1　クラス検出のプログラム

```
unsigned short  (*g_if_class_initialization_method[ IF_CLASS_INITIALIZATION_METHOD_LIST_ITEM_NUMBER ])( device_instance *dvi_ptr )
                                                                =   {
                                                                        NULL, NULL, NULL, NULL,
                                                                        NULL, NULL, NULL, NULL,
                                                                        NULL, NULL, NULL
                                                                    }; //あとから各要素に，各関数へのポインタを代入

char            *gp_class_str[]                                 =   {
                                                                        "Reserved",
                                                                        "Audio",
                                                                        "Communication",
                                                                        "HumanInterface",
                                                                        "Monitor",
                                                                        "PysicalInterface",
                                                                        "Power",
                                                                        "Printer",
                                                                        "Storage",
                                                                        "Hub",
                                                                        "VenderSpecific"
                                                                    }; //各クラスの名前(表示用)

unsigned short clshndl_initialization_method( device_instance *dvi_ptr )
{
    unsigned char       dv_class_code;
    unsigned char       if_class_code;
    unsigned short      err;

    /*  クラス・コードの読み出し  */
    dv_class_code   = devep_get_device_class_ID( dvi_ptr );
    if_class_code   = devep_get_interface_class_ID( dvi_ptr );

    /*  想定範囲外の値は，強制的に 0 にする   */
    dv_class_code   = (dv_class_code < DV_CLASS_INITIALIZATION_METHOD_LIST_ITEM_NUMBER) ? dv_class_code : 0;
    if_class_code   = (if_class_code < IF_CLASS_INITIALIZATION_METHOD_LIST_ITEM_NUMBER) ? if_class_code : 0;

    if ( dv_class_code ) // クラス・コードが 0 でなければ...登録した関数をコール
    {
        if ( g_dv_class_initialization_method[ dv_class_code ] != NULL )
            if ( NULL != (err   = (*g_dv_class_initialization_method[ dv_class_code ])( dvi_ptr )) )
                return ( err );
    }
    else if ( if_class_code ) // クラス・コードが 0 でなければ...登録した関数をコール
    {
        /*  check interface class code  */

        if_class_code   = devep_get_interface_class_ID( dvi_ptr );

        if ( if_class_code < IF_CLASS_INITIALIZATION_METHOD_LIST_ITEM_NUMBER )
        {
            if ( g_if_class_initialization_method[ if_class_code ] != NULL )
                if ( NULL != (err   = (*g_if_class_initialization_method[ if_class_code ])( dvi_ptr )) )
                    return ( err );
        }
    }
    else // ベンダ・スペシフィック・クラスのドライバがインストールされていれば，それをコール
    {
        unsigned char   i;

        for ( i = 0; i < CUSTOM_CLASS_INITIALIZATION_METHOD_LIST_ITEM_MAX; i++ )
        {
            if ( cstm_slot[ i ].vid_pid )
                if ( cstm_slot[ i ].vid_pid == make_vid_pid( dvi_ptr ) )
                    if ( NULL != cstm_slot[ i ].class_init_method )
                        if ( NULL != (err   = (*cstm_slot[ i ].class_init_method)( dvi_ptr )) )
                            return ( err );
        }
    }

    return ( 0 );
}
```

したドライバをもっていないときはどうすればよいでしょうか．PCであればそのドライバのインストールを要求するメッセージをユーザに見せて対応を促すことになるでしょう．

組み込みシステムでは，このような「後からドライバをインストールする」ことは困難かもしれません．このような場合は非対応のメッセージをユーザに示してエニュメレーション処理を終了することになるでしょう．

● クラス・ドライバの動作

クラス・ドライバの主要部分は通常のドライバと同様に初期化/廃棄に関する処理と，動作中の主たる通信の処理で構成さ

Information ―― Microsoft 社，30 周年を迎える
「すべての家庭および机の上にコンピュータを」という目標を掲げて設立された米 Microsoft 社が 9 月 21 日に 30 周年を迎えた．9 月 23 日に米国ワシントン州シアトルのセーフコフィールドにて記念式典を開催した．

表3　各クラスに割り当てられたコード

クラス・コード	クラス名	クラス・コード	クラス名
1	Audio	5	PhysicalInterface
2	Communication	6	Power
3	HumanInterface Device（HID）	7	Printer
		8	Storage
4	Monitor	9	Hub

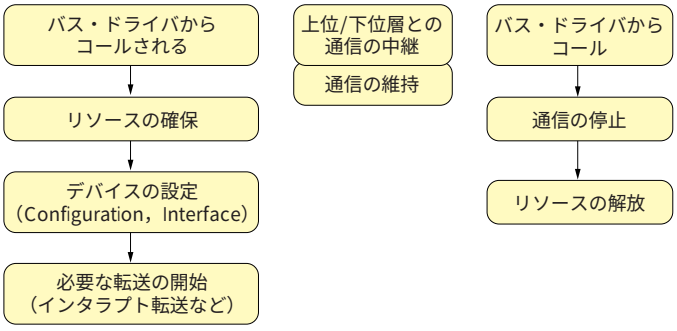


図8　クラス・ドライバの仕事

れます（**図8**）．

初期化/廃棄は，それぞれデバイスの接続/取り外しの際の処理です．動作中の主たる通信とはアプリケーションなどの上位層のソフトウェアからの通信要求を処理する部分になります．

▶ 初期化/廃棄に関する処理

クラス・ドライバの初期化/廃棄の処理は通常エニュメレーション処理に伴って行われます．デバイスが接続された際には，上述したディスクリプタによってクラスを識別し対応するドライバがロードされ，クラスの初期化処理が行われます．

エニュメレーション処理はバス・ドライバ（BD）が担当します．バス・ドライバは，接続されたデバイスのクラスを検出すると，対応するクラス・ドライバを検索しその中の初期化ルーチンを呼び出します．

クラス・ドライバの初期化処理は，そのクラス特有のリソースの確保や，デバイスを動作可能な状態に設定することなどを行います．インターフェースは複数用意されていることがあり，ホスト側のシステムはその中から目的や状況にあったものを選択し，設定します．

上記の「リソース」とは，各クラス/デバイスの属性を保存するためのメモリや，適切な転送を行うためのエンドポイントの設定，バッファの確保を指しています．さらには，転送の実行までを指すこととします．「転送の実行」までをリソースの意味に含める理由は，たとえばあるデバイスでは接続された状態においてはつねに転送を行わなくてはならないものがあり，この意味でシステムの資源を一定の割合で消費するという考え方によるものです．

このようなデバイスで継続的に行われる転送の開始は，初期化の時点で行われます．

このようなデバイスの代表的なものは，HID（キーボード，マウス）やハブです．接続されている間は，つねにインタラプト転送を用いてデバイスの状態をモニタするのですが，この転送の開始が初期化の処理として実装されます．

一方，これと反対にデバイスの接続を切り離した場合，この逆の処理を行う必要があります．データ転送中のエンドポイントがあればそれを中断，上位層のソフトウェアにレポートし，不要になったリソースを解放しなくてはなりません．これを行うルーチンが「廃棄に関する処理」となります．

▶ 主データの通信処理

初期化終了から廃棄処理開始までの接続中の通信処理を行う部分は各システムで独自の実装が行われます．上位のソフトウェア層に対するインターフェースと，下位のバス・ドライバ・レベルへのコールが用意されることになります．

上位のソフトウェア層に対しては，具体的なエンドポイントや転送の種類，データ・バッファの実装，さらにはクラス・レベルで転送を管理するタスクやスレッドを隠ぺいし，より簡単なインターフェースを提供することがおもな機能となります．

● 以降の解説について

以降の各章では，このクラス・ドライバについていくつかの一般的なクラスをとりあげ，その実現の方法と具体例を解説します．各具体例では，ホスト・ドライバ・スタックを『より簡単に実装』した「WASABI-Hot!」（以降「WASABI」と略す）でのコードを用いてその詳細を解説します．WASABIはPhilips社のフル・スピードOTGコントローラ・チップ「ISP1362」のデモとその使い方のサンプルを提供する目的で作られました．通常このコードはISP1362の評価キットの一部として配布されています．

WASABI自体については，その解説を行っている節を参照してください．また，参考文献(1)でも解説しています．

各クラス・ドライバは簡易的に実装されているため，より高い汎用性を求める場合には，各規格を詳細に検討した上での改造が必要になります．より正確な情報は下記のURLからダウンロードできる規格を参照してください．

```
http://www.usb.org/developers/devclass_docs
```

参考文献

(1) 岡野 彰文；ISP1362の概要とOn-The-Goサンプル・プログラムの詳解，Interface，2004年10月号，CQ出版社．

おかの・あきふみ
(株)フィリップスエレクトロニクスジャパン 半導体事業部

New Products ――東芝，HD DVD再生可能なノートPCを開発，来年発売
(株)東芝は，HD DVD再生ドライブを搭載したノートPCを開発した．来年の早い時期に発売を予定している．搭載されている光ディスク・ドライブの厚さは12.7mm．HD DVDの再生とDVD/CDの記録再生の光学系を一つの対物レンズで構成している．