第1章

USBの物理層からディスクリプタ，USBデバイスの初期化まで

これだけは知っておきたいUSBの基礎知識

桑野 雅彦

1 USBこそプラグ&プレイを実現したインターフェース

USB（Universal Serial Bus）は，パソコンに接続されている各種の周辺機器インターフェースを統合していくことを目的に作られたものです．あくまでも「パソコン対周辺機器」という関係であることから，周辺機器間での通信というものは配慮されていません．その代わり，通信制御のめんどうな部分をホスト側に極力移してしまうことで，ターゲット側のUSBインターフェース・ロジックは非常に単純で，低コストに抑えることができるようになっています．

また，最初からプラグ&プレイに配慮されていることも特徴の一つです．PCIカードなどのプラグ&プレイは，あくまでもパソコンの電源を切ってからカードを抜き差ししたときに自動的にハードウェアを認識して，リソースを割り振ってくれるという程度のもので，エンド・ユーザにとっては日頃それほどありがたみを感じるものではありません．

しかし，USBの場合には，パソコンを起動させたまま新しいデバイスをつなぎたくなったらその場でつなぎ，使い終わったらまた抜いておくということがごくあたりまえにできるのです．これこそまさに「プラグ」&「プレイ」と言えるのではないでしょうか．

規格が公表されてからしばらくの間は出足が鈍く，実用になるのかどうかが心配されたUSBでしたが，Windows 98以降は標準でサポートされるようになり，また少しずつではありますがハイ・スピード（480 Mbps）でも安定性が向上してきており，これからが楽しみなインターフェースです．また，USBデバイスといえば当初はキーボードやマウス程度しかありませんでしたが，最近は通信系のアダプタやMO，CD-ROMなどにもUSB対応の製品が出てくるなど，汎用インターフェースとしての広がりを見せています．

USB2.0は現在のUSB1.1との混在にも配慮されているので，将来USB2.0が主流になったとしても，USB 1.1準拠のデバイスが使用できなくなるということはありません．

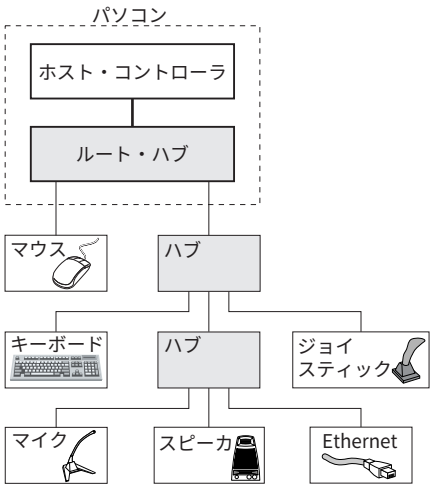


図1　USBの接続トポロジ

2 USBのアーキテクチャ

● USBの接続トポロジ

USBでは，一つのホスト・コントローラに一つのデバイスを1対1で接続することになっています（**図1**）．複数のデバイスを接続したい場合には，ホスト・コントローラの下にハブと呼ばれるデバイスを接続してポートを拡張することになります．

実際のパソコンでは，マザーボードからUSBポートが二つ出てきていますが，これはホスト・コントローラが二つあるわけではなく，チップセット内部にハブ（ルート・ハブ）が組み込まれている状態になって

いるのです(マイコンピュータのプロパティでデバイスマネージャを開くと，USBのところにルート・ハブが見える).

USBのハブがEthernetのハブなどと大きく異なるのは，このようにハブそのものがホストからはデバイスとして見えることです．つまり，ハブをつなぐとホストからはハブ・デバイスがUSBに接続されたとして認識されるのです.

前述したとおり，USBにはPnP(プラグ&プレイ)機能があり，コネクタの着脱によって自動的にデバイスの接続/切り離し処理が行われますが，デバイスが接続されたか否かは各ハブ(含ルート・ハブ)の部分で検出され，ホストがハブをチェックしにいったときにそれを検出することになります.

ハブを多段に接続することで，一つのUSBポートに多数のデバイスが接続できるようになるわけですが，無制限につなげられるわけではありません．USBでは，ハブは5段(ルート・ハブを除く)までという制約があります．また，USBでは各デバイスにアドレスを割り振りますが，アドレス・ビット長は7ビットなので，接続できるデバイスは最大127台となります.

● ホスト/ターゲット間の転送

USBの転送は，ホストがターゲットをポーリングし，それにターゲットが応答するという形で行われます．ただし，USBの場合，多数のターゲットが接続されるので，ポーリングに対する応答をターゲットのCPUのプログラムによって返していたのでは，遅すぎてバスが渋滞してしまいます．また，データ転送も同様で，データを受け取れたか否かをターゲット側のCPUが応答するまで待つわけにはいきません.

そこで，USBではエンドポイントという概念を持ち込み，ホストからのポーリングや転送動作に対する応答処理を，USBコントローラのレベルで行うようにしています.

エンドポイントは通常FIFOバッファ・メモリになっていて，ホストとターゲットの間のデータ転送はこのエンドポイントを使って行うというわけです．イメージとしては，**図2**のような状態になっていると思えばよいでしょう．ターゲットのCPUは，USBコントローラがエンドポイントFIFOに入れたデータを後からゆっくりと時間を気にせずに処理すればよいというわけです.

各エンドポイントには番号が付いていて，ホストは

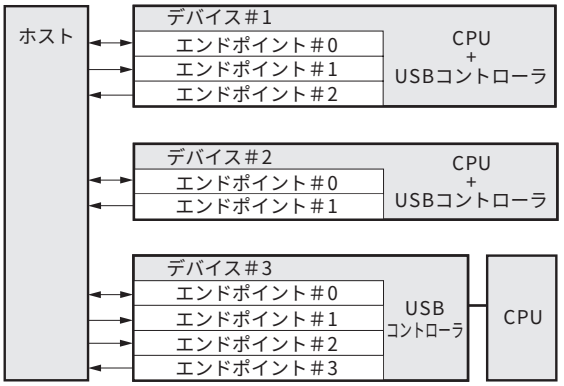


図2　USBによる接続イメージ

デバイスのUSBアドレスとエンドポイント番号を使ってアクセスにきます．USBを使ったホストとターゲット間のデータ転送は，基本的にこのエンドポイントを使ったホストとターゲット間のやりとりだけです．それ以外の，たとえばターゲットがホストに割り込みをかけたり，データを積極的にホストに送るということはできません.

ターゲットのエンドポイントのうち，エンドポイント番号0は特別なエンドポイントで，デバイスのコンフィグレーションや基本情報のやりとりなどに使用される双方向のエンドポイントです．これは，コントロール・エンドポイントと呼ばれます．コントロール・エンドポイントを使って行われる転送(コントロール転送)については，USBの規格書で内容およびデータのやりとりの手順が決められています.

これ以外のエンドポイントは単方向で，転送方向や用途，番号は各デバイスで自由に設定が可能です.

● エンドポイントの動作概略

ホストからターゲットに送られたデータは，ターゲット側のUSBコントローラによって，順次，自身のエンドポイントFIFOに収められます．ホストは起動時にターゲットからエンドポイントの情報を得ていますが，この中にエンドポイントのFIFOサイズも入っているので，データ・サイズがエンドポイントFIFOサイズより大きい場合は，ホストはエンドポイントのサイズ分以下に分割して転送動作を行います．このような，FIFOサイズ単位でのやりとりを行うことで，RS-232-CのXON/OFFやRS/CS制御のようなクリティカルなフロー制御を不要にしているのです.

ホストからのデータが正常に受け取れたか否かはUSBコントローラ・レベルで判断して，即座に応答

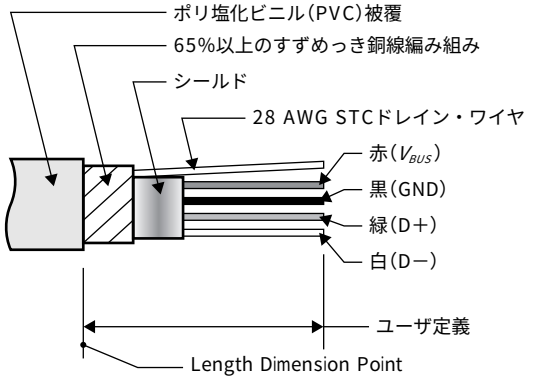


図3　USBケーブル
ケーブルについて，USB2.0の仕様で配色などが規定されている．標準の心線配色は，緑が差動信号ラインD＋，白がD－，赤が電源供給ラインV_{BUS}，黒がグラウンド(GND)，ワイヤ部がシールドとなっている

を返しますし，エンドポイントに入ったデータをターゲットのCPUが処理中であれば，ホストにNAKを返してリトライを要求するという動きをします．

逆に，ターゲットからホストへの転送の場合にも，USBコントローラがエンドポイントFIFOの状態を見ながら制御を行います．そして，エンドポイントFIFOにデータがあればそれを送り，なければホストからの転送要求にNAKを返すというぐあいです．

これによって，ホストには応答を即座に返しながらCPUの処理はある程度時間をかけて行うことが可能になるというわけです．

● 転送モード

USBは，以下のような四つの転送モードを持っています．転送はあくまでもエンドポイントを使用して行うので，言い換えれば，これらは各エンドポイントの動作モードということになります．

コントロール転送はすべてのUSBデバイスが処理できなくてはなりませんが，ほかの動作モードについては，どの動作モードのエンドポイントをいくつ実装するかは設計者の自由です．

▶バルク転送

バルク転送は，転送周期を問わず，まとまった量のデータを転送するために設けられたモードです．プリンタやスキャナ，MOやCD-ROMなどのデバイスのデータ転送用に使用されています．

▶インタラプト転送

インタラプト転送は，定周期で比較的転送頻度の低い少量のデータの転送を考えたモードといえます．ジョイスティックやマウス，キーボードなどのデータ転送や，プリンタの状態など各種ステータスの送信に使われます．

▶アイソクロナス転送

アイソクロナス転送は，音声データなどを送るときのように，データの正当性を保証しない代わりに，バンド幅を保証して定周期でのデータ転送ができるようにしたものです．

▶コントロール転送

コントロール転送は，ホストとターゲットの間でコントロール・エンドポイントを使って行われる制御用の転送です．デバイスの初期化や各種情報のやりとりに利用するので，USBデバイスは必ずコントロール転送をサポートしている必要があります．

3　USBの物理層

● USBのケーブル

USBの信号線は非常にシンプルで，1組の差動型のデータ・ラインと電源(V_{BUS})とグラウンド(GND)の計4本のワイヤとシールドしかありません(**図3**)．

データ線は双方向で利用されますが，USBの転送は半二重転送で，ホストからの要求に対してターゲットが応答するという形をとるため，パケットの衝突などは基本的に発生しません．データ・ラインは差動なので，通常はツイストペア・ケーブルを使用します．

使用されるケーブルは，データやシールドの引き出しにはAWG28が，電源にはAWG20～28を使用することになっています．USBでは，ケーブルから供給できる電流は最大500mAとなっています．これ以上の電流容量が必要な場合は，ターゲットにACアダプタなどを付けてセルフ・パワー・デバイスとして動作させる必要があります．

● USBコネクタ

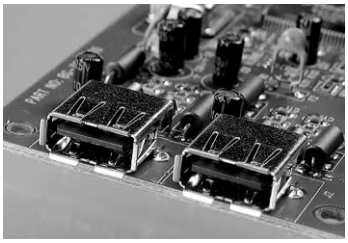
USBは単一のホストが全ターゲットのめんどうを見ることになっており，ホストを複数接続したり，ターゲットどうしを接続したりすることはできないので，ホスト側とターゲット側をまちがえてつなぐことがないよう，コネクタの形状が変えられています(**写真1**，**図4**，**図5**)．

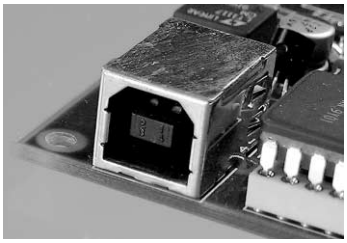
ホスト側のUSBコネクタは，平べったい長方形のような形状で，USBの四つのピン(信号と電源)が横に並んでいます．こちらを規格書ではシリーズAコネクタと呼んでいます．

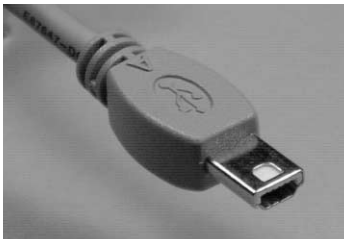
一方，ターゲット側は正方形に近いずんぐりした

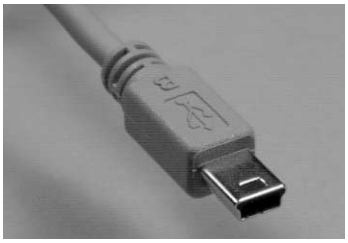

（**a**）標準Aプラグ

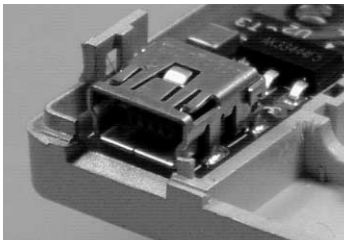

（**b**）標準Bプラグ


（**c**）標準Aレセクタプル


（**d**）標準Bレセクタプル


（**e**）ミニAプラグ


（**f**）ミニBプラグ


（**g**）ミニBレセクタプル

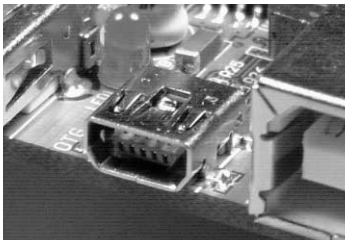
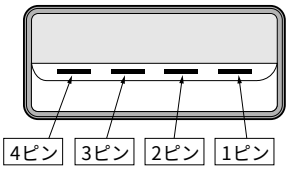
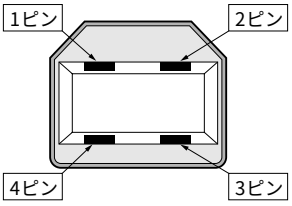

（**h**）ミニABレセクタプル

写真1
USBコネクタの外観



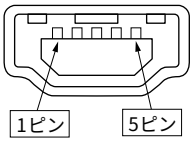
（**a**）標準Aプラグ



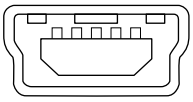
（**b**）標準Bプラグ

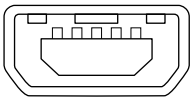
ピン番号	信号名
1	V_{BUS}
2	−Data(D−)
3	+Data(D+)
4	GND

（**f**）A/Bプラグ・ピン配置



（**c**）ミニAレセクタプル

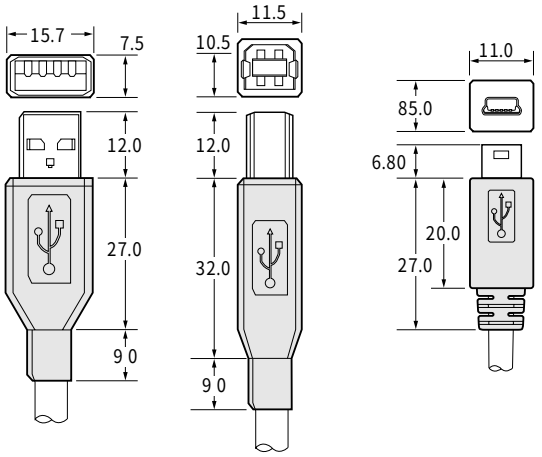

（**d**）ミニBレセクタプル


（**e**）ミニABレセクタプル

ピン番号	信号名
1	V_{BUS}
2	−Data(D−)
3	+Data(D+)
4	ID(NC)
5	GND

（**g**）ミニA/Bプラグ・ピン配置

図4　USBコネクタのピン配置



（**a**）シリーズAプラグ　（**b**）シリーズBプラグ　（**c**）シリーズミニBプラグ

図5　USBコネクタの寸法

形状で，上に2本，下に2本の接点があります．こちらはシリーズBコネクタと呼びます．

USBケーブルは必ず片方にシリーズA，反対側にシリーズBコネクタを取り付けてあるので，ホストどうし(両方ともシリーズA)やターゲットどうし(両方ともシリーズB)ではケーブルが差し込めないのです．

また，小型携帯機器向けに，よりサイズを小さくしたミニ・コネクタも定義されています．ミニ・コネクタにあるIDピンは，USB On-The-Go仕様で使う信号です．本書で解説するターゲット専用USB機器の場合は解放状態(NC)でかまいません．

また，いずれのコネクタも接点部分をよく見ると，電源と信号線とでは長さを変えてあることがわかります．これは，活栓挿抜(通電した状態でのコネクタの抜き差し)に対応したもので，挿入するときには電源ピンがまず接続され，その後で信号線が接続されます．逆に，抜くときには信号線がまず切断され，それから電源が切れるようにすることで，デバイスが破壊されないようにしているのです．

● USBのデータ信号定義

USBの信号は，D＋とD－を使った差動信号です．ホスト側(ないしハブのデバイス接続側)では両方のラインとも15kΩの抵抗でプルダウンされており，ターゲット側では転送速度に合わせて，フル・スピードまたはハイ・スピードならD＋側を，ロー・スピードならD－側を1.5kΩの抵抗でプルアップしています(**図6**)．ホストやハブは両方とも“L”レベルなら無接続，片側がプルアップされればデバイスが接続されたものと判定されます(両方ともプルアップすることは禁止)．

● USBのパケットとトランザクション

本来なら，次にUSBのパケットや各種トランザクションについて解説すべきところですが，実のところ，USBターゲット・コントローラを使って手軽にUSB周辺機器を作りたいと考えている人にとっては，この部分はUSBターゲット・コントローラが処理してくれるので，詳しく理解していなくても問題ありません．

とはいえ，USBシステムを正しく理解するには，パケットやフレーム，トランザクションについても理解する必要があります．参考文献(1)などを参照してください．

4　USBデバイスの論理的構造とディスクリプタ

● 複数の機能を内蔵したUSBデバイスも可能

たとえば一見するとUSBキーボードですが，キーボードの横にPS/2マウスのコネクタがあり，ここにPS/2マウスを挿し込むとUSBマウスの機能も増えるというUSBデバイスをよく見かけます．USBではこのように，実体としては一つのUSB機器の中に，複数の機能を内蔵することができます．

これを実現するため，そのUSBデバイスがどのよ

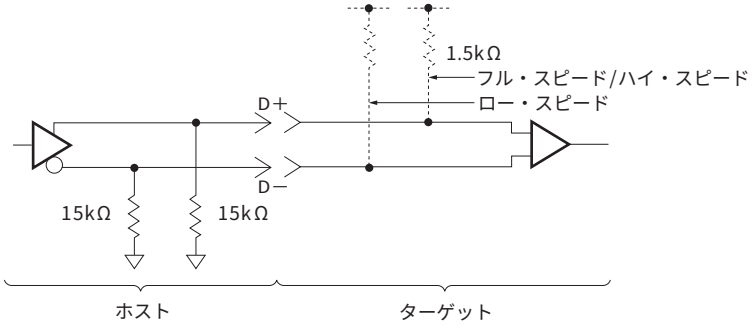


図6
USBのプルアップ/プルダウン

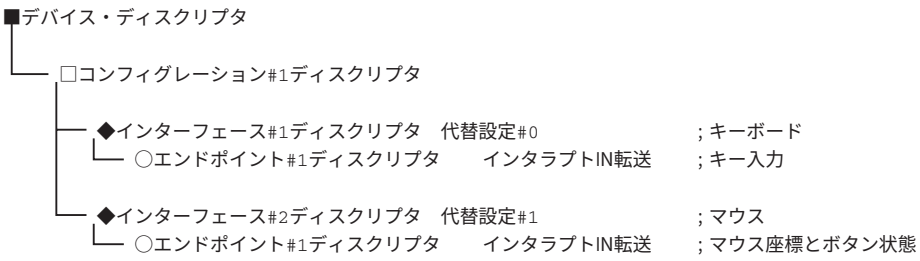


図7
USBデバイスの機能構成例

表1　デバイス・ディスクリプタ

フィールド名	サイズ(バイト)	説　明
bLength	1	ディスクリプタのサイズ(0x12で固定)
bDescriptor	1	ディスクリプタのタイプ(0x01で固定)
bcdUSB	2	BCD表現のUSB仕様リリース番号
bDeviceClass	1	クラス・コード 0：クラスなし，0xFF：ベンダ， 1～0xFE：特定
bDeviceSubClass	1	サブクラス・コード
bDeviceProtocol	1	プロトコル・コード 0　　：固有プロトコル使用せず 0xFF：ベンダ固有
bMaxPacketSize0	1	エンドポイント0の最大パケット・サイズ
idVendor	2	ベンダID(USB IFが割り当てる)
idProduct	2	プロダクトID (ベンダが割り当てる)
bcdDevice	2	BCD表現のデバイスのリリース番号
iManufacturer	1	製造者を表すストリング・ディスクリプタへのインデックス
iProduct	1	製品を表すストリング・ディスクリプタへのインデックス
iSerialNumber	1	デバイスの製造番号を表すストリング・ディスクリプタへのインデックス
bNumConfigurations	1	構成可能な数

うな仕様のものなのかを，データ・テーブルの形で階層的に記述します．これをディスクリプタと呼びます(**図7**)．

● ディスクリプタのいろいろ

▶デバイス・ディスクリプタ

これがもっともルートに位置するディスクリプタで，ベンダIDやプロダクトID，リリース番号，コンフィグレーションの数などを記述します(**表1**)．

▶コンフィグレーション・ディスクリプタ

このディスクリプタでは，後述するインターフェース・ディスクリプタとエンドポイント・ディスクリプタの長さや，USBデバイスの消費電力や電源供給方法などを列挙します(**表2**)．

表2　コンフィグレーション・ディスクリプタ

フィールド名	サイズ(バイト)	説　明
bLength	1	ディスクリプタのサイズ(0x09で固定)
bDescriptor	1	ディスクリプタのタイプ(0x02で固定)
wTotalLength	2	構成全体(構成，インターフェース，エンドポイント，その他のディスクリプタ)の長さ
bNumInterfaces	1	構成の持つインターフェースの数
bConfiguration Value	1	SetConfigurationで，この構成を選択するための引き数値(1以上)
iConfiguration	1	構成を表すストリング・ディスクリプタへのインデックス
bmAttributes	1	構成の特性．ビット単位で意味付け D7　　：“1” D6　　：自己電源 D5　　：リモート・ウェークアップ D4～D0：予約(0)
bMaxPower	1	最大バス電力消費量を2mA単位で指定

表3　インターフェース・ディスクリプタ

フィールド名	サイズ(バイト)	説　明
bLength	1	ディスクリプタのサイズ(0x09で固定)
bDescriptorType	1	ディスクリプタのタイプ(0x04で固定)
bInterfaceNumber	1	構成の中で，このインターフェースを表すインデックス番号(0ベース)
bAlternateSetting	1	SetInterfaceで，代替設定を選択するための引き数値
bNumEndpoints	1	(エンドポイント0を除く)インターフェースの持つエンドポイント数
bInterfaceClass	1	クラス・コード 0：クラスなし，0xFF：ベンダ，1～0xFE：特定
bInterfaceSubClass	1	サブクラス・コード
bInterfaceProtocol	1	プロトコル・コード 0　　：固有プロトコル使用せず 0xFF：ベンダ固有
iInterface	1	このインターフェースを表すストリング・ディスクリプタへのインデックス

表5　エンドポイント・ディスクリプタ

フィールド名	サイズ(バイト)	説　明
bLength	1	ディスクリプタのサイズ (0x07で固定)
bDescriptorType	1	ディスクリプタのタイプ (0x05で固定)
bEndpointAddress	1	エンドポイント・アドレス. ビット単位で意味付け D7　　：方向　0：OUT, 1：IN D6～D4：予約(0) D4～D0：エンドポイント番号
bmAttributes	1	属性(ビット単位で意味付け) D1～D0：転送タイプ 0：コントロール 1：アイソクロナス 2：バルク 3：割り込み D5～D2はアイソクロナス・エンドポイントのみで使用 D3～D2：同期タイプ 0：同期なし 1：非同期 2：アダプティブ 3：同期 D5～D4：ユーセージ・タイプ 0：データ・エンドポイント 1：フィードバック・エンドポイント 2：従属的なフィードバック・エンドポイント 3：(予約)
wMaxPacketSize	2	ペイロード・サイズ指定(ビットで意味付け) D10～D0：最大パケット・サイズ D12～D11：μフレームあたりの追加的なトランザクション数(HSのアイソクロナスと割り込みのみ) 0：追加なし(1トランザクション/μフレーム) 1：一つ(2トランザクション/μフレーム) 2：二つ(3トランザクション/μフレーム) 3：未使用(予約)
bInterval	1	データ転送のエンドポイントをポーリング間隔 FS/LS割り込み： ms単位(フレーム数)で指定 HSアイソクロナス/割り込み： μフレーム単位で$2^{(N-1)}$のNを指定(たとえば，bIntervalが4の場合，8μフレームに1回ポーリング) FSアイソクロナス： 1ms単位(フレーム数)で$2^{(N-1)}$のNを指定 HSバルク/コントロール： エンドポイントの最大NAKレートをμフレーム単位で指定．値0はOUT/DATAトランザクションでNAK応答しないことを意味

表4　デバイス・クラスの種類とクラス・コード

クラス仕様	デバイス・クラス・コード	インターフェース・クラス・コード
Firmware Update	—	FEh (サブクラス01h)
IrDA/USB Bridge	—	FEh (サブクラス02h)
Audio Interface	00h	01h
Communication Device	02h	—
CDC Control Interface	—	02h
CDC Data Interface	—	0Ah
HID	00h	03h
HUB	09h	09h
Mass Storage	00h	08h
Monitor	HIDと同じ	HIDと同じ
Power devices	HIDと同じ	HIDと同じ
Physical	—	05h
Printer	—	07h
ベンダ独自	—	FFh

表6　ストリング・ディスクリプタ

フィールド名	サイズ(バイト)	説　明
bLength	1	ディスクリプタのサイズ(N+2)
bDescriptorType	1	ディスクリプタのタイプ (0x03で固定)
bString	N	UNICODE文字列

▶インターフェース・ディスクリプタ

このディスクリプタで，インターフェースとしての仕様を記述します(**表3**)．先ほど説明したようなキーボードとマウスの機能が一つに内蔵されたUSBデバイスの場合は，インターフェース#1にキーボードとしての仕様を，インターフェース#2マウスとしての仕様を，インターフェース・ディスクリプタを分けて二つ記述します．インターフェースの種類は，**表4**に示すクラス・コードで示されます．

▶エンドポイント・ディスクリプタ

各エンドポイントのエンドポイント番号や転送モードの種類，最大パケット・サイズなどの仕様を記述します(**表5**)．

▶ストリング・ディスクリプタ

WindowsマシンにUSBデバイスを接続すると，まだドライバもインストールしていないのに，製品名が表示されることがあります．製品名などの文字列は，このディスクリプタで記述します．記述できるのはメーカ名や製品名だけでなく，複数の機能を内蔵したUSBデバイスの場合はそれぞれのコンフィグレーショ

表7　Device Qualitierディスクリプタ

フィールド名	サイズ（バイト）	説　　明
bLength	1	ディスクリプタのサイズ（0x0Aで固定）
bDescriptorType	1	ディスクリプタのタイプ（0x06で固定）
bcdUSB	2	BCD表現のUSB仕様リリース番号で，0x0200（USB2.0仕様）で固定
bDeviceClass	1	クラス・コード
bDeviceSubClass	1	サブクラス・コード
bDeviceProtocol	1	プロトコル・コード
bMaxPacketSize0	1	もう片方のスピードのエンドポイント0の最大パケット・サイズ
bNumConfigs	1	もう片方のスピードの構成可能な数
bReserved	1	リザーブ，0で固定

表8　Other Speed Configuratilnディスクリプタ

フィールド名	サイズ（バイト）	説　　明
bLength	1	ディスクリプタのサイズ（0x09で固定）
bDescriptorType	1	ディスクリプタのタイプ（0x07で固定）
wTotalLength	2	応答する全データのサイズ
bNumInterfaces	1	このスピードの構成でサポートするインターフェース数
bConfiguration Value	1	構成を選択するための値
iConfiguration	1	ストリング・ディスクリプタへのインデックス
bmAttibutes	1	構成ディスクリプタと同じ
bMaxPower	1	構成ディスクリプタと同じ

ンやインターフェースごとに文字列を格納できます（**表6**）．このディスクリプタは必須ではありません．

▶ **Device Qualitierディスクリプタ**

これはハイ・スピード対応デバイスで必要となるディスクリプタです．ハイ・スピード対応デバイスがフル・スピードで動作中にDevice Qualitierディスクリプタを要求されたら，ハイ・スピード時のデバイス・ディスクリプタを返します．逆にハイ・スピード動作中に要求された場合はフル・スピード時のデバイス・ディスクリプタを返します（**表7**）．ロー/フル・スピード・デバイスではこのディスクリプタは不要です．

▶ **Other Speed Configuratilnディスクリプタ**

これもハイ・スピード対応デバイスで必要となるディスクリプタです．ハイ・スピード対応デバイスがフル・スピードで動作中にDevice Qualitierディスクリプタを要求されたら，ハイ・スピード時のコンフィグレーション・ディスクリプタを返します．逆にハイ・スピード動作中に要求された場合はフル・スピード時のコンフィグレーション・ディスクリプタを返します（**表8**）．ロー/フル・スピード・デバイスではこのディスクリプタは不要です．

5　USBシステムの動作概要

● **USBデバイスの初期化手順**

図8にUSBデバイスの動作ステート図を示します．図中にある保留というのは，パワー・マネージメントなどでスリープ状態に入る場合のことです．とりあえずUSBデバイスを動かすのであれば，パワー・マネージメントについて考えなくても動作します．また，バス・パワーで動作するデバイスの場合は，USBケーブルを接続してはじめて電源が投入されます．

①の状態はリセット直後の状態です．この段階ではアドレスが割り当てられていないので，USBアドレスとしては0となります．ここでホストからさまざまなコマンドが送られてくるので，それに正しく応答しなければなりません．これをデバイス・リクエストと呼びます．デバイス・リクエストのフォーマットを**表**

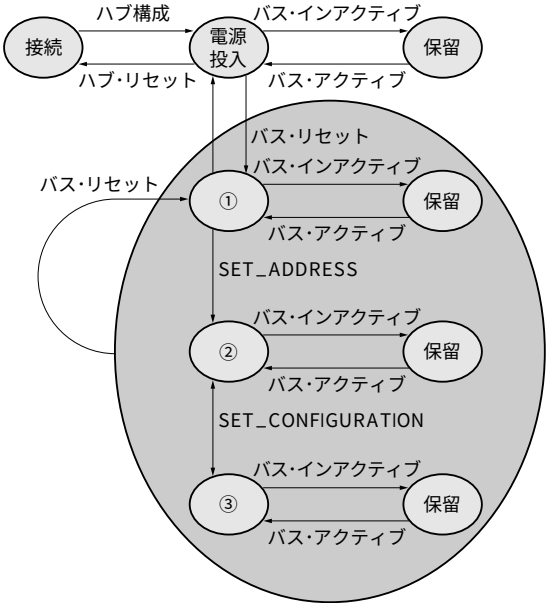


図8　USBデバイス・ステート

表9
デバイス・リクエストの引き数

フィールド名	サイズ（バイト）	説　明
bmRequestType	1	リクエストの特性．ビット単位で意味付け D7 ：データ転送方向 　0：ホスト→デバイス，1：デバイス→ホスト D6～D5：タイプ 　0：標準，1：クラス，2：ベンダ，3：予約 D4～D0：受信側 　0：デバイス，1：インターフェース， 　2：エンドポイント，3：そのほか，4以降：予約
bRequest	1	特定のリクエスト
wValue	2	bRequestが決める値
wIndex	2	bRequestが決めるインデックス， または，オフセット値
wLength	2	データ・ステージのバイト数

表10　標準的なデバイス・リクエスト

bmRequestType	bRequest	wValue	wIndex		wLength	Data	機能概要
00000000B 00000001B 00000010B	CLEAR_FEATURE(1)	DEVICE_REMOTE_WAEKUP(1)またはENDPOINT_HALT(0)	0 インターフェース・エンドポイント		0	なし	wValueで選択した機能をクリア
10000000B	GET_CONFIGURATION(8)	0	0		1	構成値	現在の構成を応答
10000000B	GET_DESCRIPTOR(6)	ディスクリプタ・タイプとインデックス値	0または言語ID		ディスクリプタ長	ディスクリプタ	wValueに指定されたディスクリプタを応答
10000001B	GET_INTERFACE(10)	0	インターフェース番号		1	代替設定値	設定されている代替設定値を応答
10000000B 10000001B 10000010B	GET_STATUS(0)	0	0 インターフェース・エンドポイント		2	指定したステータスのサイズ	デバイス・ステータスを応答
00000000B	SET_ADDRESS(5)	デバイス・アドレス	0		0	なし	wValueの値をデバイスアドレスとして設定
00000000B	SET_CONFIGURATION(9)	コンフィグレーション値	0		0	なし	wValueに指定された構成に切り替える
00000000B	SET_DESCRIPTOR(7)	ディスクリプタ・タイプとインデックス値	0または言語ID		ディスクリプタ長	ディスクリプタ	指定したディスクリプタを更新（オプショナルなリクエスト）
00000000B 00000001B 00000010B	SET_FEATURE(3)	DEVICE_REMOTE_WAEKUP(1)，ENDPOINT_HALT(0)，またはTEST_MODE(2)	0 インターフェース・エンドポイント	テスト・セレクタ	0	なし	wValueで選択した機能を有効化
00000001B	SET_INTERFACE(11)	代替設定値	インターフェース		0	なし	代替設定を切り替える
10000010B	SYNCH_FRAME(12)	0	エンドポイント		2	フレーム番号	アイソクロナス転送同期化フレーム

9に，標準的なデバイス・リクエストを**表10**に示します．

①の段階で発行されるのは，接続されたUSBデバイスがどんな素性のものかを判定するためのGET_DISCRIPTORコマンドです．ただしwValueの値により要求されるディスクリプタが異なるので，たとえばwValueの値が01hならデバイス・ディスクリプタを，05hならエンドポイント・ディスクリプタを返す必要があります．またOSによっては，USBバス・リセットも何度か発生します．

こうしてOSがUSBデバイスの素性を理解すると，SET_ADDRESSコマンドでUSBバス上のUSBアドレスが割り当てられ，USBデバイスは②の状態に移行します．そしてさらにホストからSET_CONFIGURATIONコマンドが発行されると，USBデバイスは③の状態に移行し，ここでUSBデバイスの初期化が終了します．

また，いずれの状態からでも，バス・リセットが発

生した場合は，①の状態に移行します．

なお，セルフ・パワーで動作するデバイスは，USBコネクタは非接続状態で電源が先に入る場合がありますが，USBデバイスの動作としてはUSBケーブルが接続されるまでは，停止状態で待ち続けます．

● デバイス・リクエストの処理

表10のデバイス・リクエストは，実際にはエンドポイント0に対して送られてきます．よってUSBターゲットでは，エンドポイント0の受信割り込みなどでコマンドの受信を検出し，エンドポイント0の受信バッファから受信データを読み出し，**表9**と**表10**に従ってコマンドの内容を解釈して，要求されているデータをホストに返します．ホストに返す場合も，エンドポイント0の送信バッファを使います．

なお，たとえばデバイス・ディスクリプタの長さは(現在の仕様では)実質的に12hバイト固定ですが，ホストからは先頭の数バイトしか要求してこないかもしれません．ホストが要求しているバイト数だけ返すように注意してください．

● USBデバイスのデータ転送

次に，USBターゲットの初期化が完了した後によく使うデータ転送について説明します．

▶バルクOUT転送

バルクOUT転送はホストからターゲットに対するデータ転送です．指定したUSBアドレスの指定したエンドポイントにホストからのデータが格納されます．たいていのUSBコントローラでは，ホストからのデータをすべて受け取ったところで，エンドポイントxの受信割り込みが発生します．

もし，エンドポイントにすでにデータが入っていた場合は，USBターゲット・コントローラはホストがOUTパケットに続いてデータ・パケットを送った後にホストに対してNAKを返します．また送られてきたデータは破棄します．NAKを受け取ったホストはしばらく待った後にデータを再送します．USBターゲットはバルクOUT転送によるデータを受け取ったら，すみやかにデータを取り出すようにすべきです．

▶バルクIN転送

バルクIN転送は，ターゲットからホストに対するデータ転送です．とはいえ，ホストが要求していないのに，いきなりターゲットがホストに対してバルクIN転送のデータを送り付けることはできません．USBシステムは，あくまでホストの要求に対してターゲットが応えるというアーキテクチャだからです．

そのため実際のUSBターゲットは，あらかじめエンドポイントxへホストに転送するデータをセットしておきます．USBターゲット・コントローラはホストからエンドポイントxに対するバルクIN転送の要求を受信すると，エンドポイントxにセットされたデータを送信します．送信が完了すると，たいていのUSBターゲット・コントローラは，エンドポイントxの送信完了割り込みを発生します．

もしエンドポイントxに送信すべきデータがない場合は，USBターゲット・コントローラはホストに対してNAKを返します．NAKを受け取ったホストは，しばらく待った後で再度バルクIN要求を出します．

問題は，どの段階でホストに送信するデータをエンドポイントxにセットするかです．

たとえば，別のエンドポイントyをバルクOUT転送に設定し，このエンドポイントを使ってターゲットに対して「次にバルクIN転送でzバイトのデータを要求するぞ」という意味のコマンドをあらかじめ送るようにします．またはコントロール転送のデバイス・リクエストを拡張して，次にバルクIN転送を発行する意味の独自のコマンドを発行することもできます．

それを受け取ったターゲットは，指定されたバイト数だけエンドポイントxにzバイトのデータを設定しておくことで，ホストからのバルクIN転送にすみやかに応答することができます．

▶インタラプトIN転送

指定した時間間隔で定期的に発生するバルクIN転送(ターゲットからホスト方向)と考えることができます．ただし先ほどのバルクIN転送と異なるのは，あらかじめディスクリプタにms単位(ハイ・スピードの時はマイクロ・フレーム単位)の時間を記述しているので，その時間以内にインタラプトIN転送が発行されることがわかっている点です．

インタラプトIN転送の場合，エンドポイントのデータがホストに送られると送信完了割り込みが発生します．そこでこの割り込みで次に送るべきデータを用意して，エンドポイントに設定するという処理の流れにするのが一般的でしょう．

参考文献

(1) USBハード&ソフト開発のすべて，TECH I Vol.8，CQ出版社．

くわの・まさひこ　パステルマジック