2001/6/9

ＵＳＢの解説

１．USB の基本アーキテクチャ

２．USB の通信プロトコル

３．USB のプラグ&プレイ

４．デバイスのハードウェア構成

５．デバイスのソフトウェア構成

６．汎用 USB ドライバの使い方

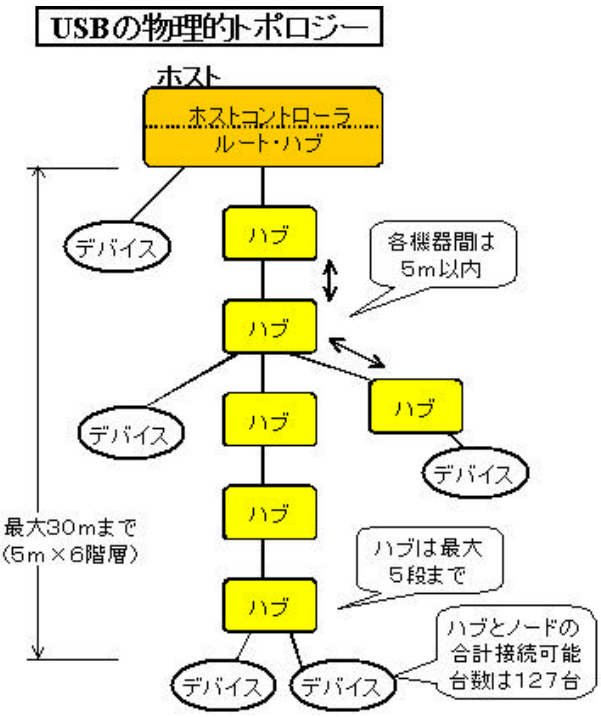
１．USB の基本アーキテクチャ

【物理的なアーキテクチャ】

USB での物理的な構成は下図のようになっていて、１台のパソコンが親機となって、ポーリング（問い合わせ）によって子機側と通信をします。つまりすべての通信の制御を親機が制御し、子機側から勝手に通信を開始することは出来ないようになっています。

このような構成図をネットワークとして見て、全体構成を**トポロジー**（Topology）と呼び、親機を**ホスト**（Host）、子機を**ノード**(Node)と呼んでいます。

実際の使用状態では、ノードには「**デバイス**」と「**ハブデバイス**」（ハブと呼ぶ）とがあります。右図のように、ホストと呼ばれる中身は、処理装置となるホストコントローラと、最初の USB バスの接続部となるルートハブが一体化されたものとなっています。そして実際のモノとしては、これがパソコン本体になります。



このルートハブに直接ノードとなるデバイスを接続するこも出来ますし、さらにデバイスの台数が多いときには、ハブを挿入して台数を増やすことが出来ます。

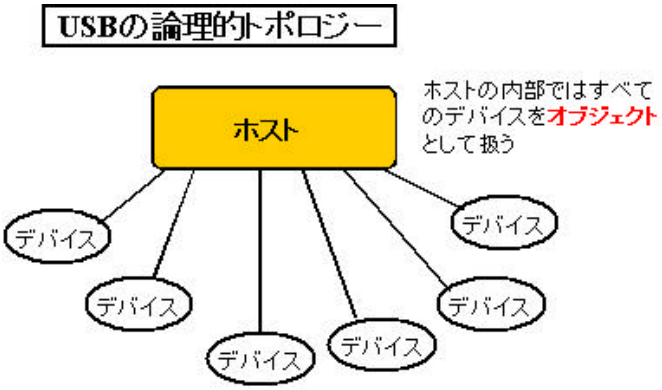
市販されているハブは、写真のような物で、４台から８台程度のデバイスを接続することが可能になっています。しかし、ハブの従属接続は、全体で５段までと制限されています。ホストの内部ではハブとデバイスすべてにアドレスが付与されて管理されます。このアドレスが最大１２７個なので、USB に接続可能なデバイスは最大１２７台ということになります。（ルートハブはアドレスが不要なので含まれません）

【論理構成とデバイスクラス】

上記は USB の物理的な構成ですが、ホストの内部でソフトウェアから見た論理的な構成は、右図のような単純な構成になっています。

ホストの中では、各デバイス（ノードとハブ両方を含む）を、**オブジェクト**として扱っています。従って、オブジェクトとなる各デバイスは、「**クラスの概念**」を持っており、それぞれに「**デバイスクラス**」を持っています。　基本のデバイスクラスはひとつで、いろいろな種類のデバイスはその基本のデバイスクラスの派生クラスとなっています。

このオブジェクト指向の概念を使っているため、見かけ上、ホストからデバイスへのアクセスはオブジェクトへのアクセスという形となります。つまり、「**メッセージ**」のやり取りでデータの送受が実行されます。具体的には、get・・・() と set・・・( ) コマンドだけでデータの送受を行うことになります。



そしてそのメッセージの処理を実行する「**メソッド**」の部分がデバイスの中のファームウェアの処理で実現されることになるわけです。従って、デバイス側はメッセージに対応したメソッド部分だけを実装すれば、全体の制御はホストが実行してくれるので、デバイスは簡単な構成で処理を実現することが可能となります。

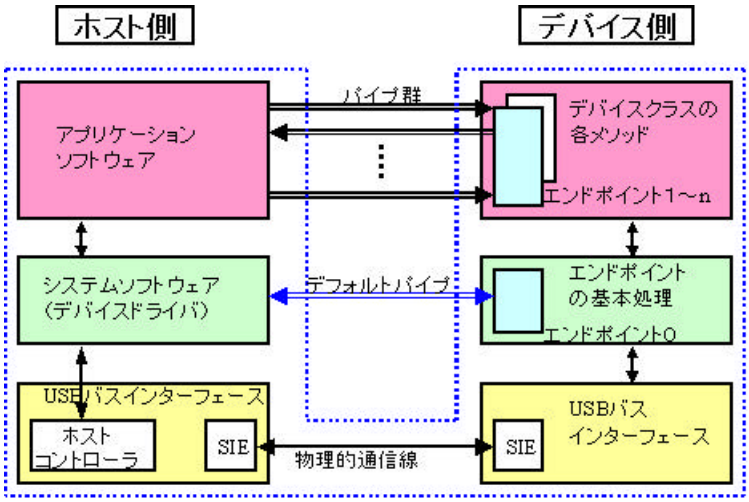
【通信のアーキテクチャ】

USB での通信の全体構成は下図で表すことが出来ます。

(1) **物理レベルの通信**

まず、物理的な通信は Wire つまり実際のケーブル接続で行われます。この時に実際に物理レベルでの通信を実行するのは、USB コントローラの IC の役割となっていますので、使う側からはまったく物理的な通信については知らなくとも USB を使うことが可能になるようになっています。

この１本のケーブル上で通信されるので、瞬間的には１個のデータの送受信しか出来ない訳ですが、タイムシェアをして見かけ上、いくつかの通信線でつながっているような使い方をします。



SIE: Serial Interface Engine

(2) **システムレベルの通信**

次のレベルではホストのシステムソフトウェア（Windows のデバイスドライバレベル）での通信で、これは必須の通信となりますので、ホストとデバイス間には「**デフォルトパイプ**」という通信の論理接続を用意します。そしてこのデフォルトパイプを使って行う通信を、「**コントロール転送**」と呼んでいます。

このコントロール転送により、リセットで初期化されたときに、システム設定を初期化したり、USB ケーブルが接続された時に、各種の設定制御をするために通信ができるようにします。つまり、このコントロール転送を使って、「**コンフィギュレーション**」を行い、デバイスの使い方の設定情報をホストとデバイス間でやりとりします。

これで使用するパイプの本数や、転送モードなどの設定情報をホストがデバイスに要求し、それを元にしてホストがデバイスの使用条件を設定します。

(3) **アプリケーションレベルの通信**

次のレベルはアプリケーションのレベルで、この間の論理的な通信線を「**パイプ**」と呼び、必要に応じて複数のパイプを設定することが出来ます。このパイプはあくまでも論理的な通信線ですから、実際の通信は１本の USB の線上で時分割で行われることになります。

このパイプを使った通信は、ホストから転送する「**OUT**」と、デバイス側から転送する「**IN**」のどちらか片方向だけの通信となっています。これに対し、コントロール転送を行うデフォルトパイプは双方向通信となっています。

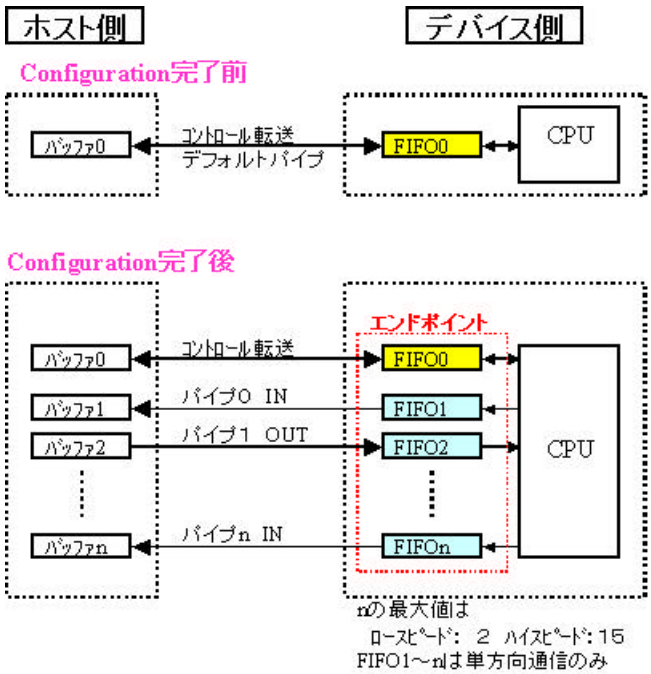
【エンドポイントとパイプ】

USB での論理的な通信はエンドポイントとパイプの概念で表現されます。

これを図で表すと下図のようになっています。

つまり、デバイス側には「エンドポイント」と呼ばれる「FIFO バッファ」が通信のための実体となり、ホスト側にも同様のバッファが用意されて、このバッファ同士が「パイプ」で接続されて、データを送受することになります。

まず、デバイスを USB に接続した直後の状態は、コンフィギュレーション前の状態となっていて、デフォルトパイプであるコントロール転送だけが通信のできる状態となっています。これに対応するのが「エンドポイント０（FIFO0）」となります



このコントロール転送を使って、コンフィギュレーションを実行し、デバイスの使い方が設定されたあとは、下側のように、コントロール転送以外に、あらたなエンドポイントが追加され、パイプが構成されます。このパイプは、片方向の通信しか出来ないようになっています。このエンドポイントの番号はホストが指定することが出来ます。

IN/OUT の方向はホストを中心に考え、ホストへの入力を IN、ホストからの出力を OUT と定義しています。またパイプの番号とエンドポイントの番号とは独立で、ホストが決めることが出来ます。

【USB の転送モード】

上記のパイプを使った USB の通信の仕方には、「バルク転送」、「インタラプト転送」、「アイソクロナス転送」があります。これにデフォルトパイプの「コントロール転送」を加えて、全部で４つの転送方法があり、それぞれの特徴は下記のようになっています。

（USB　Ver1.1 仕様）

項　目	コントロール転送	バルク転送	インタラプト転送	アイソクロナス転送
特　徴	少ないデータ量の 半二重通信	大容量データの 一括高信頼転送	小容量データの 定周期転送	一定時間内のデータ 量が保証された転送
用　途	セットアップ、設定 パラメータ転送用	記憶装置。スキャナ などの大容量高速 データの転送	計測やマンマシン 機器のデータ転送	音声などのリアル タイムな転送
転送速度	1.5Mbps/12Mbps	12Mbs	1.5Mbps/12Mbps	12Mbps
転送周期	不定	不定	Nmsec(N=1～255)	1ms/フレーム
データ量 /ﾊﾟｹｯﾄ	1～64 バイト(フル) 1～8 バイト(ロー)	8/16/32/64 バイト	1～64 バイト(フル) 1～8 バイト（ロー)	1～1023 バイト
信頼性 転送速度 遅延時間	－(再送あり) － －	◎(再送あり) ◎ △	◎(再送あり) ○ ◎	△(再送なし) ◎ ◎

この転送モードのどれを使うかは、デバイス側がコンフィギュレーション用のデータとして持っていて、USB を接続した時にコントロール転送を使ってホストにその情報を渡し、ホストで確認したあと、ホスト側から設定されてからモードが確定します。

この時ホスト側から、デバイスのアドレス付けと、パイプ番号とエンドポイント番号の割り付けが行われます。

２．USB の通信プロトコル

【USB 仕様と転送速度】

USB には VER1.1 と Ver2 により転送速度が異なっており、下記の３種類となります。Ver1.1 で用意された１．５Mbps と１２Mbps に対し、Ver2.0 で、４８０Mbps という非常に高速なモードが追加されたことになります。

USBの転送速度

出展:トランジスタ技術2000/6

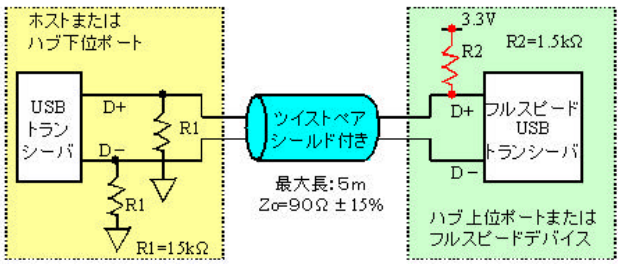
項　目		ハイスピード USB2.0	フルスピード USB1.1	ロースピード USB1.1
転送速度(Mbps)		480	12	1.5
ケーブル		フルスピードと同じ	ツイストペア シールド必要	ツイストペア不要 シールド不要
パケット サイズ (バイト)	コントロール転送	1～64	1～64	1～8
	インタラプト転送	1～3072	1～64	1～8
	バルク転送	1～512	1～64	－－
	アイソクロナス転送	1～3070	1～1023	－－

【物理的プロトコル】

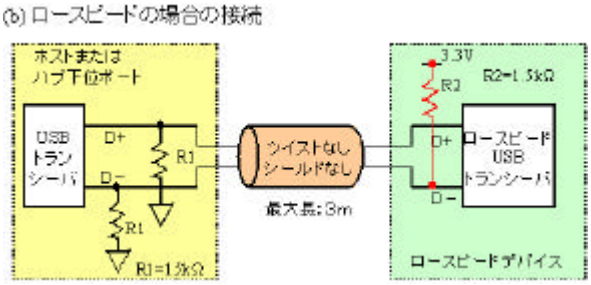
USB のケーブルに流れている信号の電気的な仕様は下記のようになっています。つまり、基本的には、D＋、D－の２本の信号線があり、それ以外に電源とグランドの都合４本の線が接続されます。これにあとシールドが加えられます。D+、D－の２本の信号線は下図のように接続されます。デバイス側でプルアップをどちら側にするかによってロースピードとフルスピードを区別するようになっています。

USBの電気的結線と転送速度検出方法

(a) フルスピードの場合の接続



(b) ロースピードの場合の接続



信号線としては D+と D-の２本しかないのですが、この２本に対して電気的な状態をいくつか区別して判定出来るようにしています。特に、信号の転送は、２本の線間の差動信号として送られますが、それ以外に、それぞれの線の信号レベルを独立に考えて、その状態で下表のように多くの状態を識別しています。従って、D＋と D-の信号ラインの電圧レベルには注意する必要があります。

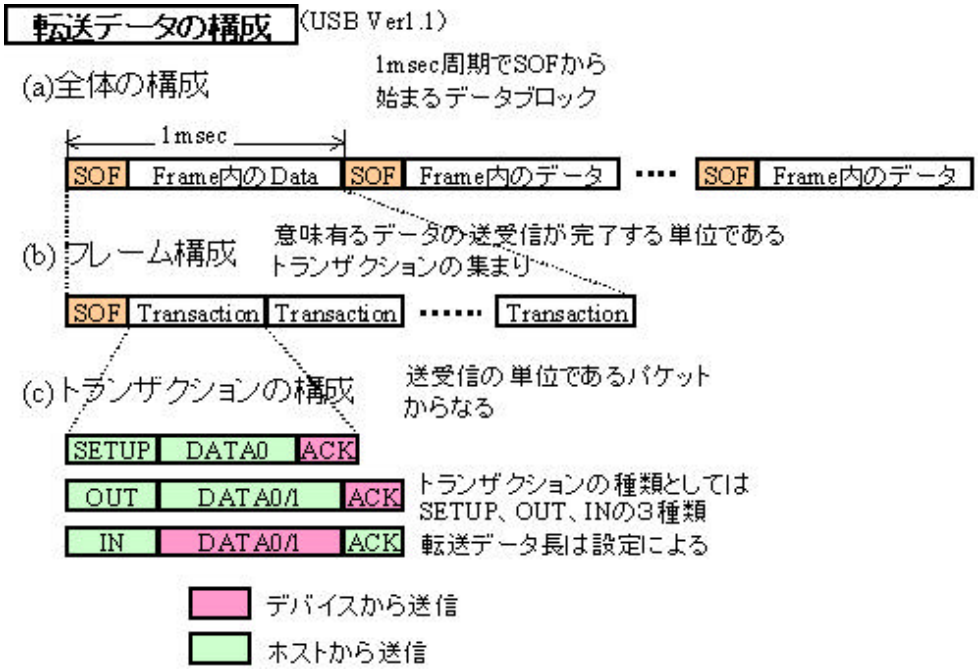
USBのバス状態とD+、D－信号レベル

USBバスの状態	出力端での信号レベル条件	入力デバイス側での信号レベル条件(要求仕様)
差動 “1”	D+ > V_H(min) かつ D－<V_{OL}(max)	論理1　D+－D－>0.2V　かつ　D+>V_{IH}(min)
差動 “0”	D－ > V_H(min) かつ D+ < V_{OL}(max)	論理0　D+－D－<－0.2V　かつ　D－>V_{IH}(min)
データ J	ロースピード ： 差動“0”と同じ フルスピード ： 差動“1”と同じ	ロースピードの時 ： “0” フルスピードの時 ： “1”
データ K	ロースピード ： 差動“1”と同じ フルスピード ： 差動“0”と同じ	ロースピードの時 ： “1” フルスピードの時 ： “0”
SE0(Single Ended0)	D+<V_{IL}(max) かつ D－<V_{IL}(max)	D+<V_{IL}(max) かつ D－<V_{IL}(max)
アイドル	信号なし	ロースピード ： D－>V_Z(min) 、D+< V_{IL}(max) フルスピード ： D－ <V_{IL}(max) 、D+>V_Z(min)
レジューム	K状態と同じ	K状態 と同じ
SOP(パケット開始)	アイドルからK状態へ遷移	アイドルからK状態への遷移
EOP(パケット終了)	J状態のあとSE0状態が2ビット以上	J状態のあとSE0が1ビット以上
非接続	無し	SE0状態が2.5μsec以上継続
接続検出	無し	アイドル状態が2msec以上継続(min 2.5μsec)
リセット	D+、D－両方<V_{OL}(max)が10msec以上	D+、D－両方<V_{IL}(max)が10msec以上継続

（注） V_{OH},V_{OL}:ドライバ出力電圧のHigh、Low
V_{IH}、V_{IL}:レシーバ入力電圧のHigh、Low

【フレームの構造】

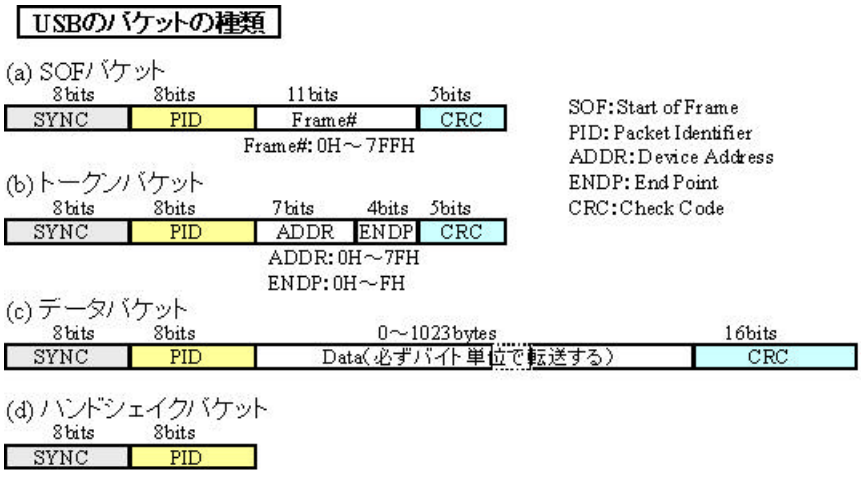
このUSBライン上を流れる実際のデータは、「**フレーム**」と呼ばれる単位で通信されています。このフレームは1msec周期で繰り返し転送されていて、すべてのデータが、このフレームの中でやり取りされています。このフレームの構造は、下図のように、「SOF（Start of Frame)」と呼ばれる「**パケット**」で始まる複数の「**トランザクション**」から成り立っています。パケットは、このUSB通信の中で通信される最小の単位となっていて、いくつかの種類があります。そしてこのパケットがいくつか通信されて、意味のあるデータ転送の単位となったものをトランザクションと呼んでいます。この関係を表したのが下図となります



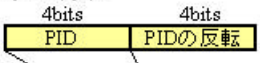
【パケットの種類】

上記のトランザクションを構成する最小の通信単位であるパケットは、実際に流れるデータの基本単位となっていて、その内容により下図のような種類があります。それぞれのパケットの最初には、かならず同期をとるための「SYNC」データが1バイト先頭に付いて送られます。

そして各パケットの最初には、そのパケットの種別を表す「PID」（Packet Identifier）が付き、それに続くデータの内容を定義しています。このPIDにはPID詳細フォーマットに示すような10種類がUSB1．1で取り決められています。PIDは1バイトで送られますが、実際のPIDは4ビットしかなく、その0，1を反転した4ビットをあわせて8ビットとしています。この反転2連送のデータを照合することにより転送の誤りチェックをすることができ、より高信頼なデータ転送をすることが可能となります。



★PID詳細フォーマット



種　別	PID名称	コード	定　　義
トークン	OUT	0001	エンドポイントへの転送
	IN	1001	ホストへの転送
	SOF	0101	フレームの開始
	SETUP	1101	セットアップ
データ	DATA0	0011	データパケット　PID偶数
	DATA1	1011	データパケット　PID奇数
ハンドシェイク	ACK	0010	データ正常受信
	NAK	1010	データ転送不成功
	STALL	1110	エンドポイントのストール
スペシャル	PRE	1100	ロースピード転送許可

【論理プロトコル】

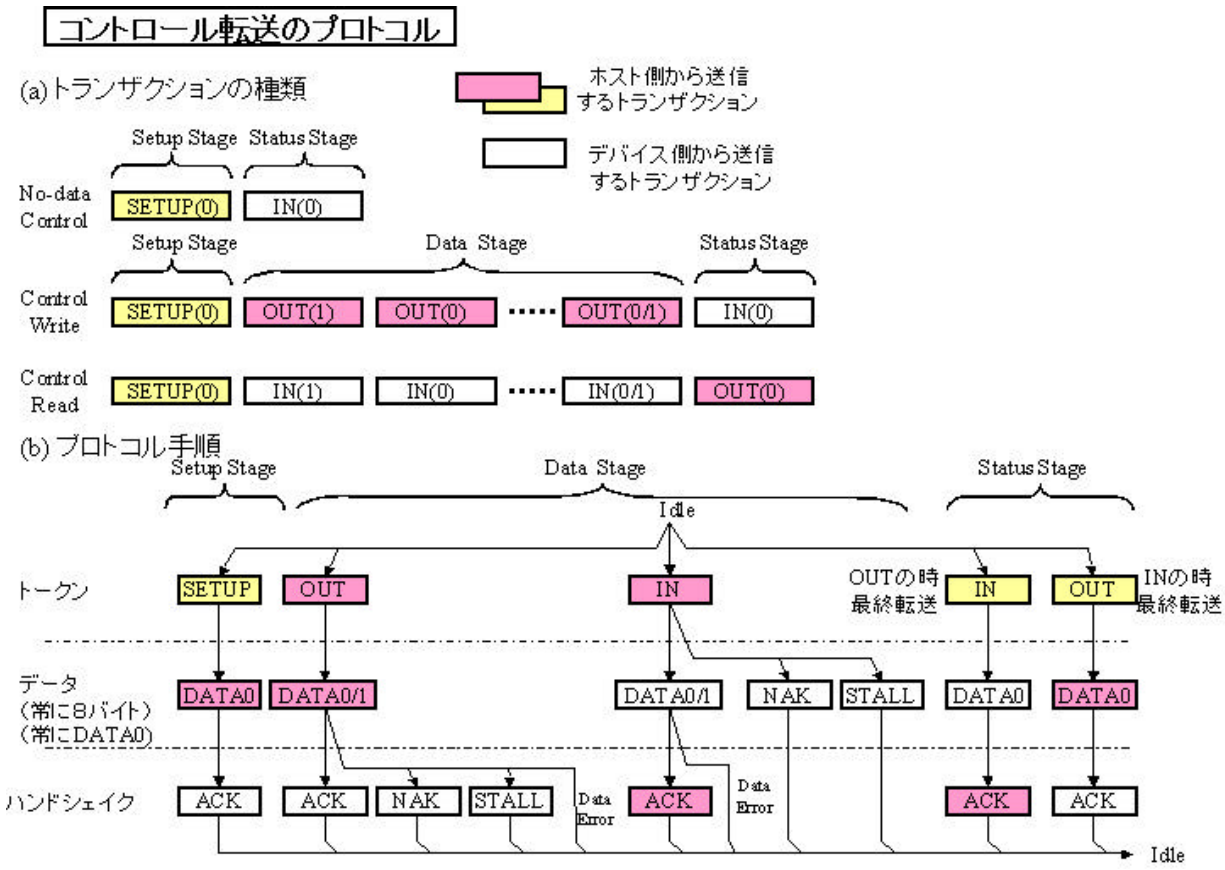
上記のようなパケットがやり取りされてひとつのトランザクションとなり、そのトランザクションが一定の手順に従ってやりとりされてやっと意味のあるデータ転送が行われます。そのデータ転送の手順は各転送モードによって決まっており、それぞれ下図であらわすことが出来ます。この手順により、デバイス側と、ホスト側の意志が通じ合うことになります。

(1) **コントロール転送のプロトコル**

コントロール転送だけが、半二重通信が可能な双方向通信パイプを使うため、プロトコルが特殊になっています。　まず全体が３つのステージに分かれています。最初にセットアップステージがあり、何をするかを要求するコマンドをホストから転送します。そしてそのコマンドが、データの送受信を要求している場合には、これに続いてデータステージが始まります。　この場合のデータは８バイト単位で複数のトランザクションになることもあり、その場合には DATA0 と DATA1 を交互にします。最後に転送結果を通知するために、ステータスステージがつながります。

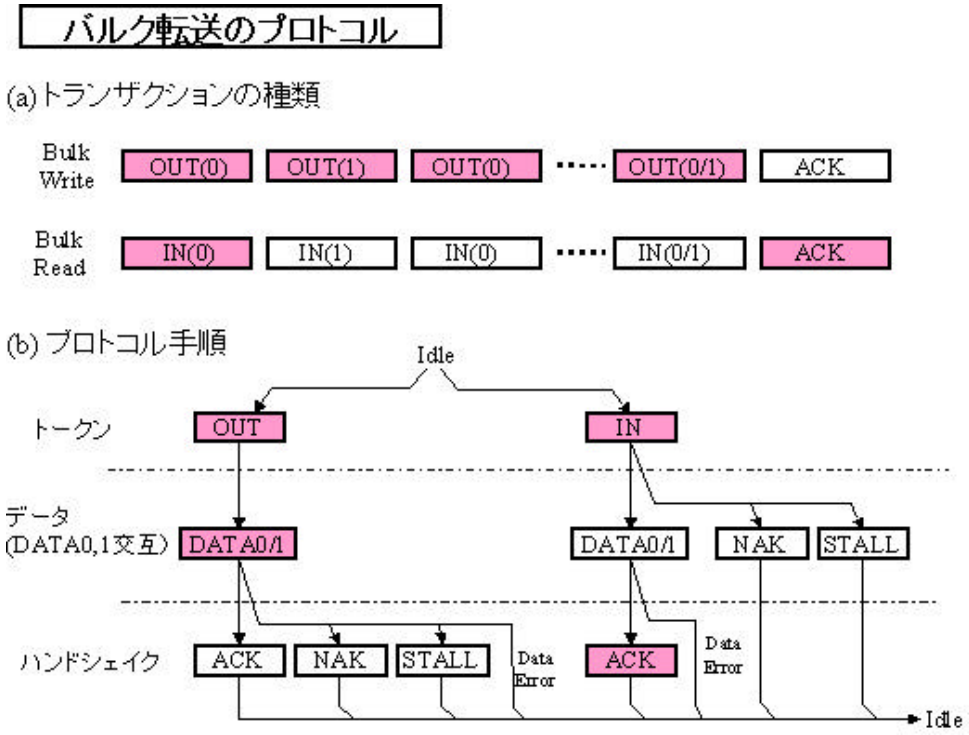
このステータスステージは特殊な扱いになっていて、直前の送信、受信データと反対向きの転送を行うことで、その転送の結果を通知します。

このプロトコル手順を図で表すと下図のようになっていて、最初は、ホストから送信される SETUP のトークン PID になります。それに引き続いて、送信データが出力されるか、逆に、データの送信を要求し受信するかになります。さらにもうひとつデータの全く無いものもあります。そしてそれぞれ最後のデータの通信が完了すると、それとは逆向きの転送が行われて結果データが返信されて完了となります。



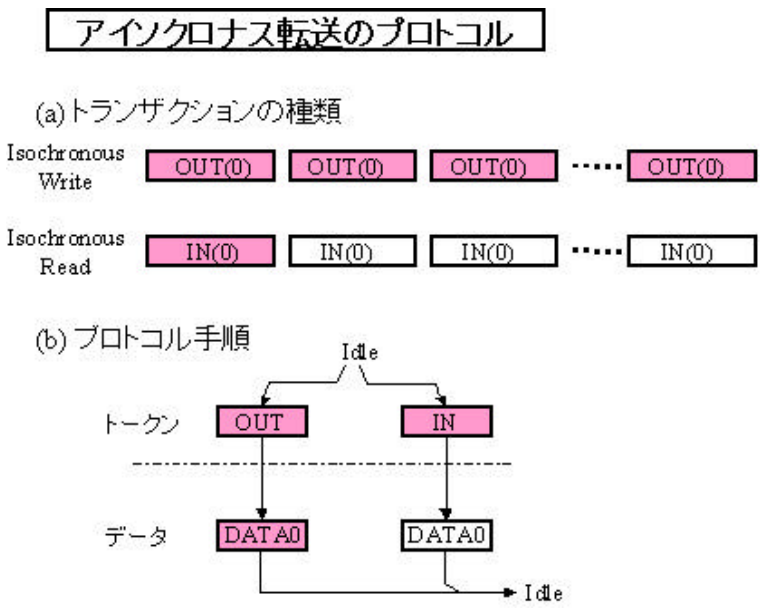
(2) バルク転送のプロトコル

バルク転送は通常のパイプを使うので、片方向通信となります。従ってもっとも基本的なプロトコルとなっていて、下図のようになります。ここで通常は、ACK を返送するルートとなりますが、受信出来ない状態の時には、待ってくれという意味で NAK を返信します。さらにデバイス側が動作不能な状態の時には、STALL を返信します。



(4) アイソクロナス転送のプロトコル

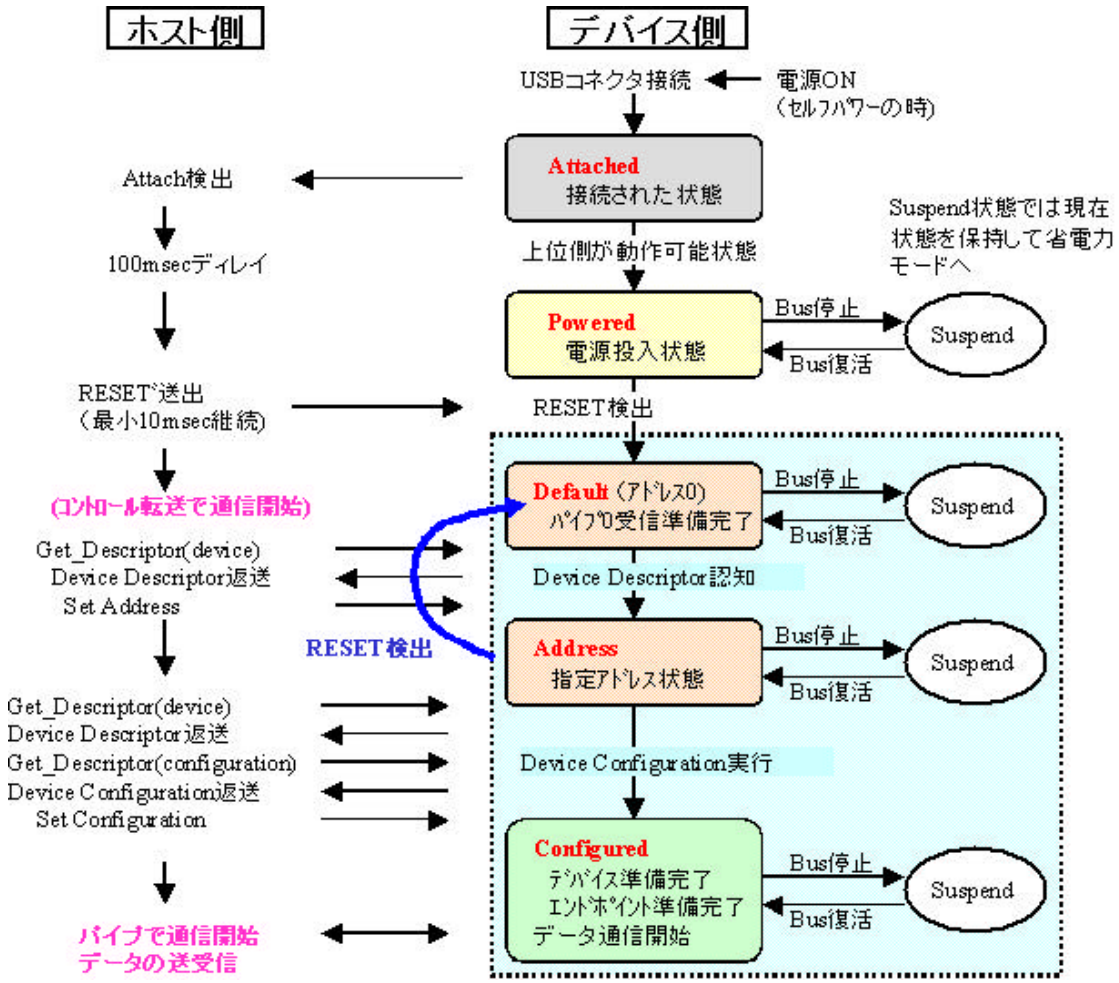
この転送の場合には、とにかくエラーがあっても良いから、一定時間以内に一定の量のデータを送受信したいという要求に応えるものなので、そのプロトコルは下図のように非常に単純になっていて、ハンドシェイクの部分が省略されています。これで高速の連続通信が確保できることになります。



３．USB のプラグ＆プレイ

【USB のプラグ＆プレイとは】
USB の特徴は、必要な時にパソコンの電源を ON にしたまま接続切り離しがいつでも可能な点にあります。これをサポートするのが「**プラグ＆プレイ機能**」となります。つまり接続された時点で自動的に必要なドライバソフトを探し、適当なものを起動して使えるようにしてくれます。そして一度接続してしまえば、あとは自由にいつでも接続、切り離しが可能となります。

このプラグ＆プレイ機能に対応させるためのプログラムがデバイス側に必要となります。まず、このプラグ＆プレイの手順を説明します。下図がその手順全体の流れとなります。
全体が大きく２段階に分かれていて、接続検出（アタッチ）から、電源投入状態までは、比較的ハードウェアに近いレベルの動作で行われ、そのあとは USB バス転送を使って、デバイス側のコンフィギュレーションを実行する段階となりソフトウェアですべて実行されます。



以下、各段階の動作を説明していきます。

(1) USB ケーブルの接続（アタッチ）
　まず、ホストとなるパソコン側からみると、USB ケーブルが接続されたということがきっかけになります。この検出は D+か D-いずれかが３．３V という状態になることで検出します。
従って、デバイス側の電源の供給方法によって、少し動作が異なり、セルフパワーとバスパワーの２つに区別されます。
　バスパワーは、USB ケーブルを通して、上位のハブかホストから電源供給を受けるタイプで、この場合にはケーブルの接続と電源投入が一緒に行われますから、接続と同時に検出されることになります。電源投入は、バスパワーはケーブル接続と同時ですから、「**Powered**」状態になるタイミングはケ

ーブル接続時となります。同時といっても細かく見ると、USB コネクタの形状に特徴があり、電源とグランドのピンが D+、D-より先に出ていて、コネクタを挿入するとき、D+、D-が接続されるより先に電源とグランドが接続されるようになています。

セルフパワーは、デバイス側が自分用の電源を内蔵しているもので、アタッチは電源 ON 状態で検知されます。しかし、セルフパワーの時は、電源投入だけでは、Powered 状態にはならず、電源 ON 状態で USB ケーブルがホストに接続された時が Powered 状態ということになります。ケーブルが先に接続されたときには、逆に電源 On にした時が Powered 状態となります。

(2) RESET

ホスト側がアタッチを検知すると、一定時間以上の RESET 状態を出力します。つまり D+,D-の両方を Low にします。これでデバイス側は RESET と認知して内部リセットを実行し、「**Default**」状態となります。この RESET 状態は、いつでも検出可能になっていて、内部リセット後は、Default 状態に戻ります。この Default 状態では、パイプ０を使ったコントロール転送が可能な状態となります。

(3) コンフィギュレーション

RESET を出力したあと、ホスト側から、コンフィギュレーションを開始します。つまり、デバイス内に保持している、「**デバイスディスクリプタ**」の情報の転送を要求します。

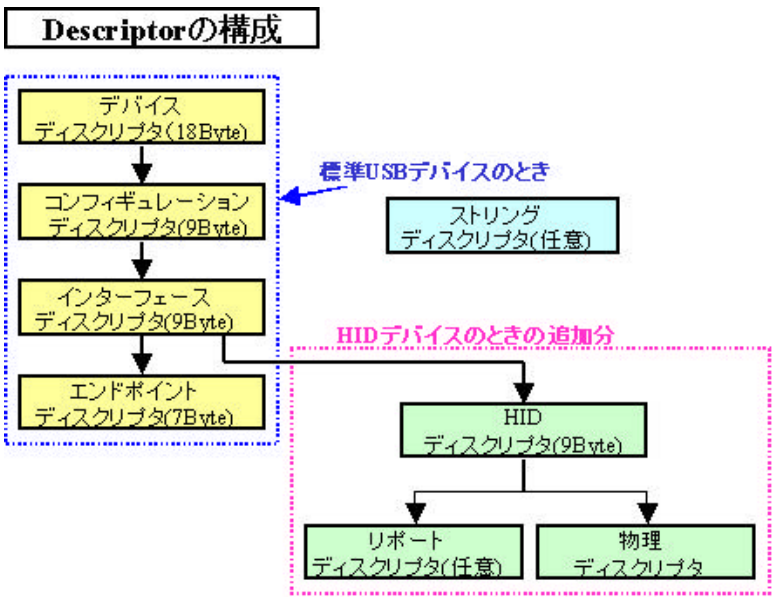
まずホストから「Get_Configuration」を転送し、データを返送を要求します。これに対し、デバイス側から「Configuration」のデータを返送します。次にホストは、このデバイスに対して、使われていないアドレスにセットアップするよう、デバイスに「Set_Address」を転送します。これでデバイスが特定のアドレスを持ったことになります。以降はこのデバイスアドレスで通信を行います。

このあと、再度ホストからディスクリプタの詳細情報を要求し、デバイスは、必要な情報を返送します。この中には、デバイスのリソースに関する情報が含まれており、パイプの使い方や、ID コードが送られます。ホスト側は、この ID コードを元にドライバを探し、適当なものを起動します。

はじめて接続検出した ID の場合には、ドライバのリンクの手順に進みます。ドライバのリンクが完了した時点で、ホストから「Set_Configuration」を送信して改めてデバイスのコンフィギュレーションをやり直して完了となります。以降は、コンフィギュレートした内容に基づいて通信が行われることになります。

【ディスクリプタの構成】

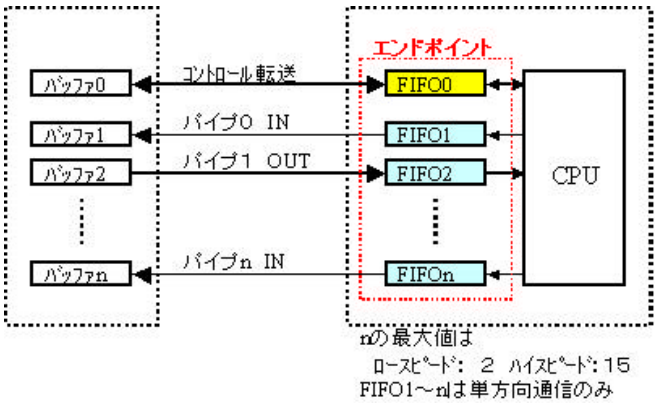
プラグ＆プレイの時にコンフィギュレーションを行いますが、そのときデバイスが持っているデバイスディスクリプタ情報で、そのデバイスの動作や機能を定義されます。
このディスクリプタの内容は、下図のようになっていて、デバイスクラスの差異により、何種類かの派生形式があります。下図の中で、基本となるのは、左半分の標準デバイスの時のものです。この中に基本的な動作モードが記述されています。下図の右半分は、HID デバイスと呼ばれる簡易なマンマシン操作機器のような場合に追加されるものです。



４．デバイスのハードウェア構成

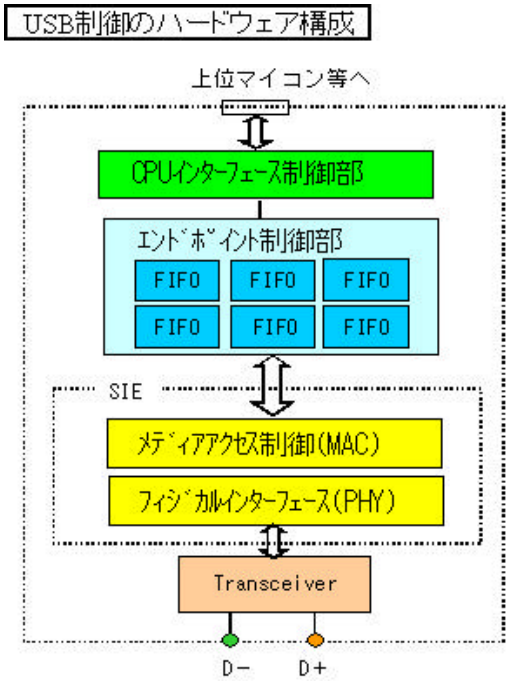
【デバイスのハードウェア構成】

デバイスの構成をハードウェアとしてみると下図のようになります。この図のように、デバイスは「**エンドポイント**」というＵＳＢ通信の対象が中心となって動作するので、これを構成要素の中心として扱います。このエンドポイントを含む USB 通信をつかさどる部分を、専用の IC として開発したものや、それを内蔵した USB マイコンが市販されています。特にデバイス用の USB コントローラは、通信プロトコルのかなりの部分を専用 IC で実行するようになっており、簡単に USB デバイスを作ることができるようになっています。



【デバイス用 USB コントローラ】

デバイス用 USB コントローラが持っているべき機能ブロックを図で表すと下図のようになります。この機能ブロックは、通信が高速であることから、高速動作を必要とするため、多くは専用の IC で構成されます。　マイコンで USB を内蔵しているものも、この USB インターフェースの部分は、マイコン内部に専用のハードウェアとして同居させているものが大部分となっています。



このデバイス用 USB コントローラの内部ブロックの機能を USB バス側から見て行くと、下記となります。

(1) トランシーバ(Transceiver)

USB バスと電気的なインターフェースをする部分で、送信用のドライバと、受信用のレシーバが、いずれも同一の２本のラインに接続されています。ここで重要な働きをするのが、レシーバ部で、単純に２本のラインに伝送されて来る差動信号のデータを受信するだけでなく、２本のそれぞれのラインの DC レベルも検出して、RESET など意味のある状態を検出できるようになっています。

(2) SIE（Serial Interface Engine）

高速のシリアルデータの送受信を実行するハードウェアブロックで、心臓部になります。ここは１２Mbps という高速通信を実現するため、非常に高速で動作する必要があります。そのため、専用のハードウェアで実現する必要があります。中味としては、フィジカルレイヤの機能をつかさどるブロックと、その上位層となって、自動応答などを果たす MAC 機能（Media Accsess Controller）などから構成されています。

(3) FIFO

エンドポイントの中心をなす FIFO（First In First Out）のデータバッファと、その FIFO の入出力の制御部から構成されます。

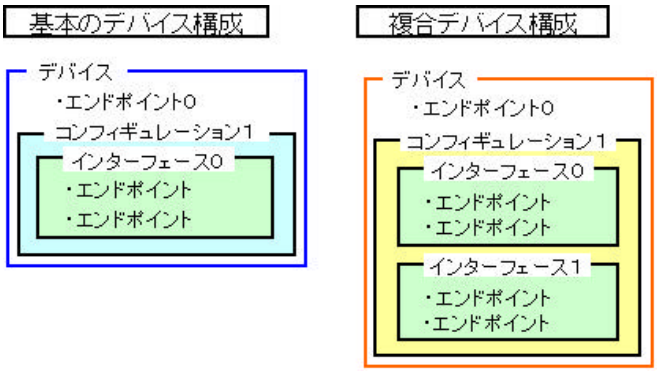
(4) CPU インターフェース部

上位のマイコンなどとのインターフェース部分です。

５．デバイスのソフトウェア構成

【デバイスのフレームワーク】
デバイスをソフトウェア構成から見たときには、前述のように、「オブジェクト」として扱われます。つまり、デバイスは、「デバイスオブジェクト」というクラスが代表となっていて、基本の枠組みが決まっています。このデバイス全体の枠組みは「**デバイスフレームワーク**」として定義されています。

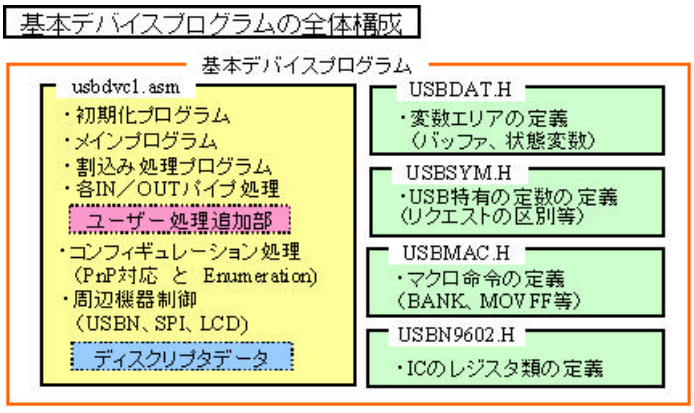
(1) 基本のデバイスのフレームワーク
最も基本的なデバイスクラスは下左図のような、エンドポイントとインターフェースとから構成されます。この基本構成では、コンフィギュレーションとして構成が定義され、その中味は、１個のインターフェースで定義される１つの要素機能だけとなります。この要素機能の中には、いくつかのエンドポイントが含まれ、これがＵＳＢ通信の対象として見えます。



(2) 複合デバイスのフレームワーク
上記の基本構成に対し、上右図はやや複雑な複合デバイスのフレームワークです。このデバイスの場合にも、コンフィギュレーションとして構成が定義されますが、その中味には、インターフェースが複数定義されていて、それぞれ独立の動作をする機能ブロックとなることが出来ます。つまり、複数の要素機能を持った複合のデバイスとして定義されていることになります。例えば、キーボードとマウスの組み合わせの複合デバイスのような場合に、このような構成が使われます。また、一つのハードウェアに対し、複数のインターフェースを持つことで、動的にインターフェースを切り替えて異なった機能を持たせることも可能です。

【ソフトウェアの構成】
実際のデバイスに必要なソフトウェアは基本的に下記のような機能部分から構成されます。

(1) 初期化プログラム
文字通りデバイスの初期化、プログラムやデバイスの状態遷移の初期化を行います。
(2) メインプログラム
デバイスの実際の機能を実現する部分で、常時はこのメインプログラムが動作していて、もともとのデバイスとしての動作を実行します。USB 通信が必要なときだけ下記の割込みにより USB 通信動作が実行されます。その結果として必要となる USB の送信、受信に関連する動作部分も含みます。



(3) USB 割込み処理
USB 通信は高速な動作であることと、デバイスコントローラが大部分の処理を実行してくれるので、プログラムとしてはコントローラの動作結果を割込みで得て、その通信のコントロールを行います。また重要な働きとして、デバイスがホストの USB バスに接続された時に実行されるデバイスコンフィギュレーションの動作を実行します。これにより、ホストにデバイスの機能内容を通知して、ホストがデバイスを正しく認識して正常動作が出来るようにします。
(4) ディスクリプタデータ
コンフィギュレーションをするときに、デバイスを定義したデータで、これをホストに渡すことでホストがデバイスを正しく認識することが可能となります。

【エンドポイントの定義】

基本デバイスプログラムとして定義されているエンドポイントは下表のようになっています。USBN9603 が、コントロール＋IN３個＋OUT３個の合計７個のエンドポイントが構成出来るようになっていますので、全部を使えるように定義しています。この定義は、ディスクリプタデータで行っていますので、これを変更するときには、ディスクリプタデータを修正する必要があります。またエンドポイントは No0 以外は全て片方向となっていますので、使うときには注意が必要です。

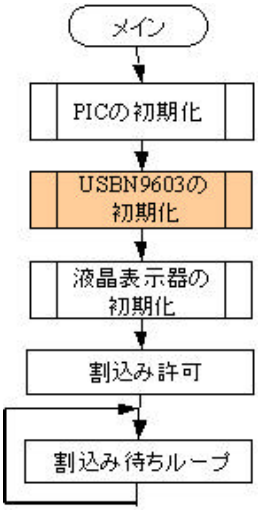
下表のように、エンドポイントはすべてバルク転送モードとして使うように設定しています。そしてパイプ番号も順番に付けていますが、エンドポイント０が異なり、番号がずれていますので使うときに間違えないよう注意が必要です。

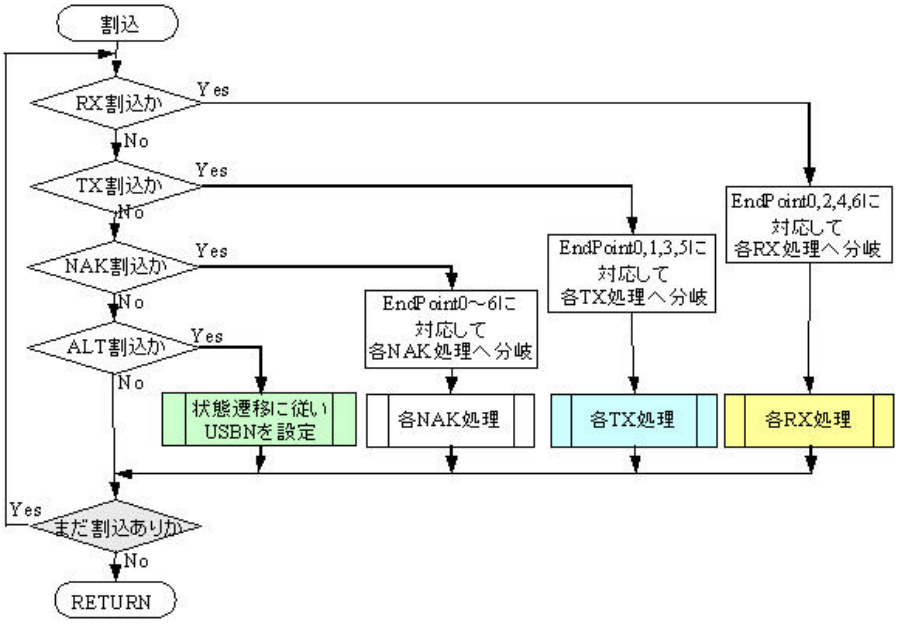
パイプ No	エンドポイント	IN/OUT	転送サイズ	ユーザー処理追加部
無し	0	IN/OUT	８バイト	無し
0	1	バルク IN	８バイト	DO_TX1
1	2	バルク OUT	８バイト	DO_RX1
2	3	バルク IN	８バイト	DO_TX2
3	4	バルク OUT	８バイト	DO_RX2
4	5	バルク IN	１６バイト	DO_TX3
5	6	バルク OUT	１６バイト	DO_RX3

【概略フロー】

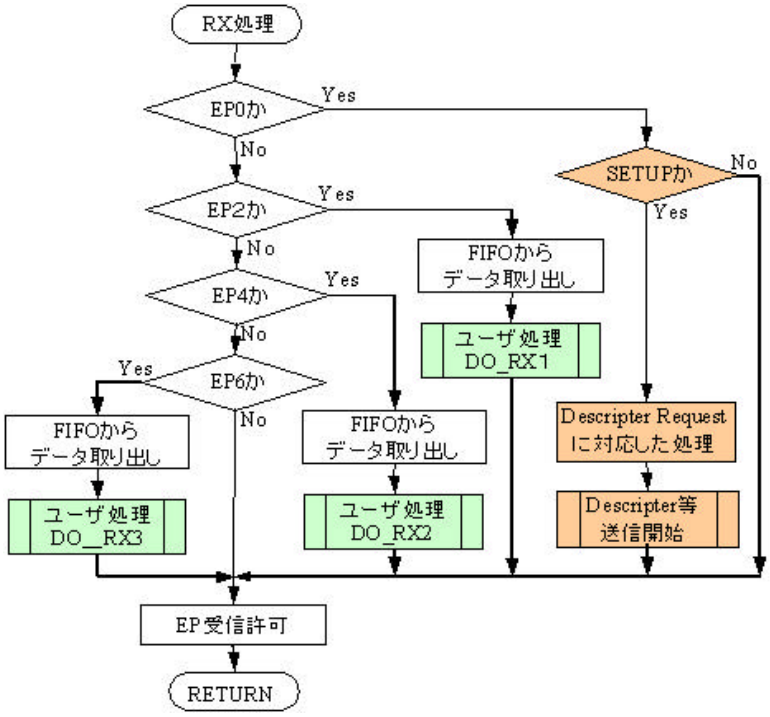
本基本デバイスプログラムの概略の流れは下図のフロー図のようになっています。まずメインプログラムで初期化したあと、単純に割込み待ちループとして USB の割り込みを待ちます。あとは USB の割込み要因によりそれぞれの処理に分岐しています。

　コンフィギュレーションが完了してしまえば、後は、ホストからのポーリングに応答するだけなので簡単なフローになります。特に、基本となる通信プロトコルは大部分 USBN9603 のデバイスコントローラが自動で処理してくれるので、デバイスのプログラムとしては、受信あるいは送信が完了した割込みで動作を開始するだけになります。また送受信処理に時間がかかるときには、USBN9603 が自動的に NAK 応答を返してホストを待たせてくれるので、一定時間以内に応答しなければならないということも無く、安心して処理を実行することが出来ます。

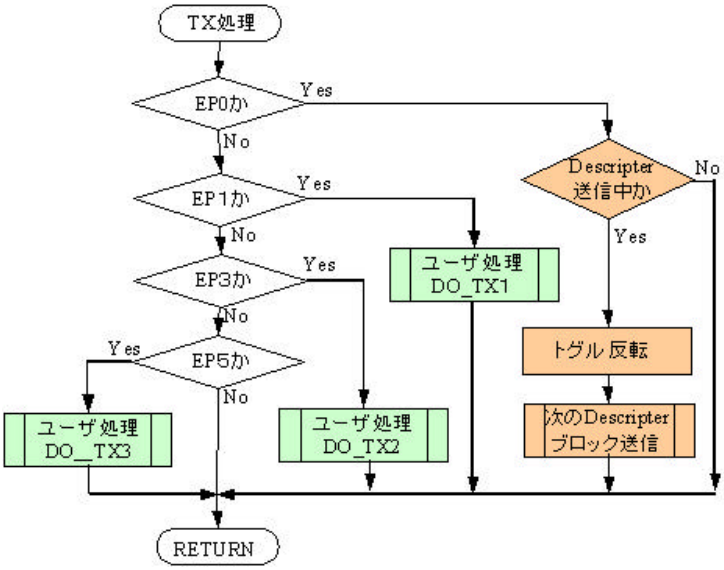




一番複雑なのは、ホストとなるパソコンに接続した時のプラグ＆プレイによるコンフィギュレーションが完成するまでの手続きです。この時の通信はエンドポイント０番で実行されますので、この処理の流れだけが、他のエンドポイントの処理と異なっています。SETUP のトランザクションの流れがこのコンフィギュレーションに関わる処理です。ディスクリプタの要求には多くの種類があるため、それぞれに対応した処理があります。他の３つの汎用エンドポイントはパイプとしてコンフィギュレーションで定義されます。そしてそれぞれにユーザ処理部を追加できるようにしています。このユーザ処理部はサブルーチンとして追加できるようにしています。
（DO_RX1、DO_RX2、DO_RX3 というサブルーチン名にしてあります）



TX 割込みに対する処理は比較的簡単で、エンドポイント０番に関する処理も、ディスクリプタの残りの送信ブロックがあれば、連続して送信を実行します。他の３つの汎用エンドポイントの処理部分にはユーザ処理部を追加できるようにしています。このユーザ処理部は、サブルーチンとして作成し追加します。
（DO_TX1、DO_TX2、DO_TX3 というサブルーチン名にしてあります）



６．汎用 USB ドライバの使い方

【汎用 USB ドライバ】
USB をパソコン側で使うには、デバイスを認識し、制御するためのデバイスドライバが必要になります。このデバイスドライバは自分で作るにはちょっと大事になりますので、ここは先人の開発されたものを使わせて頂くこととします。使用するのは、すでにあちこちの雑誌で紹介されている、柏野　政弘氏作の「汎用 USB ドライバ」です。

【汎用 USB ドライバの概要】
USB ドライバは、通常デバイス毎に開発が必要なデバイスドライバですが、汎用 USB ドライバは、それを単純化して、アプリケーションからの Read/Write をそのままデバイスへの USB の送受信コマンドとして送り出すものです。つまり、ドライバは、デバイスに依存した処理は何もせず、単純に USB 通信を実行しますので、私たちが自分で作成したデバイス用のドライバとして使い、機能はすべてアプリケーション側で実現すれば、どのようなデバイスにでも対応できることになります。
この汎用 USB ドライバは下記のようなモジュールで構成されています。
(1) 汎用デバイスドライバ　　**UUSBD．SYS**
　　デバイスドライバ本体で、これで実際の USB 通信を行いますが、デバイスに依存した処理を含まない透過なドライバです。
(2) デバイス情報ファイル　　**UUSBD．INF**
　　デバイスをセットアップする際に必要となるデバイス情報ファイルで、ここに作成したデバイスの、ベンダ ID とプロダクト ID を追加記述する必要があります。追加の仕方は UUSBD.INF ファイル中にコメントで説明してあります。
(3) API 提供ライブラリ　　**UUSBD．DLL**
　　実際にアプリケーションから USB デバイスドライバを使うときに便利な API を提供する DLL で、Win32 の標準的な DLL となっているので、Visual Basic、Visual C++、Delphi など色々なアプリケーション開発ツールから使うことが可能です。

汎用 USB ドライバと UUSBD．DLL で提供されるアプリケーション用の関数の中で、通常使用する API は下記となっています。

機　能	API 関数の形式	機　能　詳　細
USB デバイスのオープン	Uusbd_Open( )	USB デバイスをオープンしてハンドルを取得する汎用 USB デバイスドライバを使っているデバイスが見つからないときは、－１を返す。
USB デバイスのオープン 検索機能つき	Uusbd_Open_mask(flag、Class、SubClass、Vendor、Product、bcdDevice)	検索条件に該当する USB デバイスをオープンしてハンドルを取得する。該当するデバイスが見つからないときは、－１を返す。 《検索パラメータ》 flag：以下のどれを検索条件にするか指定する OR で複数条件指定可能 UU_MASK_CLASS、UU_MASK_SUBCLASS、 UU_MASK_VENDOR、UU_MASK_PRODUCT UU_MASK_BCDDEVICE、UU_MASK_NO Class：デバイスのクラスコード SubClass：デバイスのサブクラスコード Vendor：ベンダ ID Product：プロダクト ID ｂｃｄDevice：デバイスリリース番号
USB デバイスのクローズ	Uusbd_Close(husb)	husb で指定されたハンドルの USB デバイスをクローズしシステムからはずす。
パイプのオープン	Uusbd_OpenPipe(husb、interface_num、pipe_num)	husb で指定された USB デバイスのエンドポイントにアクセスするためのファイルハンドルを取得する。 エンドポイントとパイプの番号は異なるので注意。指定したエンドポイントが無い時は－１を返す。このハンドルを使って ReadFile、WriteFile を行えば USB デバイスにアクセスできる。

【インストール方法】
汎用 USB ドライバをインストールするには、下記の手順で行います。

(1) **デバイス情報ファイルにユーザーのデバイスを追加する。**
追加個所は２ヶ所あります。それぞれ同じベンダーID、**プロダクト** ID **とします。この** ID **は個人使用であれば何でも構いませんのでデバイスで設定したものに合わせます。**
(2) **デバイスドライバのインストール**
USB **デバイスを接続すると、はじめて接続した場合には、ドライバ検索のウィザードが起動しますので、それに従ってインストールします。途中で、要求される** INF **ファイルと** UUSBD．SYS **ファイルの所在場所には、実際に両ファイルがあるディレクトリを指定します。あとはドライバが自動的にインストールされ** USB **デバイスがデバイスマネージャに登録されます。**
(3) UUSBD.DLL **を** Windows¥system32 **にコピーする**

【使い方】
実際にアプリケーションから、USB **ドライバにアクセスする方法は下記のようにします。これで、あたかもファイルシステムと同じような扱いで** USB **デバイスが扱えることになります。**

《基本的な使い方》
(1) USB **デバイスをオープンする**
Uusbd_Open( )**関数か、**Uusbd_opne_mask( )**関数を使って目的の** USB **デバイスをオープンし、デバイスハンドラ番号(husb)を取得します。**
(2) **パイプをオープンする**
上記で取得したデバイスハンドラを使って、Uusbd_OpenPipe( )**関数でパイプをオープンしパイプのハンドラ(handle)を取得します。必要なパイプをすべてオープンしておきます。**
(3) **パイプハンドラを使って** USB **デバイスにアクセスする**
ファイルシステムと同じあつかいとなりますので、下記のように VB **などの標準関数を使ってアクセスします。**
・USB **デバイスの指定パイプからの** Input
ReadFile(handle、Buffer、Number、pNumber、pOverlap) **関数でアクセスします。**
入力した Number **バイトのデータは** Buffer **に格納されます。**
・USB **デバイスの指定パイプへの** Output
WriteFile（handle、Buffer、Number、pNumber、pOverlap） **関数でアクセスします。**
データは Buffer **から** Number **バイトだけ送信されます。**
(4) **終了時には** USB **デバイスをクロースする**
Uusbd_Close( )**関数を使って全てのパイプとデバイスをクローズします。**

【実際の使用例】
下記は、USB **接続のデータロガ-で使用した実際の例です。**
(1) USB **デバイスのオープン処理**
まず、検索付きデバイスオープンで USB **デバイスを検索しデバイスハンドラを取得します。発見できなければ－１が返されるのでエラー処理をします。発見出来たときには、４つのパイプをオープンしハンドラ値を取得します。それぞれオープン出来なければ－１が返されるのでエラー処理をします。パイプのハンドラには後で区別が付きやすい変数名を使います。このデバイスのパイプはすべてバルク転送モードになっています。**

```
Private Sub Form_Load()
'
'   ロガ-USB制御　初期化

    hUSB = Uusbd_Open_mask(UU_MASK_VENDOR + UU_MASK_PRODUCT, 0, 0, Vendor, Product, 0)
    If hUSB = -1 Then
        MsgBox "USBロガーが見つかりません"
        End
    End If

    hCMD = Uusbd_OpenPipe(hUSB, 0, 1)
    If hCMD = -1 Then
        MsgBox "コマンド出力のパイプ１を開けませんでした"
        End
    End If

    hLED = Uusbd_OpenPipe(hUSB, 0, 3)
    If hLED = -1 Then
        MsgBox "LED制御のパイプ３を開けませんでした"
        End
    End If

    hMSR = Uusbd_OpenPipe(hUSB, 0, 4)
    If hMSR = -1 Then
        MsgBox "計測データのパイプ４を開けませんでした"
        End
    End If

    hLCD = Uusbd_OpenPipe(hUSB, 0, 5)
    If hLCD = -1 Then
        MsgBox "LCD表示制御のパイプ５を開けませんでした"
        End
    End If

    Timer1.Enabled = False
```

(2) USB デバイスへのアクセス

実際にパイプがオープンできれば、そのハンドラを使ってデバイスにアクセスします。

下記の例では、まず OUT パイプ１にコマンドを出力し、そのコマンドに対するデバイスからの応答を IN パイプ４から入力しています。ReadFile、WriteFile 関数はエラーが発生すると、0 が返されるのでエラー処理をしています。

```
Private Sub Command2_Click()
'
'   計測コマンド出力（バルクOUT　パイプ１）

    Dim L As Long, I As Long
    Dim data(8) As Byte
    Dim log_data(16) As Byte

    Timer1.Enabled = False

    data(0) = 1     'Command
    L = WriteFile(hCMD, data(0), 8, I, 0)
    If L = 0 Then   'エラー発生時
        MsgBox "コマンド出力のパイプ１の書き込みでエラーが発生"
        End
    End If

    L = ReadFile(hMSR, log_data(0), 16, I, 0)
    If L = 0 Then
        MsgBox "計測データ入力のパイプ４の読み出しでエラーが発生"
        End
    End If
```

(3) USB デバイスのクローズとプログラム終了

プログラム終了時点で USB デバイスもクローズして、他で使えるようにフリーの状態にします。

パイプを全部クローズしてから、デバイスをクローズします。パイプのクローズには、VB 標準関数の CloseHandle( )関数を使います。

```
Private Sub Form_Unload(Cancel As Integer)
'
'   プログラム終了

    CloseHandle (hLCD)      'バルクOUTパイプ５クローズ
    CloseHandle (hMSR)      'バルクINパイプ４クローズ
    CloseHandle (hLED)      'バルクOUTパイプ３クローズ
    CloseHandle (hCMD)      'バルクOUTパイプ１クローズ
    Uusbd_Close (hUSB)      'USBデバイスクローズ
    End
End Sub
```