(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-334273
(P2004-334273A)
(43)公開日　平成16年11月25日(2004.11.25)

(51)Int.Cl.[7]	F I	テーマコード（参考）
G06F 13/38	G06F 13/38 350	5B077

審査請求 未請求　請求項の数 4　ＯＬ　（全 22 頁）

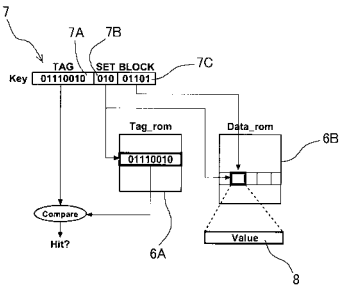
(21)出願番号　特願2003-124988(P2003-124988)
(22)出願日　平成15年4月30日(2003.4.30)

(71)出願人　000125369
学校法人東海大学
東京都渋谷区富ヶ谷２丁目２８番４号
(74)代理人　100087468
弁理士　村瀬　一美
(74)代理人　100120879
弁理士　井口　恵一
(72)発明者　清水　尚彦
神奈川県平塚市北金目１１１７　学校法人東海大学内
(72)発明者　名野　響
神奈川県平塚市北金目１１１７　学校法人東海大学内

Ｆターム(参考)　5B077 BA09 MM02 NN02

(54)【発明の名称】デバイスおよびデバイスの制御方法およびデバイス制御用プログラム

(57)【要約】
【課題】ホストからのリクエストの判別を少ない処理ステップで高速に行なう。
【解決手段】デバイスは、予め定められた一定処理によりリクエストから抽出されるキー７と対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリュー８が格納された記憶部６Ｂを有し、ホストから受信したリクエストから一定処理によってキー７を作成する処理と、作成されたキー７に対応するバリュー８を記憶部６Ｂから検索する処理と、検索されたバリュー８に基づいて当該リクエストに応じた処理とを実行する。
【選択図】　図１



【特許請求の範囲】
【請求項１】
ホストからのリクエストを受信し、該リクエストに応じた処理を実行するデバイスにおいて、予め定められた一定処理により前記リクエストから抽出されるキーと対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリューが格納された記憶手段を有し、ホストから受信したリクエストから前記一定処理によってキーを作成する処理と、上記作成されたキーに対応するバリューを前記記憶手段から検索する処理と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行することを特徴とするデバイス。
【請求項２】
ホストからのリクエストを受信したデバイスに当該リクエストに応じた処理を実行させる制御方法において、予め定められた一定処理により前記リクエストから抽出されるキーと対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリューを記憶手段に格納しておき、前記デバイスに、ホストから受信したリクエストから前記一定処理によってキーを作成する処理と、上記作成されたキーに対応するバリューを前記記憶手段から検索する処理と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行させることを特徴とするデバイスの制御方法。
【請求項３】
ホストから受信したリクエストから、予め定められた一定処理によりキーを作成する手段と、前記キーと対応付けられる箇所に、該当リクエストに応じた処理に必要な情報を示すバリューが予め格納されている記憶手段から、上記作成されたキーに対応するバリューを検索する手段と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行する手段として、デバイスを機能させることを特徴とするデバイス制御用プログラム。
【請求項４】
前記キーはタグ、セット、ブロックの３つのフィールドで構成され、前記記憶手段は、対象リクエストに基づく前記セットの値をアドレスとして当該リクエストに基づく前記タグの値が格納されたタグ記憶部と、対象リクエストに基づく前記セットおよび前記ブロックの値をアドレスとして当該リクエストに関する前記バリューが格納されたバリュー記憶部とを有し、上記作成されたキーについて、前記セットの値をアドレスとする前記タグ記憶部の値と、前記タグの値とを比較し、一致する場合に、前記セットおよび前記ブロックの値をアドレスとして前記バリュー記憶部から前記バリューを読み出す手段として、デバイスを機能させることを特徴とする請求項３記載のデバイス制御用プログラム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、デバイスおよびデバイスの制御方法およびデバイス制御用プログラムに関する。さらに詳述すると、本発明は、ＵＳＢデバイスおよびＵＳＢデバイスの制御方法およびＵＳＢデバイス制御用プログラムに関する。
【０００２】
【従来の技術】
ＵＳＢ（Ｕｎｉｖｅｒｓａｌ　Ｓｅｒｉａｌ　Ｂｕｓ）は、パーソナルコンピュータの周辺機器のインターフェースとして、広く利用されている。ＵＳＢのバスは差動シリアルバスであり、システムはホスト主導となっている。ＵＳＢバスに接続されたＵＳＢデバイスは、ＵＳＢホスト（パーソナルコンピュータなど）に対して割り込みをかけてデータの転送を行うのではなく、必ずＵＳＢホストからのリクエストにＵＳＢデバイスが応答して通信を行うという特徴がある。また、ＵＳＢではプラグアンドプレイとホットプラグに対応しており、ＵＳＢホストの電源を入れたままで、ＵＳＢバスにＵＳＢデバイスを接続できる。ＵＳＢデバイスの接続後、ＵＳＢホストはＵＳＢデバイスの構成を認識してドライバ等の必要なソフトウェアをインストールし、その後、ＵＳＢデバイスが使用可能となる。
【０００３】

ＵＳＢホストは、デバイスの構成情報が記述されているディスクリプタと呼ばれるデータを、ＵＳＢデバイスから受信し、当該ディスクリプタを元にＵＳＢデバイスを認識し、必要なソフトウェアをインストールする。ＵＳＢデバイスは、ＵＳＢバスに接続した後、ＵＳＢホストからのリクエストを受信し、このディスクリプタをＵＳＢホストに送信する必要がある。標準的なディスクリプタとしては、例えば次のものがある。第１にデバイスディスクリプタがある。これは、ＶｅｎｄｏｒＩＤやＰｒｏｄｕｃｔＩＤなどの製品情報やデバイスクラス（デバイスの種類）といったＵＳＢデバイスの基本的な情報を示す。第２にコンフィグレーションディスクリプタがある。これは、デバイスの電源供給方法や消費電力の情報、インターフェースの個数を示す。第３にインターフェースディスクリプタがある。これは、エンドポイントの個数やインターフェースのクラスを示す。第４にエンドポイントディスクリプタがある。これは、エンドポイントの転送タイプや最大パケットサイズ等を示す。第５にストリングディスクリプタがある。これは、各ディスクリプタが使用するストリングデータ（文字データ）を記述する。尚、上記に例示したディスクリプタ以外にも、デバイスクラスに特有のディスクリプタが存在する。

【０００４】

ＵＳＢバスには１２７台までＵＳＢデバイスを接続する事が可能である。ＵＳＢバスに接続されたＵＳＢデバイスに、１から１２７までのいずれかのユニークアドレスを付加する事で、ＵＳＢホストは個々のＵＳＢデバイスを判別するようにしている。ＵＳＢバスに接続直後は、デバイスアドレスは０であり、その後、ＵＳＢホストからのリクエストによりユニークアドレスを付加される。当該リクエストが発行されると、ＵＳＢデバイスは、指定されたアドレスをセットし、それ以降は自分のアドレスに対するリクエストにのみ応答する。

【０００５】

ＵＳＢ通信は、ＵＳＢホストとＵＳＢデバイスのエンドポイント（ＥＰ）間で行う。エンドポイントは０～１５番まで持たせる事が可能で、１つのエンドポイントに、使用する転送の種類と転送方向を指定する。但し、ＥＰ０は例外でＵＳＢデバイスのコンフィグレーションの為に予約されている。ＵＳＢデバイスは、ホストからのリクエストがどのエンドポイントに対するものか判別し、対応した通信を行う。

【０００６】

この様にエンドポイントを切替える事で、デバイスに複数の機能を持たせる事が可能となる。例えば図２０のＵＳＢデバイス１００は、デバイス１００Ａとデバイス１００Ｂの機能を備えている。このような構成は、例えばＵＳＢキーボードにＰＳ／２マウスの接続ポートがある場合などが該当する。この場合、キーボードとマウスの間ではＰＳ／２のプロトコルで通信が行なわれ、ホストから見れば１つのＵＳＢデバイスがキーボードとマウスの機能を持っている事になる。例えばこのＵＳＢデバイス１００では、ＥＰ０へのリクエストを受信した場合、デバイスコンフィグレーションの為の通信を行い、ＥＰ１へのリクエストを受信した場合、デバイス１００Ａのデータを ＵＳＢホストに送信する通信を行う。また、ＥＰ２へのリクエストを受信した場合は、デバイス１００Ｂのデータを ＵＳＢホストに送信する通信を行う。

【０００７】

図２１及び図２２にＵＳＢの転送フォーマットを示す。ＵＳＢには３つの転送単位があり、最小単位としてパケットがある。ＵＳＢホストとＵＳＢデバイスは、このパケットを数回送受信する事で、トランザクションを構成する。そして、トランザクションを組み合わせる事でトランスファを作る。

【０００８】

パケットは、ホストやデバイスがバス時間を確保する単位であり、ＳＹＮＣ（同期）パターンで始まり、ＥＯＰ（Ｅｎｄ Ｏｆ Ｐａｃｋｅｔ）までの連続したビット列で構成される。このパケットは、ＵＳＢデバイス側のエンドポイントとしてのＦＩＦＯ（Ｆｉｒｓｔ Ｉｎ Ｆｉｒｓｔ Ｏｕｔ）バッファと、ホスト側のエンドポイントとしてのＦＩＦＯバッファ間で送受される。パケットのフォーマットを図２１に示す。尚、図２１（Ａ）

はトークンパケット（ＯＵＴパケット、ＩＮパケット、ＳＥＴＵＰパケット）のフォーマットを示し、（Ｂ）はデータパケットのフォーマットを示し、（Ｃ）はハンドシェイクパケット（ＡＣＫパケット、ＮＡＫパケット）のフォーマットを示す。

【０００９】

パケットの各フィールドは、サイズが決まっており（但し、ＤＡＴＡフィールドは例外で０～１０２４バイトまで可変となる）、フィールド単位で次の様な意味がある。ＰＩＤ（Ｐａｃｋｅｔ　ＩＤ）フィールドは、パケットの種類を表す。ＡＤＤＲフィールドは、転送の対象となるデバイスアドレスを示す。ＥＰフィールドは、デバイスのエンドポイントを示す。ＤＡＴＡフィールドには、ホストからのリクエスト内容や、各種データ（キーボードであれば入力データ、プリンタであれば印刷データなど）がセットされる。また、トークンパケットやデータパケットにおいては、エラーチェックの為に、ＣＲＣ５フィールド、ＣＲＣ１６フィールドがある。ＣＲＣ（Ｃｙｃｌｉｃ　Ｒｅｄｕｎｄａｎｃｙ　Ｃｈｅｃｋ）は、通信エラーの検出に使用され、エラー検出の対象となるデータからＣＲＣコードを算出し、その値からエラーの有無をチェックできる。

【００１０】

このパケットを複数組み合わせたトランザクションは、図２２のようになっており、必ずＵＳＢホストから発行されるトークンパケットから始まり、ハンドシェイクパケットで終るようになっている。ＵＳＢデバイスは、ＵＳＢホストからトークンパケットを受信すると、トークンパケット内のＰＩＤフィールドの値からパケットの種類を判別し、ＡＤＤＲフィールドの値から自分のアドレスに対するリクエストか否か判別し、自分へのリクエストであれば、トランザクションを形成する。

【００１１】

さらに、トランザクションを複数回行う事で、トランスファと言う通信単位を構成する。トランスファには、（１）コントロール転送、（２）バルク転送、（３）インタラプト転送、（４）アイソクロナス転送、と言う４種類のフォーマットがあり、転送の目的に応じて使い分けられている。ＵＳＢデバイスは、どの転送の場合も、トークンパケットのＡＤＤＲフィールドおよびＥＰフィールドのチェックが必要である。

【００１２】

特に、コントロール転送で使用されるＳＥＴＵＰトランザクションでは、ＵＳＢデバイスは、ＳＥＴＵＰパケットの後にＵＳＢホストから送信されるデータパケット内のＤＡＴＡフィールドの値から、ＵＳＢホストからのリクエストの内容を判別しなければならない。当該リクエストの内容は、デバイスの構成情報（ディスクリプタ）の送信リクエストや、デバイス内のステータスのセット等である。このデータパケットのＤＡＴＡフィールドは８バイトであり、ｂｍＲｅｑｕｅｓｔ、ｂＲｅｑｕｅｓｔ、ｗＶａｌｕｅ、ｗＩｎｄｅｘ、ｗＬｅｎｇｔｈの５つのフィールドに更に区分される。これらの各フィールドは表１に示す意味を持つ。また、リクエストの種類とその時のＤＡＴＡフィールドのパターンを表２に示す。

【００１３】

【表１】

フィールド名	サイズ(バイト)	内容
bmRequestType	1	リクエストの特性、ビット単位で意味を持つ 7ビット：データ転送方向 0：ホスト→デバイス / 1：デバイス→ホスト 6～5ビット：タイプ 0：標準 / 1：クラス / 2：ベンダ / 3：予約 4～0ビット：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：その他/ 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

【００１４】
【表２】

bmRequestType	bRequest	wValue	wIndex	wLength	リクエストの内容
00000000B 00000001B 00000010B	CLEAR FEATURE(1)	DEVICE REMOTE WAKEUP(1) ENDPOINT HALT(0)	0 インターフェース エンドポイント	0	wValueで選択した機能をクリア
10000000B	GET CONFIGURATION(8)	0	0	1	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	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	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	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	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	wValueに指定された構成に切替える
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	指定したディスクリプタを更新
00000000B 00000001B 00000010B	SET FEATURE(3)	DEVICE REMOTE WAKEUP(1) ENDPOINT HALT(0) TEST MODE(2)	0 インターフェース エンドポイント	0	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	代替設定値を切替える
10000010B	SYNCH FRAME(12)	0	エンドポイント	2	アイソクロナス転送同期化フレーム

【００１５】
ＳＥＴＵＰトランザクションを含む転送を行う場合、ＵＳＢデバイスは表２に示す様なリクエストパターンを判別し、それに応じた通信を行う必要がある。尚、表２に示すリクエストは、標準デバイスリクエストと呼ばれるＵＳＢデバイス共通のリクエストである。この他に、デバイスの種類（デバイスクラス）に応じて、特有のリクエストも存在する。
【００１６】
従来、ＵＳＢデバイスに実装されているＵＳＢ２．０コントローラでは、ほとんどの処理をハードウェア化することで、処理の高速化を図っている（例えば非特許文献１参照）。
【００１７】
【非特許文献１】
インターネット＜ＵＲＬ：ｈｔｔｐ：／／ｗｗｗ．ｃａｓｔ－ｉｎｃ．ｃｏｍ／ｃｏｒｅｓ／ｃｕｓｂ２／ｉｎｄｅｘ．ｓｈｔｍｌ＞
【発明が解決しようとする課題】
ＳＥＴＵＰトランザクションにおいてＵＳＢデバイスが行なうＵＳＢホストからのリクエ

スト内容の判別をソフトウェアにより行なうと、例えば図２３のフローチャートのようになる。ＤＡＴＡフィールド内の「ｂｍＲｅｑｕｅｓｔＴｙｐｅ」と「ｂＲｅｑｕｅｓｔ」から、リクエストがＧＥＴ　ＤＥＳＣＲＩＰＴＯＲ（ＵＳＢホストへのディスクリプタ送信）であるかチェックする（図２３のＳ１０１）。そして、ＧＥＴ　ＤＥＳＣＲＩＰＴＯＲの場合は、ＤＡＴＡフィールド内のｗＶａｌｕｅ（２バイト）がディスクリプタタイプとインデックスを示している為、送信ディスクリプタの決定には図２３に示す様なｗＶａｌｕｅの比較が必要となる（図２３のＳ１０２，Ｓ１０３，Ｓ１０４…）。そして、当該比較回数は、ＵＳＢデバイスが持つディスクリプタの数に比例し、ディスクリプタの数が増える程、多くの命令ステップが必要となる。このため、特にＵＳＢ２．０で定義されるハイスピードモードに対応するためには、超高速のコントローラ用ＣＰＵが必要となり、コスト高および消費電力が大きくなる等の問題がある。また、図２３の処理をハードウェア化する場合、専用回路を組み込むためコスト高となる、回路の増設により消費電力が大きくなる、ハードウェア化に伴いソフトウェアで可能な柔軟な制御が困難となる、等の問題がある。このため従来技術では、特に携帯電話やＰＤＡ（Ｐｅｒｓｏｎａｌ　Ｄｉｇｉｔａｌ　Ａｓｓｉｓｔａｎｔｓ）などの低消費電力機器に対して、ＵＳＢ２．０コントローラを導入することが困難であった。

【００１８】

そこで本発明は、ホストからのリクエストの判別を少ない処理ステップで高速に行なうデバイスおよびデバイスの制御方法およびデバイス制御用プログラムを提供することを目的とする。

【００１９】

【課題を解決するための手段】

かかる目的を達成するため、請求項１記載の発明は、ホストからのリクエストを受信し、該リクエストに応じた処理を実行するデバイスにおいて、予め定められた一定処理により前記リクエストから抽出されるキーと対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリューが格納された記憶手段を有し、ホストから受信したリクエストから前記一定処理によってキーを作成する処理と、上記作成されたキーに対応するバリューを前記記憶手段から検索する処理と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行するようにしている。

【００２０】

また、請求項２記載の発明は、ホストからのリクエストを受信したデバイスに当該リクエストに応じた処理を実行させるデバイスの制御方法において、予め定められた一定処理により前記リクエストから抽出されるキーと対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリューを記憶手段に格納しておき、前記デバイスに、ホストから受信したリクエストから前記一定処理によってキーを作成する処理と、上記作成されたキーに対応するバリューを前記記憶手段から検索する処理と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行させるようにしている。

【００２１】

また、請求項３記載のデバイス制御用プログラムは、ホストから受信したリクエストから、予め定められた一定処理によりキーを作成する手段と、前記キーと対応付けられる箇所に、該当リクエストに応じた処理に必要な情報を示すバリューが予め格納されている記憶手段から、上記作成されたキーに対応するバリューを検索する手段と、上記検索されたバリューに基づいて当該リクエストに応じた処理を実行する手段として、デバイスを機能させるようにしている。

【００２２】

従って、リクエストの種類が増えても、リクエストに対する応答処理のステップは増加することがなく、当該応答処理が単純化され、高速に処理することが可能となる。

【００２３】

また、請求項４記載の発明は、請求項３記載のデバイス制御用プログラムにおいて、前記キーはタグ、セット、ブロックの３つのフィールドで構成され、前記記憶手段は、対象リ

クエストに基づく前記セットの値をアドレスとして当該リクエストに基づく前記タグの値が格納されたタグ記憶部と、対象リクエストに基づく前記セットおよび前記ブロックの値をアドレスとして当該リクエストに関する前記バリューが格納されたバリュー記憶部とを有し、上記作成されたキーについて、前記セットの値をアドレスとする前記タグ記憶部の値と、前記タグの値とを比較し、一致する場合に、前記セットおよび前記ブロックの値をアドレスとして前記バリュー記憶部から前記バリューを読み出す手段として、デバイスを機能させるようにしている。

【００２４】

従って、対象リクエストに基づいてキーが作成されると、このキーに対応した特定のバリューが得られる。

【００２５】

【発明の実施の形態】

以下、本発明の構成を図面に示す実施形態に基づいて詳細に説明する。

【００２６】

図１から図３に本発明のデバイス、デバイスの制御方法、デバイス制御用プログラムの実施の一形態を示す。特に本実施形態では、ＵＳＢデバイスに本発明を適用した例について説明する。以下、本実施形態では、ＵＳＢホスト（例えばパーソナルコンピュータなど）を単にホストとも呼び、ＵＳＢデバイス（例えばマウスやキーボードなどのパーソナルコンピュータの周辺機器）を単にデバイスとも呼ぶ。

【００２７】

このデバイス１は、ホストからのリクエストを受信し、該リクエストに応じた処理を実行するものであり、予め定められた一定処理によりリクエストから抽出されるキー７と対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリュー８が格納された記憶手段６を有し、ホストから受信したリクエストから一定処理によってキー７を作成する処理と、上記作成されたキー７に対応するバリュー８を記憶手段６から検索する処理と、上記検索されたバリュー８に基づいて当該リクエストに応じた処理を実行するようにしている。また、本実施形態のデバイス１は、例えば図３に示すように、ホストからのデータを受信する入力部３と、上述した各処理を実行する制御部４（例えばＣＰＵやＭＰＵ等）と、ホストにデータを送信する出力部５とを有している。

【００２８】

また、デバイス１の制御方法は、ホストからのリクエストを受信したデバイス１に当該リクエストに応じた処理を実行させる方法であり、予め定められた一定処理によりリクエストから抽出されるキー７と対応付けられる箇所に、当該リクエストに応じた処理に必要な情報を示すバリュー８を記憶手段６に格納しておき、デバイス１に、ホストから受信したリクエストから一定処理によってキー７を作成する処理と、上記作成されたキー７に対応するバリュー８を記憶手段６から検索する処理と、上記検索されたバリュー８に基づいて当該リクエストに応じた処理を実行させるようにしている。

【００２９】

また、デバイス制御用プログラムは、ホストから受信したリクエストから、予め定められた一定処理によりキー７を作成する手段と、キー７と対応付けられる箇所に、該当リクエストに応じた処理に必要な情報を示すバリュー８が予め格納されている記憶手段６から、上記作成されたキー７に対応するバリュー８を検索する手段と、上記検索されたバリュー８に基づいて当該リクエストに応じた処理を実行する手段として、デバイス１を機能させるようにしている。

【００３０】

ここで、バリュー８が示す「リクエストに応じた処理に必要な情報」とは、例えば当該処理を実行するための命令（コマンド）や、当該処理の実行に用いるデータまたは当該処理の実行に必要となるパラメータ等である。当該命令や当該データ等そのものをバリュー８として設定しても良く、当該命令や当該データ等が格納されているメモリアドレスをバリュー８として設定しても良い。

【００３１】
本実施形態では、ホストから送られてくるデータに対するデバイス１側での高速解析のために、ハッシュ（連想配列とも呼ばれる）を利用する。ハッシュにおいて関連付けられる二つ値を、本明細書ではキー７（Ｋｅｙ）及びバリュー８（Ｖａｌｕｅ）とする。当該ハッシュの仕組は必ずしも限定されないが、処理の単純化および高速化の観点からは、シノニムの発生しないもの（即ち、リクエストとキー７とバリュー８とが１対１対１で対応するもの）が特に好ましい。例えば本実施形態では、ハッシュとして、図１に示す「１　Ｗａｙ　Ａｓｓｏｃｉａｔｉｖｅ（ダイレクトマップとも呼ばれる）」を用いている。
【００３２】
この場合、キー７はタグ７Ａ（ＴＡＧ）、セット７Ｂ（ＳＥＴ）、ブロック７Ｃ（ＢＬＯＣＫ）の３つのフィールドで構成される。記憶手段６は、対象リクエストに基づくセット７Ｂの値をアドレスとして当該リクエストに基づくタグ７Ａの値が格納されたタグ記憶部６Ａ（Ｔａｇ＿ｒｏｍ）と、対象リクエストに基づくセット７ＢおよびブロックＣの値をアドレスとして当該リクエストに関するバリュー８が格納されたバリュー記憶部６Ｂ（Ｄａｔａ＿ｒｏｍ）とを有している。
【００３３】
そして、ホストから受信したリクエストに基づいて作成されたキー７について、セット７Ｂの値をアドレスとするタグ記憶部６Ａの値と、タグ７Ａの値とを比較する。これら２つの値が一致する場合には、セット７ＢおよびブロックＣの値をアドレスとしてバリュー記憶部６Ｂからバリュー８を読み出す。これにより、対象リクエストに基づいてキー７が作成されると、このキー７に対応した特定のバリュー８が得られる。一方、ホストから受信したリクエストが対象リクエストでない場合は、当該非対象リクエストに基づいて作成されたキー７については、当該キー７におけるセット７Ｂの値をアドレスとするタグ記憶部６Ａの値と、当該キー７におけるタグ７Ａの値とは一致しない。この場合、当該非対象リクエストに対応したバリュー８は存在せず、バリュー８の取り出しは行なわない。
【００３４】
例えば本実施形態では、ＳＥＴＵＰトランザクションにおけるリクエスト内容の判別に、上記ハッシュを利用する。当該リクエスト内容は、ＳＥＴＵＰパケット後にホストから受信するデータパケット内のＤＡＴＡフィールドに、記述されている。そこで本実施形態では、ＳＥＴＵＰパケット後にホストから受信するデータパケット内のＤＡＴＡフィールドから、ハッシュのキー７を作成する。
【００３５】
一方、当該リクエストの内容は主に、ディスクリプタの送信または各種ステータスのセットである。この場合、デバイス１では、リクエストがディスクリプタの送信であるか又はステータスのセットであるかを判断し、さらに、ディスクリプタの送信である場合は、どの種のディスクリプタを何バイトの送信サイズで送信するかを判断し、ステータスのセットである場合は、どのステータスをどの値にセットするかを判断する必要がある。そこで、例えば本実施形態では、リクエストがディスクリプタの送信である場合は、送信すべきディスクリプタがセットしてあるデータメモリ６Ｃのアドレスと、ディスクリプタのデフォルトの送信サイズとをバリュー８として設定し、リクエストがステータスのセットである場合は、セットすべきステータスのアドレスや値をバリュー８として設定するようにしている。
【００３６】
以下に、キー７とバリュー８の設定例について説明する。例えば本例では、ＧＥＴ　ＤＥＳＣＲＩＰＴＯＲ（ＵＳＢホストへのディスクリプタ送信）、ＳＥＴ　ＡＤＤＲＥＳＳ（デバイスアドレスのセット）、ＳＥＴ　ＣＯＮＦＩＧＵＲＡＴＩＯＮ（デバイス１のコンフィグレーションのセット）、との３つのリクエストに対応したキー７とバリュー８の設定例を示す。また、デバイス１は、デバイスディスクリプタ、コンフィグレーションディスクリプタ、インターフェースディスクリプタ、エンドポイントディスクリプタの４つのディスクリプタのみ有するものとする。また、デバイス１が持つコンフィグレーション値

は１のみとする。

【００３７】

ホストから送信されるリクエストとＤＡＴＡフィールドのビットパターンは表３の様になる。

【００３８】

【表３】

リクエストタイプ	DATA フィールドパターン							
GET DESCRIPTOR(デバイスディスクリプタ)	10000000	00000110	00000000	00000001	00000000	00000000	xxxxxxxx	nnnnnnnn
GET DESCRIPTOR(コンフィグレーションディスクリプタ)	10000000	00000110	00000000	00000010	00000000	00000000	xxxxxxxx	nnnnnnnn
SET ADDRESS	00000000	00000101	xxxxxxxx	00000000	00000000	00000000	00000000	00000000
SET CONFIGURATION	00000000	00001001	00000001	00000000	00000000	00000000	00000000	00000000

【００３９】

尚、リクエストされるディスクリプタのサイズやホストからセットされるユニークアドレスは、ＵＳＢホストの環境に依存する為、表３では「ｘ」と「ｎ」で示した。また、表３中のディスクリプタのリクエストに、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パターンが無いのは、これらが個々にリクエストされず、コンフィグレーションディスクリプタのリクエストの際にまとめて送信される為である。

【００４０】

次に、表３からキー７を設定する。表３の中で、ビットパターンが確定し尚且つ他のリクエストとビットパターンの異なるビットを取り出すと、表４の様になる。

【００４１】

【表４】

リクエストタイプ	DATA フィールドパターン
GET DESCRIPTOR(デバイスディスクリプタ)	1——- —-0110 ——— ——01 ——— ——— ——— ———
GET DESCRIPTOR(コンフィグレーションディスクリプタ)	1——- —-0110 ——— ——10 ——— ——— ——— ———
SET ADDRESS	0——- —-0101 ——— ——00 ——— ——— ——— ———
SET CONFIGURATION	0——- —-1001 ——— ——00 ——— ——— ——— ———

【００４２】

さらに、表４に基づいてキー７におけるタグ７Ａ（ＴＡＧ）、セット７Ｂ（ＳＥＴ）、ブロック７Ｃ（ＢＬＯＣＫ）の各フィールドを設定すると、例えば表５のようになる。尚、表５中の「ＳＥＴ＿２」はセット７Ｂの２ビット目となることを示しており、「ＳＥＴ＿１：０」はセット７Ｂの１ビット目と３ビット目となることを示している。本例のセット７Ｂは「ＳＥＴ＿２」と「ＳＥＴ＿１：０」の計３ビットで構成する。

【００４３】

【表５】

リクエストタイプ	Key の各フィールド			
	BLOCK	SET_2	TAG	SET_1:0
GET DESCRIPTOR(デバイスディスクリプタ)	1	0	110	01
GET DESCRIPTOR(コンフィグレーションディスクリプタ)	1	0	110	10
SET ADDRESS	0	0	101	00
SET CONFIGURATION	0	1	001	00

【００４４】

以上の様に、ホストから受信したＤＡＴＡフィールドから必要なビットを抽出してハッシュのキー７を作成すれば、各リクエストに応じたバリュー８を得ることができる。

【００４５】

次に、ハッシュにより得られるバリュー８の設定例を示す。ディスクリプタの送信リクエ

ストが発生した場合、送信ディスクリプタの判別が必要である。そこで例えば図２に示す様に、記憶手段６が有するデータメモリ６Ｃに予め各ディスクリプタをセットしておく。そして、ディスクリプタの送信リクエストから作成されるキー７については、送信すべきディスクリプタがセットされているアドレスと、送信サイズとを、ハッシュを利用して得られるバリュー８とする。例えば、バリュー８を４バイトとし、上位２バイトを用いて送信すべきディスクリプタがセットされているアドレスを指定し、下位２バイトを用いて送信サイズを指定する。従って、受信したＤＡＴＡフィールドに対応するディスクリプタがセットされているアドレスと、ディスクリプタの送信サイズを、キー７に基づいて得られたバリュー８の値から取得することができる。

【００４６】

尚、ホストからのリクエストでは、送信するディスクリプタのサイズも指定される為、ハッシュにより得られたデフォルトサイズと、ホストからのリクエストサイズを比較し、小さな方で送信するようにする。これは、ＵＳＢでは、ホストから指定されるデータ長に対してデバイス１が持つデータ長が短い場合にはデバイス１のデータ長で転送することが許されている一方、逆にホストが指定するデータ長がデバイス１が持つデータ長よりも短い場合には、ホストが指定するデータ長で送信を行なう必要があるためである。

【００４７】

ディスクリプタをホストへ送信する場合は、ハッシュを使用して取得したアドレスに基づいて、データメモリ６Ｃからデータを順にロードし、パケットフォーマットに従って送信する。

【００４８】

一方、デバイス１のステータスをセットするＳＥＴ ＡＤＤＲＥＳＳやＳＥＴ ＣＯＮＦＩＧＵＲＡＴＩＯＮの場合は、ディスクリプタのリクエストと区別する為に、例えばディスクリプタのリクエストでは得る事の無いビットパターン（例えばＸ０００００００００やＸ１１１１００００等）をバリュー８に設定する。ハッシュによって得られたバリュー８が当該ビットパターンであれば、ディスクリプタのリクエストでは無いと判断し、例えば当該ビットパターンに応じて各ステータスのセットを行うようにする。

【００４９】

尚、上記の例では、ＧＥＴ ＤＥＳＣＲＩＰＴＯＲ、ＳＥＴ ＡＤＤＲＥＳＳ、ＳＥＴ ＣＯＮＦＩＧＵＲＡＴＩＯＮの３つのリクエストと４種のディスクリプタとに対応させたが、対象リクエストを増やしても、各リクエストのＤＡＴＡフィールドのパターンは一意であり、キー７を構成し直すことで対応することが可能である。

【００５０】

以上のように本発明によれば、ハッシュを用いることにより、デバイス１が備えるディスクリプタが増えても、送信すべきディスクリプタを判断する比較演算命令は増える事は無い。即ち、ディスクリプタの数に応じて比較演算命令を繰り返す必要は無い。従って、ハッシュの導入によりリクエストの内容判別の処理は単純化され、当該処理に要する時間を大幅に短縮できる。これにより、上記判別処理をＣＰＵを用いてソフトウェアにより実現することができる。さらに、比較的性能の低い低速のＣＰＵであっても、高速な処理が可能となることから、デバイス１の低消費電力化が図れる。また、ソフトウェアによる柔軟な制御も可能となる。本発明によって、携帯電話やＰＤＡなどの低消費電力機器に対してもＵＳＢ２．０の高速転送コントローラの導入が可能となる。

【００５１】

【実施例】

本願発明者等は、本発明に基づいてＵＳＢ２．０用コントローラの設計を実際に行なった。以下に、当該設計の内容を実施例として記載する。本実施例では、高速回路部を物理層に分離するＵＴＭＩ（ＵＳＢ２．０ Ｔｒａｎｃｅｉｖｅｒ Ｍａｃｒｏｃｅｌｌ Ｉｎｔｅｒｆａｃｅ）仕様を採用して、ＵＳＢデバイスに実装するＵＳＢ２．０用コントローラ１０の設計を行った。ＵＴＭＩは、ＵＳＢバスから受信したシリアルデータを８ビットにパラレル変換し、トランシーバ側のバスに出力する。またトランシーバ側とのバスクロ

ックにはＵＳＢクロックの８分の１が使用される。このためＵＳＢパケット処理をバイト単位で行える様にコントローラ１０を設計した。

【００５２】

本実施例では、ハイスピード（ＨＳ：４８０Ｍｂｐｓ）モードへの対応を考えて、ホストからのリクエストに対するレスポンス時間の短縮を狙って設計を行なった。このために、コントローラ１０内にハッシュコントローラ１１（ハッシュモジュールとも呼ぶ）を実装し、リクエストに対する応答の高速化を行った。特に、本実施例では、ハッシュによる送信ディスクリプタの判別を導入した。本実施例では、リクエストを受信するＵＳＢデバイス１が、「ｂＲｅｑｕｅｓｔ」と「ｗＶａｌｕｅ」の計４バイトにより送信ディスクリプタを決定する点に注目し、「ｂＲｅｑｕｅｓｔ」と「ｗＶａｌｕｅ」からハッシュのキー７を作成するようにした。キー７によって得られるバリュー８には、各ディスクリプタがセットされているデータメモリ６Ｃ内のアドレスとディスクリプタサイズを設定した。

【００５３】

これにより、デバイス１はリクエストを受信して、ハッシュコントローラ１１にキー７を入力するだけで、瞬時に送信ディスクリプタの情報を取得出来る。この方法を用いれば、デバイス１が持つディスクリプタの数が増えても、その判別処理の命令サイクル数は変化せず、リクエストに対するレスポンス時間に影響を与えない。

【００５４】

さらに、データメモリ６Ｃにはパケットフォーマットに従ってディスクリプタをセットし、送信処理を簡単にした（図５参照）。これによりアドレスとサイズを元に、データメモリ６Ｃから順に値をロードしバスに送信するだけでパケット送信が行える。さらに、データ送信をファームウェアで行うと、多くの命令サイクルが必要なので、ＤＭＡ（Ｄｉｒｅｃｔ　Ｍｅｍｏｒｙ　Ａｃｃｅｓｓ）コントローラ１２を実装した。

【００５５】

ＵＴＭＩインターフェースをターゲットとして設計したデバイスコントローラ１０のブロック図を図４に示す。コントローラ１０の信号線「ＲＸＡｃｔｉｖｅＩｎ」は、ＵＴＭＩの信号線「ＲＸＡｃｔｉｖｅ」と接続されており、信号線「ＲＸＡｃｔｉｖｅ」からＭＰＵ１３（ｓｎｘ２ｐｕ２）に割り込みを発生させる。パケットを受信する場合は、信号線「ｒｃｖ＿ｄａｔａ」から行う。受信したデータは同時にＣＲＣ５、ＣＲＣ１６のモジュール１５，１６にも入力され、エラーチェックが行なわれる。これらのＣＲＣモジュール１５，１６のエラーフラグは、ＭＰＵ１３からアクセス可能となっている。パケット送信は、ＭＰＵ１３がＤＭＡコントローラ１２にリクエストを送る事で行う。デバイス制御用プログラムは例えばファームウェアとしてメモリ２２に格納されている。以下に、ＣＲＣ算出モジュール１５，１６、ハッシュコントローラ１１、ＤＭＡコントローラ１２、ＭＰＵ１３の説明をする。

【００５６】

ＣＲＣ算出モジュール１５，１６は、図６に示すように、シフトレジスタとＸＯＲゲートによって構成されている。各レジスタの初期値は１である。また、受信したパケットの規定フィールドをこの回路に入力し、図６の各レジスタが同図中の下に示す値になれば、ＣＲＣエラーが無い事を示す。設計したＣＲＣモジュール１５，１６は、バイト単位で図６のモジュールと論理的に等しいＣＲＣ計算を行う。図７の様にＣＲＣ５用モジュール１５、ＣＲＣ１６用モジュール１６には、それぞれ５ビット、１６ビットのレジスタがあり、初期値は１である。信号線「ｉｎ」から１バイトデータが入力されると、それに対するＣＲＣの計算結果がレジスタにセットされる。各ＣＲＣモジュールの信号線「ｏｕｔ」にはレジスタの値が出力され、ＣＲＣ１６では上位８ビットを信号線「ｏｕｔ１」から出力し、下位８ビットを信号線「ｏｕｔ０」から出力する。また、ＣＲＣエラーが発生していない時には信号線「ｎｏＥｒｒｏｒ」に１が出力される。

【００５７】

次に、ハッシュコントローラ１１について説明する。ＵＳＢのデバイスリクエストは、ＳＥＴＵＰトランザクションに含まれるデータパケット内のＤＡＴＡフィールドにより決定

される。本実施例では、これらのＤＡＴＡフィールドをハッシュとして利用し、リクエスト判別の高速化を行った。本実施例では、ハッシュの参照に「２　ｗａｙ　ａｓｓｏｃｉａｔｉｖｅ　方式」を採用した（図８参照）。ハッシュコントローラ１１は、ＭＰＵ１３からＤＡＴＡフィールドの上位４バイトを受信すると、それを元に１２ビットのキー７を生成する（図９参照）。キー７は、図８に示すように、タグ７Ａ（ＴＡＧ）、セット７Ｂ（ＳＥＴ）、ブロック７Ｃ（ＢＬＯＣＫ）の３フィールドで構成される。ハッシュコントローラ１１は、セット７Ｂの値をアドレスとして、第一タグ記憶部１７Ａ（Ｔａｇ＿ｒｏｍ００）と第二タグ記憶部１７Ｂ（Ｔａｇ＿ｒｏｍ０１）から値をロードし、タグ７Ａの値と比較を行う。その結果が真（当該比較が一致）であれば、セット７Ｂとブロック７Ｃの値をアドレスとして、第一バリュー記憶部１８Ａ（Ｄａｔａ＿ｒｏｍ００）又は第二バリュー記憶部１８Ｂ（Ｄａｔａ＿ｒｏｍ０１）からロードした値を、ＭＰＵ１３に返す。もし偽（当該比較が不一致）であった場合は、０をＭＰＵ１３に返す。第一バリュー記憶部１８Ａおよび第二バリュー記憶部１８Ｂのデータは、Ａｄｄｒｅｓｓフィールド８ビットとＳｉｚｅフィールド８ビットの計１６ビットで構成されている。これら各フィールドの値は、それぞれデータメモリ６Ｃのアドレスと、送信サイズであり、ディスクリプタの送信時にＤＭＡコントローラ１２が使用する。尚、ハッシュがヒットしなかった場合は、ファームウェアで個別に対応するようにする。

【００５８】

パケットの送信処理はＤＭＡコントローラ１２が行う。ホストに送信するパケットは、予めデータメモリ６Ｃに図５の様にバイト単位でセットする。尚、符号２０で示すデータがパケットＩＤ（ＰＩＤ）であり、符号２１で示すデータがディスクリプタの値である。ＤＭＡコントローラ１２は、ＭＰＵ１３から送信開始アドレスと送信サイズと共に、リクエストを受信すると、データメモリ６Ｃから１バイトずつロードし、ＵＴＭＩに送信する（図１０参照）。コントロール転送のＤＡＴＡフィールドの最大サイズは８バイトである。ＭＰＵ１３から受信した送信サイズが９バイト以上の時は、パケットＩＤ１バイトとＤＡＴＡフィールド８バイトを送信し、未送信データの開始アドレスとサイズを、内部レジスタ「Ｓｔａｒｔ　Ａｄｄｒｅｓｓ　Ｒｅｇ」（符号２３）と「Ｓｅｎｄ　ＳｉｚｅＲｅｇ」（符号２４）に保存する（図１０参照）。再度ＭＰＵ１３から送信リクエストが有れば残りの送信を行う。ＤＭＡコントローラ１２のステート遷移を図１１に示す。ＤＭＡコントローラ１２は、ＭＰＵ１３からリクエストを受信すると、「ｄｍａｒｕｎ１」ステートでＣＲＣ１６用モジュール１６のレジスタを全て１にリセットし、「ｉｄｌｅ」ステートに遷移する。ＵＴＭＩの信号線「ＴＸＲｅａｄｙ」が１になると、ＵＴＭＩが受信可能状態となる為、「ｄｍａｒｕｎ２」ステートに遷移し、メモリのデータをバスに送る。最初に送信する１バイトデータは、必ずパケットＩＤ（ＰＩＤ）であり、この値から送信するパケットの種類を判別し、ＣＲＣ１６の付加が必要かを判断する。ハンドシェイクパケットの場合、ＣＲＣフィールドの付加が必要無いので、ＰＩＤフィールドの送信のみで「ｄｍａｒｕｎ２」ステートを終了し、送信完了をＭＰＵ１３に通知する。データパケットの場合は、ＣＲＣフィールドの付加が必要な為、１バイトのデータフィールドをバスに送信すると同時に、ＣＲＣ１６モジュールへ入力する。ＤＡＴＡフィールドを８バイト送信するか、ＭＰＵ１３から受信したメモリサイズ分送信し終ると、「ｃｒｃｓｅｎｄ１」ステート、「ｃｒｃｓｅｎｄ０」ステートと順に遷移し、ＣＲＣ１６用モジュール１６で計算されたＣＲＣ１６をバスに出力する。ＣＲＣフィールドの送信が完了すると、ＭＰＵ１３にパケット送信完了を通知する。ＵＴＭＩへデータを転送するタイミングは図１２の様になる。今回設計したモジュールは図１２に示すバスクロックの２倍で動作させる事を想定している。そのため、信号線「ＴＸＲｅａｄｙ」からの信号検出後、最初の１バイト（ＰＩＤ）は１クロック送信し、その後のデータは２クロックずつ送信する。

【００５９】

ＭＰＵ１３（ｓｎｘ２ｐｕ２）は１６ビットのオリジナルプロセッサであり、図１３にブロック図を示す。このＭＰＵ１３は、表６に示す様な特徴と命令セットを持つ。

【００６０】

【表６】

ハーバードアーキテクチャ		RISC タイプ	
汎用レジスタ 4 本 ($0,$1,$2,$3)		即値 (I) 8bit	
割り込み 有 (割り込み発生時の PC は repc レジスタに待避)			
3 段パイプライン (if, dec/exe/mem, web)			
命令セット			
ADD	$A $B $C	加算	$A ← $B + $C
AND	$A $B $C	論理積	$A ← $B & $C
NOT	$A $B $C	反転	$A ← ^$B
SR	$A $B $C	右バイトシフト	$A ← 0x00 || $B<15:8>
SLT	$A $B $C	比較	$if($B ≦ $C): $A ← $B $else: $A ← $C
HAS	$A $B $C	HASH コントローラの制御	
LDA	$A $B I	即値加算	$A ← $B + I
LD	$A $B I	メモリ読み出し	$A ← MemoryRead($B + I)
ST	$A $B I	メモリ書き込み	MemoryWrite(($B + I), $A)
IO	$A $B I	UTMI インターフェースからのデータ受信	
BZ	$A $B I	条件分岐	if($A == 0) pc ← $B + I
BAL	$A $B I	強制分岐	$A ← pc + 1 , pc $B + I
INA	$A $B I	割り込みアドレス指定	intadd ← $B + I
RET	X $B I X:Flag(2bit)	割り込み復帰	if(X == 0): IntFlag ← 0, PC ← $B + I else: IntFlag ← 0, PC ← repc
DMA	X $B $C	DMA コントローラの制御	
SET	$A $B X	USB 専用命令 (アドレスチェック or コンペア)	

【００６１】

ＭＰＵ１３は、１６ビット汎用レジスタを４つ持っており、即値は８ビットである。また、このＭＰＵ１３は、「フェッチ」ステージ、「デコードおよび命令実行およびメモリアクセス」ステージ、「ライトバック」ステージの３段パイプラインである。そして、このＭＰＵ１３は、割り込み線を持ち、割り込みのジャンプ先アドレスはＩＮＡ命令を用いて変更可能である。割り込みからの復帰はＲＥＴ命令を用いる。ＲＥＴ命令は表６の様に（＄Ｂ＋Ｉ）のアドレスか、割り込み発生時に待避されたＰＣ（プログラムカウンタ）に、ジャンプ出来る。パケットの受信は常に割り込みからの動作を考えているため、割り込みに関する命令を設計し、ファームウェアの自由度を高めた。右シフト命令（ＳＲ命令）は、扱うデータがバイト単位のため、バイトシフトを採用した。比較命令（ＳＬＴ命令）は、ディスクリプタの送信サイズを決定する為に使用する。ＵＴＭＩからの受信ではＩＯ命令を使用する。このＩＯ命令を実行すると、ＵＴＭＩの信号線「ｒｃｖ＿ｄａｔａ」から１バイトデータを受信する。ＵＴＭＩのデータレシーブタイミングは、図１４の様になる。ＭＰＵ１３はＵＴＭＩとのバスクロックの２倍で動作しているため、ファームウェアで調整し２クロックに１回データを受信する。ＵＴＭＩから受信するデータはバイト単位であるが、ＵＳＢのパケットには２バイトで意味を持つデータが有る。その場合は、表６に示すＩＯ命令のフラグＩを図１５の様に使用して、２バイトに整形する。ＵＳＢデバイス１をバスに接続すると、コンフィギュア時にホストからユニークアドレスを割り当てられる。ホストは、そのアドレスを利用してリクエストのターゲットデバイスを指定する。ＳＥＴ命令はセットされたユニークアドレスを、ＭＰＵ１３の内部レジスタに保持し、以降の通信ではアドレスのコンペアを行う。ファームウェアでは、ＳＥＴ命令を用いて、受信したリクエストが自分に対するものかチェック可能である。ＤＭＡ命令、ＨＡＳ命令は、それぞれデバイスコントローラ１０内にあるＤＭＡコントローラ１２とハッシュコントローラ１１を制御する為にある。

【００６２】

ＤＭＡ命令は、２種類の動作を行う。１つは、送信開始アドレスと送信サイズを指定して、ＤＭＡコントローラ１２にリクエストを発行する。リクエストされたＤＭＡコントローラ１２は、受信したアドレスとサイズを元にパケットを送信する。もう１つは、アドレスとサイズを送らずにリクエストのみ発行する。この場合、ＤＭＡコントローラ１２が使用するアドレスとサイズは、前回のパケット送信後に保存された値を使用する。ＤＭＡ実行中にＬＤ命令、ＳＴ命令の様なメモリアクセス命令がフェッチされると、パイプラインはＤＭＡコントローラ１２から実行完了の通知が来るまでストールする。ただし、メモリアクセスを行なわない命令がフェッチされた場合は、パイプラインはストールしない。

【００６３】

ＨＡＳ命令は、ホストからのリクエストパターンを判別する為に使用し、必要な値をレジスタ「＄２」、レジスタ「＄３」にセットし実行する。「＄２」と「＄３」のビット結合した値がハッシュのキー７となり、ハッシュコントローラ１１は対応したデータを返す。もしハッシュがヒットすると、「＄２」にはデータメモリ６Ｃのアドレス、「＄３」には送信サイズをセットする。ハッシュのキー７がヒットしなかった場合は、「＄２」と「＄３」に０を返す。

【００６４】

次に、ＵＳＢ２．０用コントローラ１０のＵＳＢパケット処理について説明する。トランザクション開始後、最初に受信するトークンパケットに対して、ＭＰＵ１３は図１６の様な処理を行う。この処理では、ハッシュコントローラ１１は使用せずにパケット内の各フィールドから次のチェックを行う。即ち、ＰＩＤフィールドの値から、実行するトランザクションを判別する（図１６のＳ１）。また、ＡＤＤＲフィールドの値と、ホストから割り当てられたユニークアドレスとを比較して自分に対するリクエストかどうかをチェックする（図１６のＳ２）。自分に対するリクエストであれば、ＥＰフィールドの値から、どのエンドポイントに対するリクエストかをチェックする（図１６のＳ３）。尚、自分に対するリクエストでなければ、受信したパケットは無視する（図１６のＳ３’）。ＰＩＤが「ＳＥＴＵＰ」で、且つ自分に対するリクエストである場合、次に受信するデータパケットからデバイスリクエストの種類を判別するようにする。

【００６５】

当該データパケットのＭＰＵ１３による処理は図１７の様になる。パケット内のデータフィールドのうち、ハッシュとして利用するデータをハッシュコントローラ１１に入力する（図１７のＳ４，Ｓ５，Ｓ６）。

【００６６】

ハッシュコントローラ１１では、ＭＰＵ１３からデータを受信すると、図１８の様な処理を行う。始めに、受信したデータからタグ７Ａ（ＴＡＧ）、セット７Ｂ（ＳＥＴ）、ブロック７Ｃ（ＢＬＯＣＫ）　フィールドから成るキー７を生成する（図１８のＳ７）。そして、各フィールドを使用して、タグ記憶部（Ｔａｇ＿ｒｏｍ）１７Ａ，１７Ｂ、バリュー記憶部（Ｄａｔａ＿ｒｏｍ）１８Ａ，１８Ｂからデータをロードする（図１８のＳ８，Ｓ９）。タグ記憶部１７Ａ，１７Ｂからロードしたデータが、タグ７Ａの値と等しければ（図１８のＳ１０；Ｙｅｓ）、ハッシュがヒットしているので、バリュー記憶部１８Ａ，１８Ｂからロードした値をＭＰＵ１３に送信する（図１８のＳ１１）。もしタグ７Ａの値と異なっていれば（図１８のＳ１０；Ｎｏ）、ハッシュがヒットしなかった事を示す「０」を送信する（図１８のＳ１２）。

【００６７】

ＭＰＵ１３では、ハッシュコントローラ１１からの帰り値に基づいて分岐を行う（図１７のＳ１３）。帰り値が０の場合（図１７のＳ１３；Ｙｅｓ）、ハッシュがヒットしなかった事になる（本実施例では、ディスクリプタの送信リクエストではなかった事になる）ので、ホストから指定されたステータスのセット等を行う（図１７のＳ１４）。もし、帰り値が０以外の場合は（図１７のＳ１３；Ｎｏ）、ハッシュがヒットした為、帰り値はデータメモリ６Ｃ内のアドレスと送信サイズである。コントローラ１０は、帰り値を元に、データメモリ６Ｃ内のデータをパケットに整形して、送信する（図１７のＳ１５）。尚、送

信サイズについては、「受信したデータフィールドで指定されているサイズ」と「ハッシュによって取得したサイズ」を比較して、小さい方を使用する。

【００６８】

以上のように構成されるデバイスコントローラ１０の入出力に、ＵＴＭＩの仮想モジュールを図１９の様に接続し、シミュレーションを行なった。尚、シミュレーションにはｓｅｃｏｎｄｓを用いた。また、デバイスコントローラ１０からの出力はＵＴＭＩ仮想モジュールには接続せずに、コントローラ１０からの出力を直接チェックした。バスに流すデータは、ＵＴＭＩの仕様にあるタイミングで行い、パケットを流す際のコントローラ１０の動きと、記述したファームウェアの動作を確認した。

【００６９】

以上の条件でシミュレーションを行った結果、以下のような正常動作が確認出来た。即ち、（１）ＵＴＭＩの信号線「ＲＸＡｃｔｉｖｅ」の信号検出によるＭＰＵ１３の割り込み発生、（２）多重割り込み防止の割り込み制御、（３）ＲＥＴ命令による割り込み処理からの復帰と割り込みフラグの動作、（４）ＩＯ命令を使用した信号線「ｒｃｖ＿ｄａｔａ」からの連続したデータ受信、（５）受信したデータのＣＲＣエラー検出、（６）受信したデータから必要なデータをハッシュコントローラ１１に入力し、ハッシュがヒットした際の正常なメモリのアドレスとサイズの取得、（７）ハッシュがヒットしなかった際の０データの受信、（８）ハッシュコントローラ１１から得られたメモリアドレスとサイズをＤＭＡ命令を使用してＤＭＡコントローラ１２へ出力（信号線「ｘｍｉｔ＿ｄａｔａ」への出力）、（９）ＤＭＡ命令実行中にメモリアクセス命令が発生した場合のパイプラインのストールと、ＤＭＡ命令終了後のストールの解除、（１０）信号線「ＴＸＲｅａｄｙ」からの信号検出によるバスクロックごとのデータ送信、（１１）送信パケットの種類を判別し、正常なサイズを送信。データパケットを送信する場合はＣＲＣ１６の正常な算出と送信、（１２）複数回データパケットを送信した場合のＤＭＡコントローラ１２内のアドレスレジスタのインクリメントと残り送信サイズのディクリメント、（１３）トークンパケット受信時のデバイスアドレスのセット、である。

【００７０】

以上をまとめると、デバイスコントローラ１０では次の動作が可能である。即ち、（１）割り込み発生によるパケットの正常な受信、（２）リクエストを判別して正常なパケットの送信、（３）ホストからセットされるユニークなデバイスアドレスのセットとトークンパケット受信時のアドレスのコンペア、（４）ハッシュコントローラ１１とＤＭＡコントローラ１２を使用したリクエストの迅速な応答、である。

【００７１】

デバイスコントローラ１０をＤＥＭＯ社ｄｅｍｏライブラリで合成した結果は、表７の様になった。なお、入出力のトライステートで発生する遅延は無視した。

【００７２】

【表７】

最大値遅延 [ns]	ゲートサイズ	消費電力
5.33796e+01	3449	6686.3

【００７３】

また、ＡＬＴＥＲＡ社のａｌｔｅｒａライブラリで合成し、Ｑｕａｒｔｕｓ ＩＩ ２．１ でコンパイルした結果を表８に示す。実装のターゲットデバイスはＣＱ出版社のＦＰＧＡ開発パッケージＳｔｒａｔｉｘ評価キットＥＰ１Ｓ１０であり、ＡＬＴＥＲＡ Ｓｔｒａｔｉｘ ＥＰ１Ｓ１０Ｆ７８０Ｃ７ＥＳを搭載している。

【００７４】

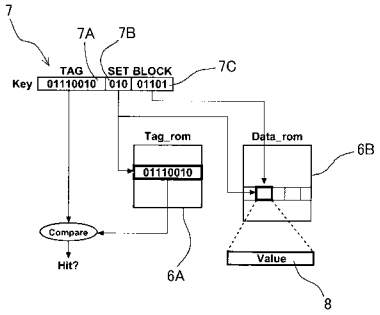
【表８】

Device name	EP1S10F780C7ES
最大動作周波数	149.54 MHz
Total logic elements	248/10,570

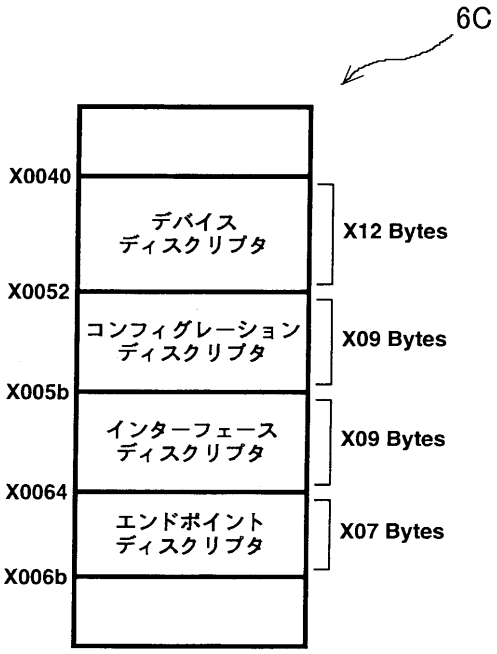
【００７５】
以上のように本実施例では、リクエストに対して少ない命令サイクルで応答出来るコントローラ１０が設計できた。ＭＰＵ１３の動作周波数はＵＳＢバスの４分の１（ＵＴＭＩインターフェースがＭＰＵ１３側に供給するクロックの２倍）を想定しており、ハイスピード（ＨＳ：４８０Ｍｂｐｓ）モードで使用する場合は１２０ＭＨｚでの動作が必要である。設計したＭＰＵ１３は表８の様に最大動作周波数は１４９．５４ＭＨｚとなり、ハイスピードモードへの対応が可能である。
【００７６】
なお、上述の実施形態は本発明の好適な実施の一例ではあるがこれに限定されるものではなく、本発明の要旨を逸脱しない範囲において種々変形実施可能である。例えば、本発明に用いるハッシュは上記に例示した「１　Ｗａｙ　Ａｓｓｏｃｉａｔｉｖｅ」や「２　Ｗａｙ　Ａｓｓｏｃｉａｔｉｖｅ」のようなビット抽出による方法に限らず、シノニムを発生しない完全ハッシュを実現するハッシュ関数を利用したものであっても良い。
【００７７】
また、上述の実施形態ではリクエストの一部を対象リクエストとしてハッシュに対応させたが、正常な全てのリクエストに対応するようにハッシュを構成しても良い。例えば、「ＳＥＴ　ＡＤＤＲＥＳＳ」リクエストのＤＡＴＡフィールドをハッシュに入力すると、自動的にユニークアドレスがセットされ、それ以降のトークンパケットをハッシュに入力すると、自分に対するリクエストであれば真（例えば「１」）を返し、他のデバイスへのリクエストであれば偽（例えば「０」）を返すようにしても良い。
【００７８】
また、本発明は必ずしもＵＳＢデバイスへの適用に限定されず、ホストからのリクエストに応じた処理を実行するデバイスに本発明を適用しても良い。
【００７９】
【発明の効果】
以上の説明から明らかなように、本発明のデバイスおよびデバイスの制御方法およびデバイス制御用プログラムによれば、ホストより受信したリクエストからキーを作成し、このキーに対応するバリューを記憶手段から検索し、このバリューに基づいてリクエストに応じた処理を実行するので、リクエストの種類が増えても、リクエストの内容を判別する比較演算命令を増やす必要はなく、リクエストの内容判別の処理が単純化される。このため、当該処理に要する時間を大幅に短縮できる。これにより、上記判別処理をＣＰＵを用いてソフトウェアにより実現することが可能となる。さらに、比較的性能の低い低速のＣＰＵであっても、高速な処理が可能となることから、デバイスの低消費電力化が図れる。また、ソフトウェアによる柔軟な制御が可能となる。本発明によって、携帯電話やＰＤＡなどの低消費電力機器に対してもＵＳＢ２．０等の高速転送用コントローラの導入が可能となる。
【図面の簡単な説明】
【図１】本発明のデバイスおよびデバイスの制御方法およびデバイス制御用プログラムの実施の一形態を示し、キーとバリューを対応付けるハッシュの一例を示す概念図である。
【図２】リクエストに応じた処理に必要な情報が格納された記憶手段のデータ構造の一例を示す概念図である。
【図３】本発明のデバイスの概略構成の一例を示すブロック図である。
【図４】本発明の一実施例であるＵＳＢデバイスのブロック図である。
【図５】上記実施例に用いられ、リクエストに応じた処理に必要な情報が格納された記憶

手段のデータ構造の一例を示す概念図である。
【図６】上記実施例におけるデバイスが備えるＣＲＣモジュールの回路を示す概念図である。
【図７】上記ＣＲＣモジュールのブロック図である。
【図８】上記実施例のデバイスにおけるキーとバリューを対応付けるハッシュの一例を示す概念図である。
【図９】上記実施例のデバイスにおけるキーの作成方法を示す概念図である。
【図１０】上記実施例のデバイスが備えるＤＭＡコントローラを示す概略構成図である。
【図１１】上記ＤＭＡコントローラのステート遷移図である。
【図１２】上記実施例のデバイスにおけるデータ送信のタイミングを示す図である。
【図１３】上記実施例のデバイスが備えるＭＰＵのブロック図である。
【図１４】上記実施例のデバイスにおけるデータ受信のタイミングを示す図である。
【図１５】上記ＭＰＵが備えるＩＯ命令の動作を示す図である。
【図１６】上記実施例のデバイスにおける処理（トークンパケットの受信処理）の一例を示すフローチャートである。
【図１７】上記実施例のデバイスにおける処理（データパケットの受信処理）の一例を示すフローチャートである。
【図１８】上記実施例のデバイスにおける処理（ハッシュコントローラの処理）の一例を示すフローチャートである。
【図１９】上記実施例のデバイスをＵＴＭＩ仮想モジュールと接続した様子を示すブロック図である。
【図２０】ＵＳＢデバイスの構成例を示す概念図である。
【図２１】ＵＳＢ通信に用いられるパケットのフォーマットを示し、（Ａ）はトークンパケットを（Ｂ）はデータパケットを（Ｃ）はハンドシェイクパケットを示す。
【図２２】ＵＳＢ通信におけるトランザクションを示す概念図である。
【図２３】ＳＥＴＵＰトランザクションにおいてＵＳＢデバイスが行なうＵＳＢホストからのリクエスト内容の判別を、ハッシュを用いずに行なう場合の処理の流れを示すフローチャートである。
【符号の説明】
１　ＵＳＢデバイス（デバイス）
６　記憶手段
６Ａ　タグ記憶部
６Ｂ　バリュー記憶部
７　キー
７Ａ　タグ
７Ｂ　セット
７Ｃ　ブロック
８　バリュー

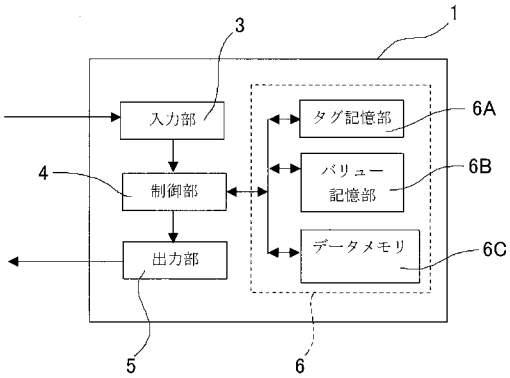
【図１】


7
7A
7B
TAG
SET BLOCK
7C
Key
01110010
010
01101
Tag_rom
Data_rom
6B
01110010
Compare
Hit?
6A
Value
8

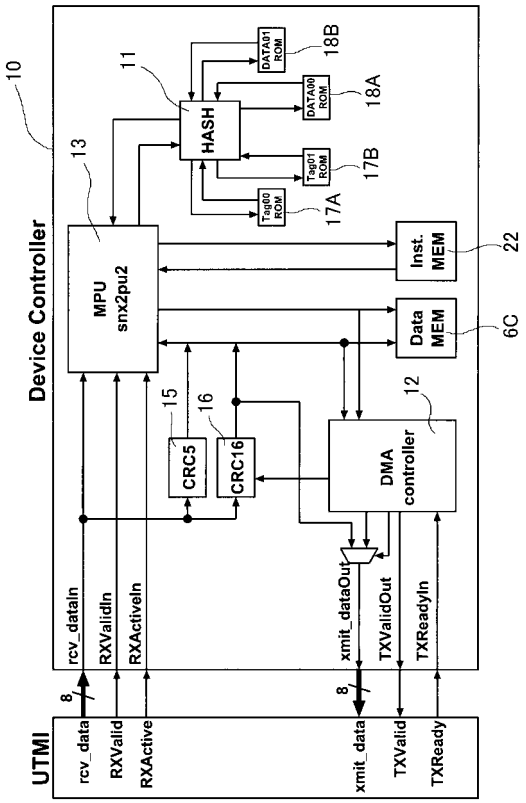
【図２】


6C
X0040
デバイス
ディスクリプタ
X12 Bytes
X0052
コンフィグレーション
ディスクリプタ
X09 Bytes
X005b
インターフェース
ディスクリプタ
X09 Bytes
X0064
エンドポイント
ディスクリプタ
X07 Bytes
X006b

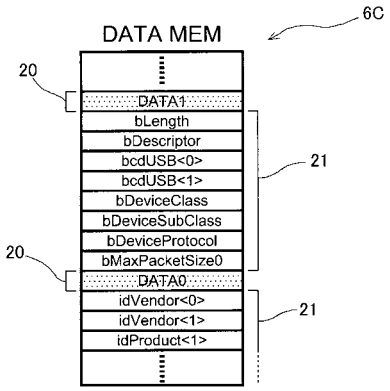
【図３】


1
3
入力部
タグ記憶部
6A
4
制御部
バリュー
記憶部
6B
出力部
データメモリ
6C
5
6

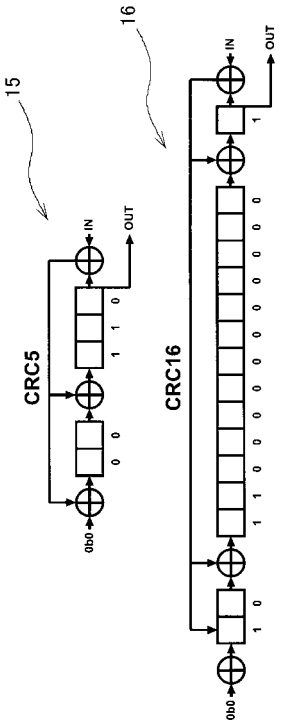
【図４】


10
Device Controller
11
DATA01
ROM
18B
DATA00
ROM
18A
HASH
13
Tag01
ROM
17B
Tag00
ROM
17A
MPU
snx2pu2
Inst.
MEM
22
Data
MEM
6C
12
DMA
controller
15
CRC5
16
CRC16
rcv_datain
RXValidIn
RXActiveIn
xmit_dataOut
TXValidOut
TXReadyIn
8
8
UTMI
rcv_data
RXValid
RXActive
xmit_data
TXValid
TXReady

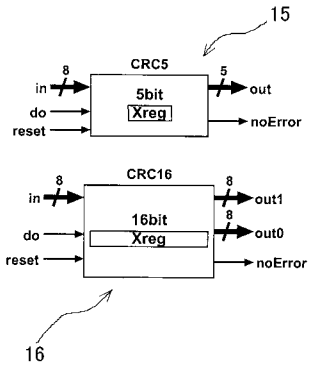
【図５】


DATA MEM
6C
20
DATA1
bLength
bDescriptor
bcdUSB<0>
bcdUSB<1>
bDeviceClass
bDeviceSubClass
bDeviceProtocol
bMaxPacketSize0
20
DATA0
idVendor<0>
idVendor<1>
idProduct<1>
21
21

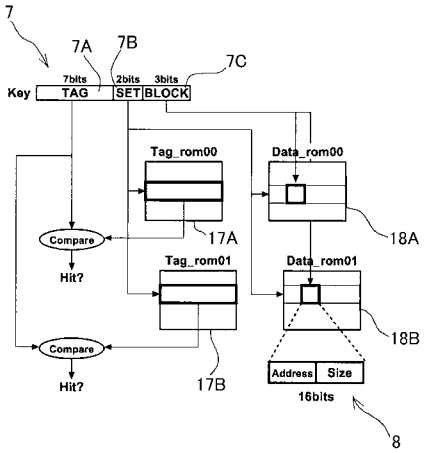
【図６】


16
15
IN
OUT
CRC5
CRC16
0b0
0b0

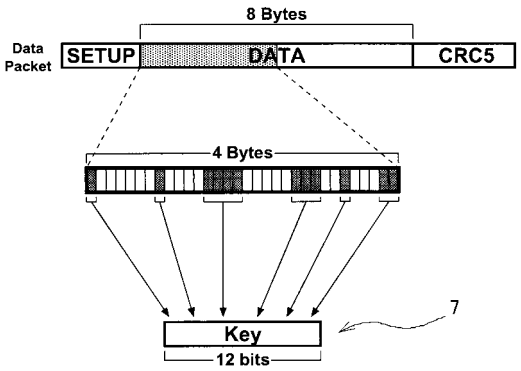
【図７】


15
CRC5
8
in
5bit
Xreg
5
out
do
reset
noError
CRC16
8
in
16bit
Xreg
8
out1
8
out0
do
reset
noError
16

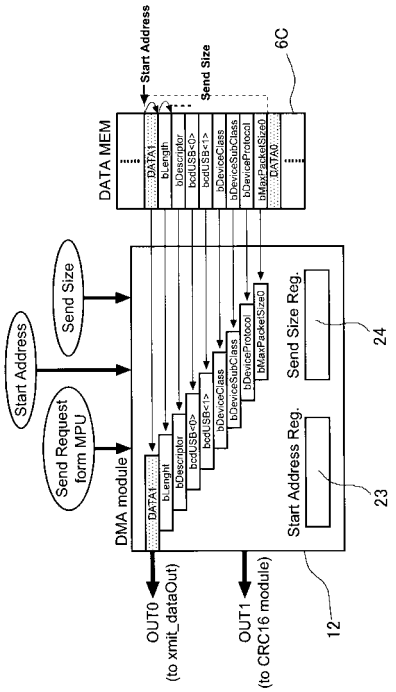
【図８】


7
7A
7B
7C
7bits
2bits
3bits
Key
TAG
SET
BLOCK
Tag_rom00
Data_rom00
18A
17A
Compare
Hit?
Tag_rom01
Data_rom01
18B
Compare
Hit?
17B
Address
Size
16bits
8

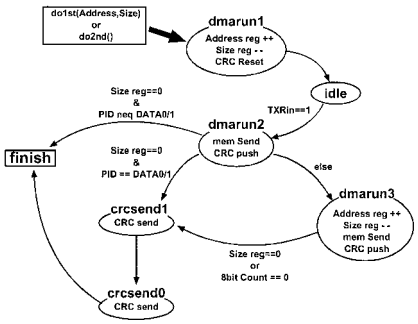
【図９】


8 Bytes
Data Packet
SETUP
DATA
CRC5
4 Bytes
7
Key
12 bits

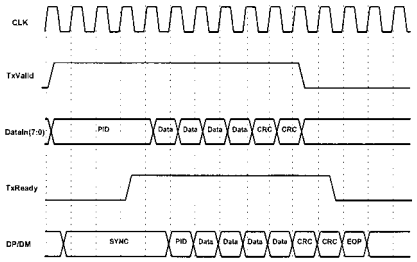
【図１０】



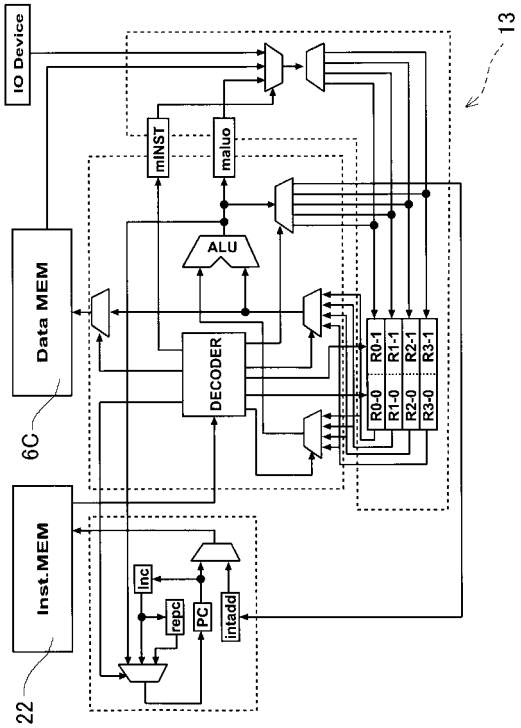
【図１１】



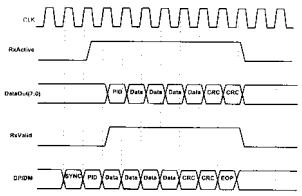
【図１２】



【図１３】



【図１４】

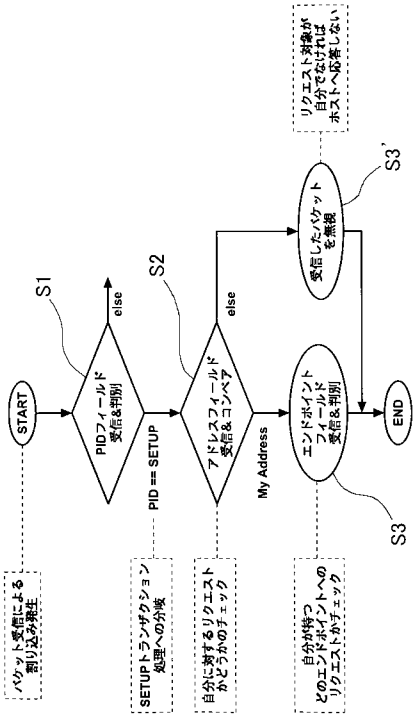


【図１５】

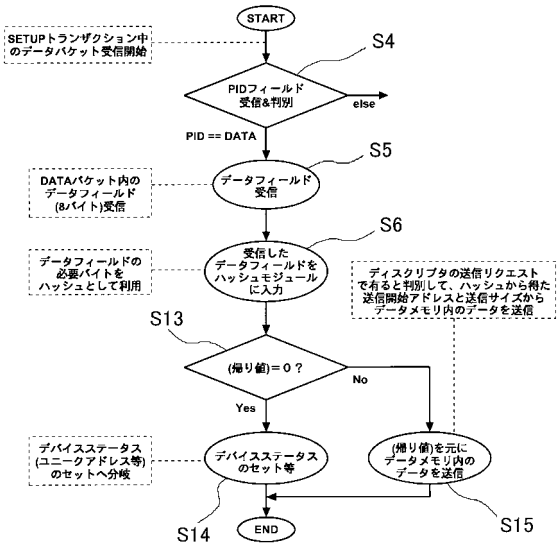
IO $A $B l	rcv_dataIn	X80
IO $1 $0 1	$1 X2a0f	X0080
IO $1 $0 2	$1 X2a0f	X800f
IO $1 $0 3	$1 X2a0f	X2a80

IO $A $B l	rcv_dataIn	X60	X80
IO $1 $0 2 IO $1 $0 3	$1 X2a0f		X8006

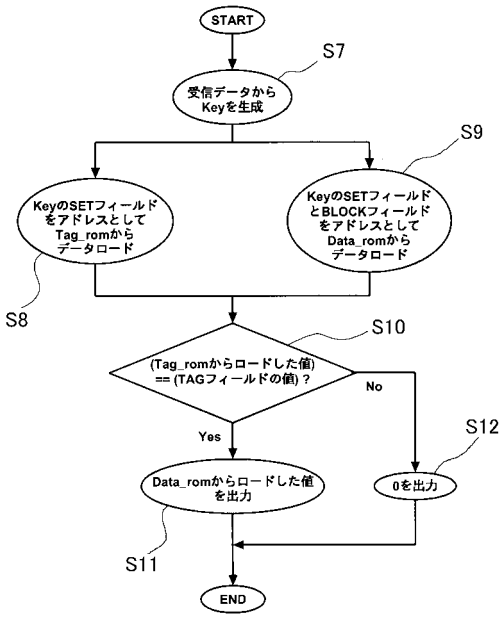
【図１６】



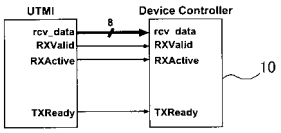
【図１７】



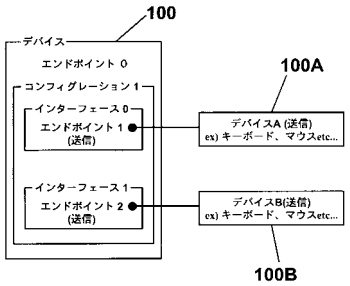
【図１８】



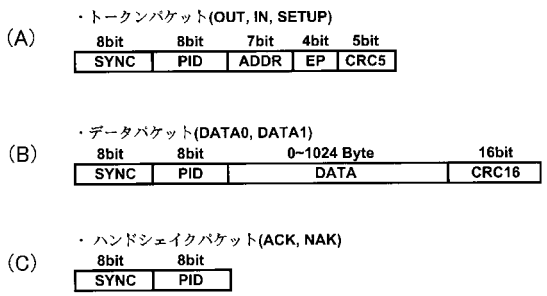
【図１９】



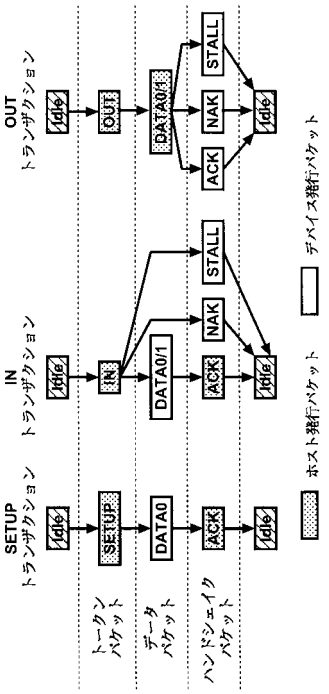
【図２０】



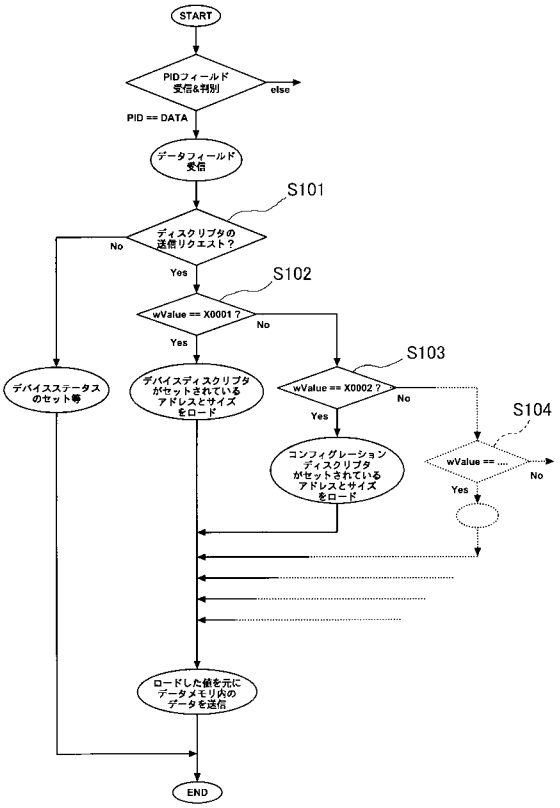
【図２１】


(A)
・トークンパケット(OUT, IN, SETUP)
8bit 8bit 7bit 4bit 5bit
SYNC PID ADDR EP CRC5
(B)
・データパケット(DATA0, DATA1)
8bit 8bit 0~1024 Byte 16bit
SYNC PID DATA CRC16
(C)
・ハンドシェイクパケット(ACK, NAK)
8bit 8bit
SYNC PID

【図２２】


OUT
トランザクション
IN
トランザクション
SETUP
トランザクション
idle
OUT
IN
SETUP
DATA0/1
DATA0
STALL
NAK
ACK
トークン
パケット
データ
パケット
ハンドシェイク
パケット
デバイス発行パケット
ホスト発行パケット

【図２３】


START
PIDフィールド
受信&判別
else
PID == DATA
データフィールド
受信
S101
ディスクリプタの
送信リクエスト？
No
Yes
S102
wValue == X0001？
No
Yes
S103
wValue == X0002？
No
Yes
S104
wValue == ...
No
Yes
デバイスステータス
のセット等
デバイスディスクリプタ
がセットされている
アドレスとサイズ
をロード
コンフィグレーション
ディスクリプタ
がセットされている
アドレスとサイズ
をロード
ロードした値を元に
データメモリ内の
データを送信
END