USB Stack - Design der Systemschnittstelle

Franz Hirschbeck
sifrhirs@stud.uni-erlangen.de

Inhaltsverzeichnis

1 Das Datenmodell des USB

1.1 Die Ziele des Standards

Der USB-Standard wurde entworfen um die Handhabe von Peripheriegeräten für den Benutzer zu vereinfachen. Probleme wie fehlende freie Steckplätze und Adapter oder das komplizierte Einstellen von Übertragungsraten sollten umgangen werden. Zusätzlich sollte der neue Bus über eine Stromversorgung kleinere Geräte komplett versorgen können und dabei hotplug-fähig sein.

1.2 Die Topologie der Endgeräte

Die einfache Erweiterbarkeit um neue Endgeräte, die auch Function genannt werden, wird dadurch erreicht, dass jeder freie USB Steckplatz mit Hilfe eines Hubs um mehrere Steckplätze erweitert werden kann. Die dadurch entstehende hierarchische Struktur wird durch den am Host befindlichen Root-Hub-Controller und den USB-Treiber von den Endgerätetreibern verborgen. Um eine komplizierte Koordinierung zu vermeiden wird ein Single-Master-Bus verwendet, d.h. Übertragungen werden ausschließlich vom Host initiiert. Die Functions verhalten sich passiv und senden nur auf Kommando.

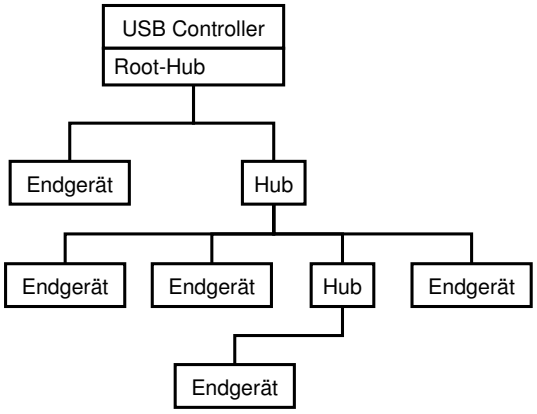

Abbildung 1: physikalische Topologie des USB: An jedem Port können mit Hilfe eines Hub mehrere Geräte angeschlossen werden.

Hubs stellen dabei nicht nur die physikalische Verbindung zur Verfügung, sondern melden dem Host auch, wenn sich Veränderungen an Ports ergeben haben. Sie können auch zusätzlich Strom für die angeschlossenen Geräte liefern, sofern sie selbst an ein Netzteil angeschlossen sind. Würde ein Port zu viel Strom brauchen, so dass andere Geräte beeinträchtigt würden, wird ein Gerät abgelehnt.

Um preisgünstige Geräte anbieten zu können kennt der USB Standard zwei Gerätetypen. Full-Speed-Devices beherrschen komplexere Übertragungsmöglichkeiten und können das volle Spektrum des USB-Standards ausschöpfen.

Low-Speed-Devices sind dabei sehr eingeschränkt. Sie arbeiten mit niedrigerer Frequenz auf der Leitung, haben nur zwei Übertragungsarten und haben kleine Buffer, aber dafür kostengünstiger in der Herstellung.

So können bis zu 127 verschiedene USB Geräte an einem Bus gleichzeitig betrieben werden.

1.3 Konfigurationen,Interfaces und Pipes

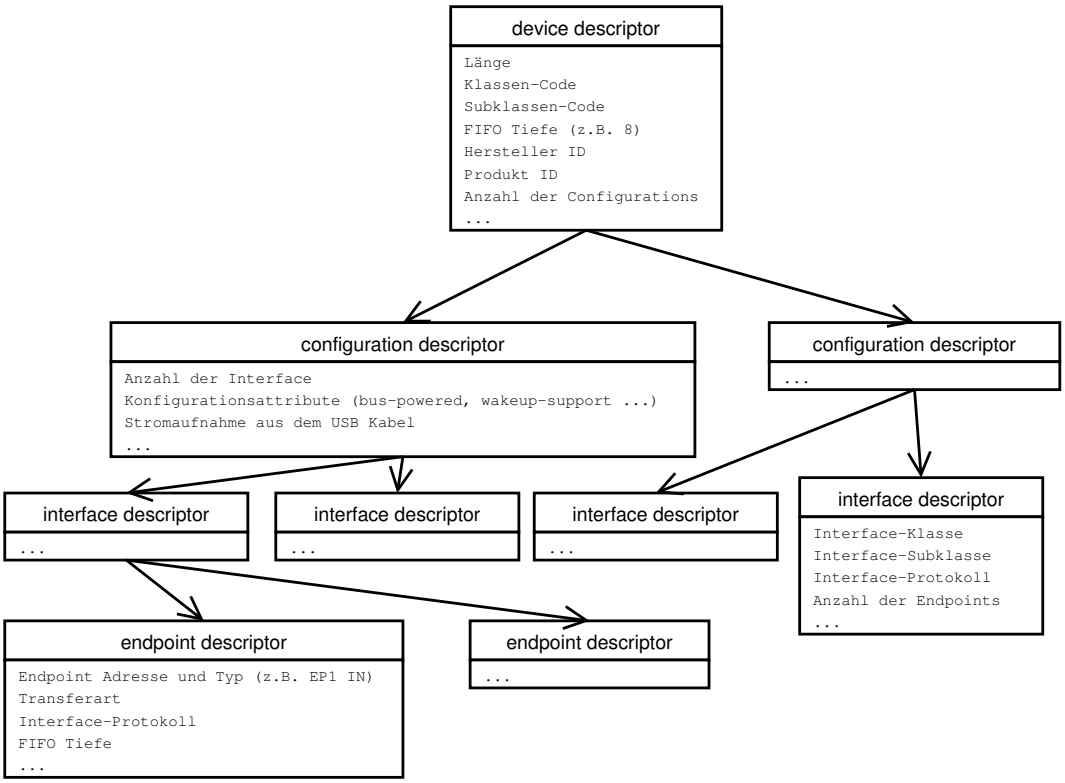

Abbildung 2: Eine (nicht vollständige) Beispielhierarchie der Descriptoren einer Function.

Um Kommunizieren zu können haben Endgeräte verschiedene logische Kanäle, die Pipes genannt werden und von 0 bis 15 durchnummeriert werden. Auf der Endgerätseite gibt es für jede Pipe einen Endpoint. Der Endpoint 0 ist dabei für Konfigurationszwecke reserviert. Diesen Endpoint muss jedes Endgerät besitzen. Pipes sind unidirektional, entweder senden sie Daten vom Gerät zum Host oder umgekehrt. Eine Ausnahme besteht dabei in den Setup-Pipes für Controltransfers. Eine Kommunikation direkt zwischen 2 Functions ist also nicht möglich.

Die Endpoints werden in Interfaces und diese wiederum in Configurations gegliedert und entsprechende Datenstrukturen angelegt, die Descrip-

tor genannt werden. Jedes Gerät besitzt einen Devicedescriptor, in dem Informationen zum Identifizieren und Klassifizieren stehen, mit denen der passende Treiber ausgewählt werden kann. Für jeden Betriebszustand hat ein Gerät einen Configurationdescriptor, der Informationen zum Stromverbrauch enthält. Einer Configuration ist immer mindestens ein Interfacedescriptor zugewiesen, der einen Übertragungsmodus und somit einer Menge von Transferdescriptoren wiederspiegelt.

1.4 Die Transfertypen

Verschiedenartige Geräte stellen verschiedene Ansprüche an ihre Übertragungsschnittstelle, wenn diese nicht erfüllt werden können, kann das Gerät nicht ordnungsgemäss arbeiten. Es gibt Geräte, die garantierte Übertragungsraten benötigen, da sie selbst einen konstanten Datenstrom erzeugen oder empfangen. Mit Zwischenspeichern können zwar zwischenzeitliche Unterbrechungen überbrückt werden, dennoch ist eine Flusssteuerung notwendig. Ein zweites Problem ist die Reaktionszeit, also die Zeit in der eine Information, etwa eine Mausbewegung auch wirklich bei dessen Treiber ankommt. Die dritte Forderung, die eigentlich mehr allgemein an den Bus gestellt wird, ist eine effiziente Ausnutzung der physikalischen Leitung und schnelle Übertragung der Daten.

Für jeder dieser Kriterien gibt es beim USB einen eigenen Übertragungsmodus, die in den folgenden Unterkapiteln genauer betrachtet werden sollen.

Die Übertragung zu den Functions erfolgt dabei in 1-ms Frames. Innerhalb dieser Frames werden im Normalfall mehrere Geräte nacheinander angesprochen und es finden verschiedenartige Übertragungen statt. Die Reihenfolge und Übertragungsbedingungen muss der Hostcontroller, bzw. dessen Treiber festlegen.

1.4.1 Der Control-Transfer

Um die Geräte steuern, konfigurieren und anfangs enummerieren zu können gibt es den Control Transfer. Dabei wird ein Kommando und eventuell Daten als Option bzw. Antwort übertragen. Es gibt eine Reihe von Requests, die in der USB Spezifikation definiert sind, zusätzlich dazu kann es noch gerätespezifische Requests geben.

1.4.2 Der Interrupt-Transfer

Der Interrupttransfer hat trotz der Namensgleichheit nichts mit Prozessorinterrupts zu tun. Vielmehr handelt es sich um einen Transfermodus, bei dem der Host in regelmässigem Abstand einen Transfer initiiert. Hat das Gerät neue Daten werden diese, häppchenweise übertragen. Die Frequenz der Abfragen bestimmt das Gerät im Transferdescriptor als ein Vielfaches

einer Millisekunde bzw. eines Frames. Bei Lowspeed-Devices ist diese Transfertypen die einzige um Daten zu übertragen.

1.4.3 Der Isochronous-Transfer

Ein isochroner Transfer wird benutzt, wenn eine konstakte Übertragungsrate erforderlich ist. Es wird garantiert, dass in jedem Frame für eine bestimmte Zeit Daten übertragen werden können. Die bei den anderen Transfertypen übliche Fehlerkorrektur ist hier nicht möglich.

1.4.4 Der Bulk-Transfer

Die restliche Zeit, die nach den anderen Übertragungsraten noch übrig bleibt soll nicht ungenutzt bleiben. In den Bulk-Transfers werden alle Daten übertragen, die nicht dringend sind, wie Druckaufträge, Festplattenzugriffe etc. So wird der Bus immer optimal ausgelastet.

1.5 Das Hot Plug'n'Play

Beim Anstecken eines neuen Gerätes hat ein Gerät die Adresse 0. Danach bekommt es in einem Vorgang der Enumerieren genannt wird eine neue freie Adresse vom USB Treiber zugewiesen und wird initialisiert. Damit der USB Treiber das Einstecken mitbekommt melden Hubs, wenn sich an ihren Ports etwas verändert über eine eigenen Pipe.

2 Die Implementierung des USB-Stack

Die Implementierung des USB-Stack teilt sich in zwei Aufgaben, zum einen müssen die USB Geräte verwaltet werden und entsprechende Treiber geladen oder entladen werden und zum anderen muss diesen Endgerätetreibern eine Schnittstelle angeboten werden, so dass er unabhängig von anderen Geräten arbeiten kann. Das Kernstück dabei ist der USBDeviceManager, der als Portal verwendet werden kann und sowohl Geräte als auch Treiber verwaltet. Bei ihm kann sich ein Treiber mit der zuständigen Produkt- und Hersteller-ID registrieren. Beim Anstecken eines neuen Geräts wird dieses enumeriert, geprüft ob genug Strom zur Verfügung steht, die Deskriptoren augelesen und schlieslich der Treiber geladen, der zu diesem Gerät registriert wurde. Falls kein Treiber registriert ist, wird das Gerät als unbekanntes Gerät nur enumeriert.

2.1 Die Schnittstelle für Treiber

Ein Endgerätetreiber kann in eine von USBDevice abgeleiteten Klasse realisiert werden. Bei Starten des USB-Controller-Treibers wird dann auch eine

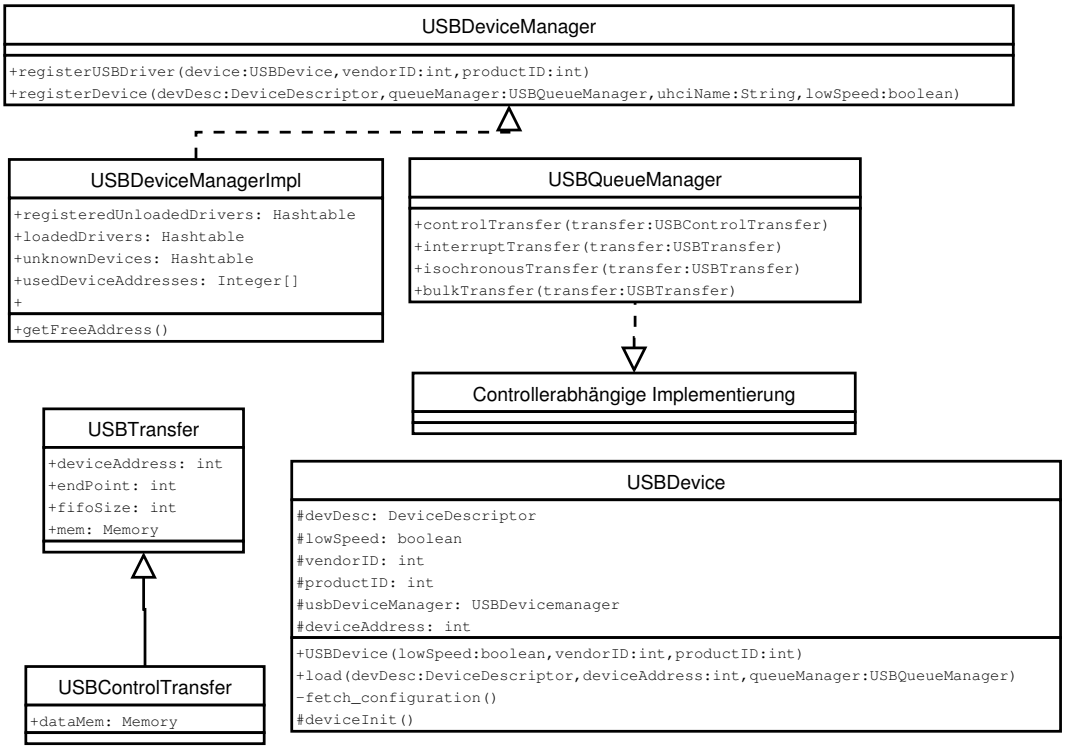

Abbildung 3: Die wichtigsten Klassen, Variablen und Methoden der USB Implementierung

Instanz dieser Klasse erzeugt. Der Konstruktor registriert dabei die Treiber im USBDeviceManager.

Beim Laden bekommt der Treiber die Informationen über das enumerierte Gerät, die er braucht um es anzusprechen. Das sind der DeviceDescriptor, aus dem der Treiber entnehmen kann, welche Version des Geräts vorliegt, die Geräteadresse und natürlich den USBQueueManager, mit dem Transfers gestartet werden können. Die anderen Descriptoren stehen dann bereits in member Variablen, da sie beim Enumerieren geholt werden. Dann kann der Treiber eine Configuration und ein Interface wählen und bekommt so die Endpointdescriptoren und kann damit die Pipes ansprechen.

Um einen Transfer vorzunehmen muss der Treiber die Daten, die verschickt oder empfangen werden sollen in ein Memory Objekt packen. Dieses Objekt und die Daten für den Transfer werden in ein Objekt der Klasse USBTransfer gekapselt und mit der passenden Methode des QueueManager übertragen.

Bei einem Control-Transfer wird ein Befehl geschickt, dem noch eine Datenübertragung folgen kann, aber nicht muss. Der Befehl, der vom Endgerätetreiber in das Memoryobjekt mem geschrieben sorgt dafür, dass das Gerät eine anschliesende Übertragung erwartet. Da diese vom Controller-

treiber eingeleitet werden muss, ist dem USBControlTransfer das Memoryobjekt memData hinzugefügt, dass bei ausbleibender Übertragnung null, ansonsten das Memory in dem die übertragenen Daten stehen. Es können sowohl Daten geschickt als auch empfangen werden, dementsprechend wird memData bereits vor dem Transfer beschrieben oder nachher ausgelesen.

2.2 Das Interface zum Controllertreiber

Damit nicht für jeden Controllertreiber die USB Funktionalität neu implementiert werden muss, wird hier das Interface USBQueueManager benutzt. Die Implementierung des QueueManager, der Transferaufträge annehmen kann und in Datenstrukturen für den Controller umsetzt, sowie diesen steuert ist in einem eigenen Controllertreiber ausgelagert. Der Controllertreiber muss beim Starten auch eine Instanz des USBDeviceManager starten. Der USBQueueManager kann auch als Portal benutzt werden.

2.3 Ausstehende Entwicklungen und Implementierungen

Die Implementierung ist allerdings noch in einem sehr frühen Stadium der Entwicklung. Folgende Punkte sind noch nicht implementiert, bzw. für die Zukunft geplant:

- Unterstützung für mehrere Controller, die jeweils einen eigenen QueueManager besitzen.
- Ein Treiber sollte mehrfach instanzierbar sein, so dass er mehrere gleichartige Geräte handhaben kann, auch an unterschiedlichen Controllern.
- Bisher kann ein Treiber auch fremden Geräten Daten schicken, da hier bislang kein Schutzmechanismus eingesetzt wird. Ein Treiber sollte aber nur Pipes zu dem eigenen Gerät aufbauen können.
- Die Aufrufe des QueueManagers initieren lediglich die Kommunikationsprozedur, die dann in einem anderen Thread abläuft bzw. vom Controller vollzogen wird. Der Treiber erhält bislang keinerlei Rückmeldung, wann der Transfer abgeschlossen ist. Im Idealfall werden die Aufrufe solange blockiert bis Daten erfolgreich übertragen wurden. Der Misserfolg einer Übertragung sollte dem Programm über einen Rückgabewert mitgeteilt werden.
- Der einzige Transfer, der bislang funktioniert ist der ControlTransfer. Für die anderen Transfertypen ist noch nicht sinnvoll festgelegt, wie ein QueueManager Aufruf zu realisieren ist. Bei Interrupttransfers, die in regelmässigen Abstand Geräte abfragen können ist beispielsweise vorstellbar, dass solangen blockiert wird, bis eine Abfrage erfolgreich war.

Der Isochronous-Transfer benötigt noch einen Synchronisationsmechanismus, da Framelänge und damit die exakte Datenübertragunsrate geringfügig variieren können.

3 weiterführende Informationen und Quellen

Literatur

[1] Dipl.-Ing. Hans Joachim Kelm. „USB 1.1, Universal Serial Bus“. Franzis Verlag, 2000, Poing.

[2] Die USB 1.1 Spezifikation:
http://www.usb.org/developers/docs/usbspec.zip
veröffentlicht von Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, NEW Corporation am 23.9.1998

[3] Die USB-Stack Implementierung für jx im Rahmen des AKBP II