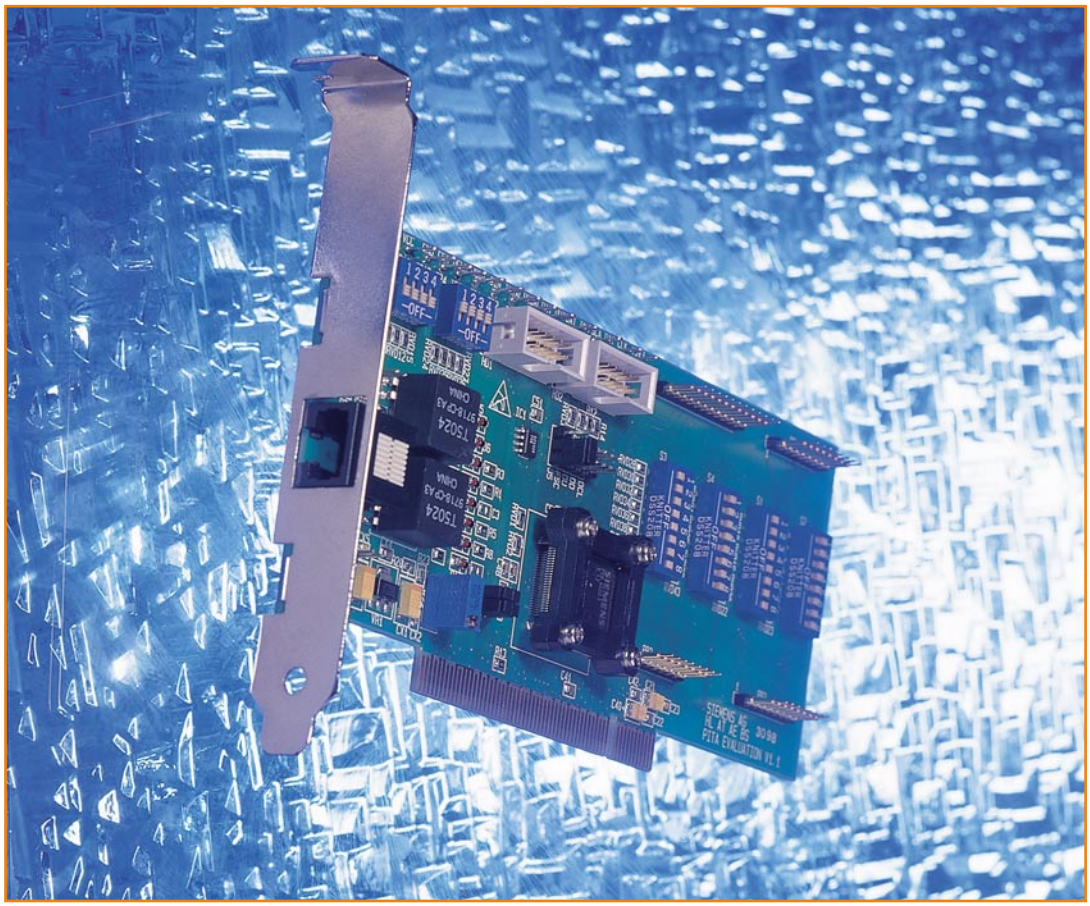

PCI-Grundlagen Teil 3

Nachdem wir im zweiten Teil den PCI-Controller und seine Einsatzmöglichkeiten sowie die Entwicklungsumgebung von der Hardwareseite her behandelt haben, beschäftigen wir uns jetzt mit den Werkzeugen und Hilfsmitteln zur Entwicklung des Treibers, der das Ansprechen der PCI-Karte erst ermöglicht.

Über den Treiber an die Hardware

Als Schaltstelle zwischen Hardware und Anwendung bildet der Gerätetreiber ein zentrales Element, er ermöglicht es, die Hardware „anzusprechen", d. h., ihre Funktionen zu aktivieren.

Wir wollen uns in diesem Artikel mit den Grundzügen der Treiberentwicklung beschäftigen, die zugehörigen Werkzeuge und Hilfsmittel kennenlernen und am Schluß sehen, wie eine fertige Applikation aussehen kann.

Natürlich ist es im Rahmen einer solchen Artikelserie, die ja lediglich einen Überblick geben soll, nicht möglich, auf die Tiefen der eigentlichen Treiberentwicklung einzugehen. Vielmehr soll das Gefühl vermittelt werden, was es bedeutet, Hardware in einem PC anzusprechen.

Wer jedoch nach dem Lesen „Lust auf mehr" bekommen hat, der sei auf das bereits erwähnte Buch „Hardware programmieren unter Windows" von Logemann/Olfs/Reiners verwiesen, das Details zu diesem Thema enthält und sich auf diese Artikelserie bezieht.

Treiberprogrammierung

Wer eine möglichst schnelle und kostengünstige Arbeitsplattform für das Ansprechen der Hardware benötigt, wird nicht darum herumkommen, Treiber für seine Hardware zu entwickeln.

Treiber sind kleine Programme, die in ein System eingebunden sind und sich darum kümmern, daß das, „was ein Programm von einer Hardware möchte", auch dort ankommt (und umgekehrt) – siehe Abbildung 11. Sie bilden also die Schnittstelle zwischen Soft- und Hardware.

Wie schon gesagt: Treiber sind Bestandteile des Betriebssystems und werden vom Benutzer eigentlich nie wirklich gesehen. Sie besitzen solch kryptische Endungen wie *.SYS, *.VxD oder *.DLL, je nachdem, welche Aufgabe sie haben und in welchem Betriebssystem (Windows NT oder Windows 9x oder 3.1) sie benutzt werden.

An dieser Stelle sei schon jetzt gesagt: Wer Treiber programmieren will, der sollte auf eine saubere Programmierung achten. Denn jedesmal, wenn ein Treiber in seiner Umgebung nicht exakt funktioniert, kann man davon ausgehen, daß das System neu gebootet werden muß, da der Treiber im Gegensatz zum Anwendungsprogramm direkt im und nicht „auf" dem System arbeitet. Hier sieht man schon den Unterschied zur Anwendungsprogrammierung.

In Abbildung 11 werden die verschiedenen Ebenen deutlich erkennbar. Im dort dargestellten User-Mode (oder auch Ring 3

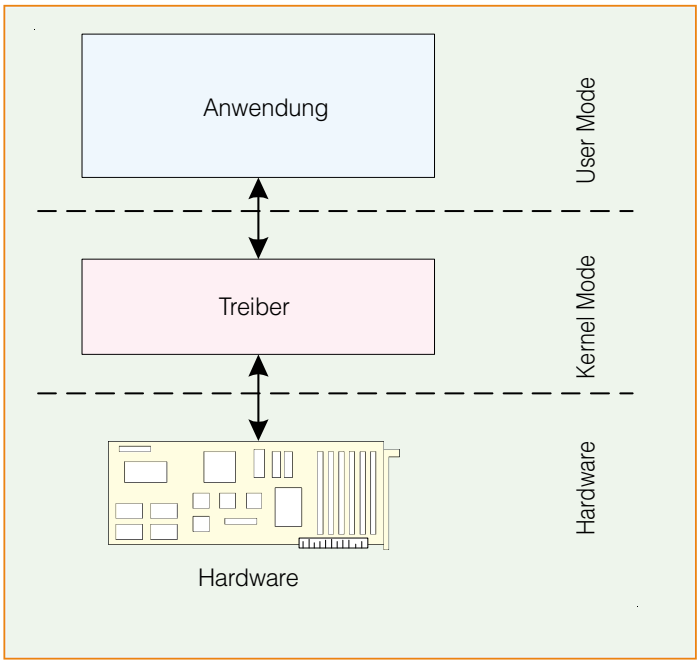

Bild 11: Der Treiber dient als Schnittstelle zwischen Hardware und Anwendungssoftware, das Bild zeigt die verschiedenen Software-Ebenen für eine Applikation

genannt) kann der Programmierer sich noch am meisten „erlauben". Hier werden noch sehr viele Fehler vom System abgefangen und können ihm so (fast) nichts anhaben.

Im Kernel-Mode (oder auch Ring 0) sieht das Ganze schon anders aus.

Hier besitzt man sämtliche Privilegien, über die auch das Betriebssystem verfügt. Es ist also einfacher, von hier aus das System „lahmzulegen". Vom Kernel-Mode aus ist es nun möglich, die Hardware direkt anzusprechen.

Für die Programmierung eines Treibers als Schnittstelle zwischen Hardware und Anwendungssoftware sind eine Reihe professioneller Entwicklungswerkzeuge sowie eine „saubere" Programmierung notwendig, da der Treiber tief ins jeweilige Betriebssystem eingreift.

Entwicklungswerkzeuge

Windows Gerätetreiber werden nicht nur durch das allseits bekannte „Makefile" generiert (compilieren und linken), sondern auch mit Hilfe zusätzlicher Includes und anderer Werkzeuge.

Zu einer Windows-Gerätetreiber-Entwicklungsumgebung gehören noch diverse andere Tools, welche da wären:

- Software Development Kit (SDK), derzeit „Platform SDK"
- Driver Development Kit (DDK) – je nach System
- Ein Editor und ein Compiler – da wird sowieso das SDK und DDK benötigen und diese von Microsoft kommen, ist es sinnvoll, hierfür das letzte Developer Studio (derzeit Version 6.0) von Microsoft einzusetzen.
- Ein Low-Level-Debugger (z. B. SoftIce von CompuWare oder den Debugger von Microsoft namens „WinDebug").

Der größte Unterschied zwischen diesen beiden Debuggern ist der Preis: WinDebug kostet „nichts", weil es mit dem DDK geliefert wird - im Gegensatz dazu bietet SoftIce mehr Komfort und stellt unter Treiberentwicklern mehr oder weniger ein Standard-Werkzeug dar.

Von diesen Tools bildet das DDK die eigentliche Schnittstelle zum Betriebssystem.

Es enthält eine Ansammlung von Funktionen, Strukturen usw., mit denen man das System abstrahieren und auf dessen Funktionalitäten zugreifen kann.

Außerdem enthält das DDK viele nützliche Beispieltreiber und Tools, die dem Entwickler als Basis für eigene Entwicklungen dienen.

Ausführliche Informationen zu diesem Thema findet man auf der Homepage von Microsoft:

www.microsoft.com oder
www.microsoft.com/hwdev

(wobei die letztere Adresse eher auf Software-Entwickler für Hardware abzielt). Wenn man noch ein wenig komfortabler arbeiten möchte und nicht ständig das Rad neu erfinden will, sollte man überlegen, ob man nicht irgendwelche existierenden Bibliotheken von anderen Anbietern für die eigene Lösung einsetzen kann.

Ein weit verbreitetes Tool in diesem Zusammenhang ist „Driver:: Works" - die beiden Doppelpunkte stimmen!

Dieses Produkt kommt von Vireo-Software und ist bei CompuWare erhältlich:

www.compuware.com

Driver:: Works ist ein sehr mächtiges Tool und enthält von Haus aus schon einige wichtige Treiberbeispiele, auf die man seine eigenen Entwicklungen aufsetzen kann.

Ein weiterer Vorteil dieses Werkzeugs ist, daß man objektorientiert in C++ programmieren kann. Das ist äußerst sinnvoll, um eine leichte Wiederverwendbarkeit der einzelnen Module (Klassen) zu gewährleisten.

Wenn einmal ein vernünftig durchdachter Rahmen für einen Treiber existiert, sollte es dann einfach sein, die dort implementierten Klassen abzuleiten und an neue Anforderungen anzupassen.

Debuggen von Treibern

Das Debuggen von Treibern gestaltet sich ein wenig schwieriger als das von Anwendungsprogrammen. Im Gegensatz zu Anwendungsprogrammen liegen die Treiber „tiefer" im System. Mit Hilfe der oben angegebenen Debugger ist es möglich, Debugausgaben zu erzeugen.

Weiterhin kann man auch einen Treiber Schritt für Schritt debuggen.

Bei den meisten Treibern ist dies aber oft nicht richtig möglich, weil in einem Treiber häufig zeitkritische Abläufe stattfinden - die also eigentlich nicht zu unterbrechen sind.

Wie schon gesagt, besitzen die beiden genannten Debugger sowohl Vor- als auch Nachteile.

Der Microsoft-Debugger ist zwar kostenlos – aber man benötigt zwei Computer, die über die serielle Schnittstelle per Nullmodem-Kabel miteinander verbunden sind.

Mit SoftIce ist es möglich, einen Treiber auf einer Maschine zu debuggen.

Im Endeffekt muß jeder Entwickler selbst entscheiden, mit welchem Tool er lieber arbeitet.

Am Schluß sei jedoch bemerkt: Ein Debugger ist das wichtigste Tool bei der Soft-

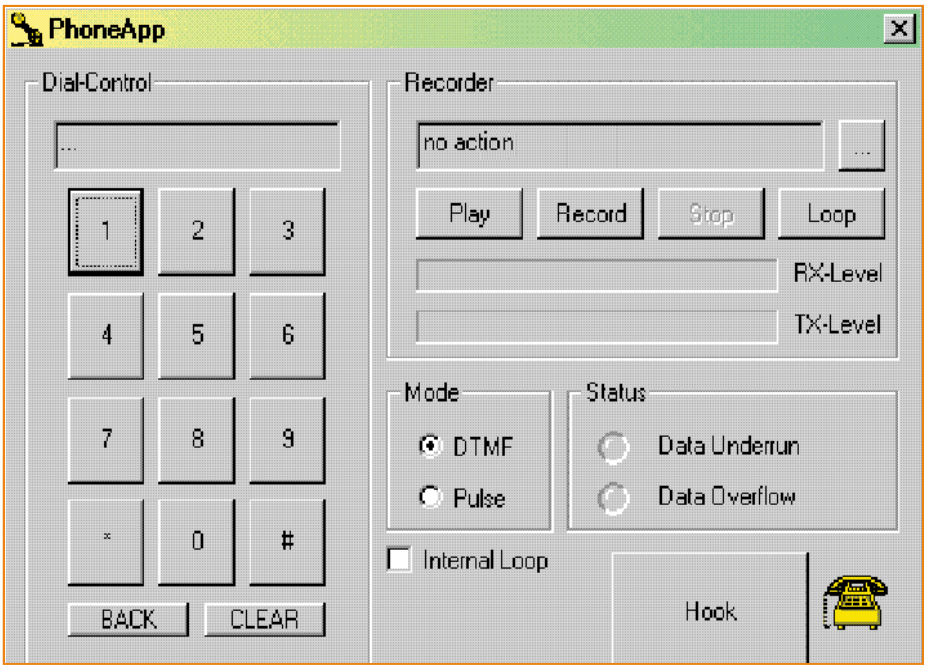

Bild 12: Screenshot der PCI-Board-Applikation

ware-Entwicklung, auf das niemand verzichten kann.

Anwendung

Um die Treiberentwicklung ein wenig schmackhafter zu machen, wollen wir zum Abschluß einmal ein kurzes Szenario durchspielen, das zudem aus dem wirklichen Leben gegriffen ist, da sowohl die Hardware als auch die Software erhältlich sind:

Das Board ist ein PCI-Board mit einem PCI-Controller und einem analogen Frontend, und die Software ist der CD entnommen, die zum bereits erwähnten Buch „Hardwareprogrammierung unter Windows" über ELV erhältlich ist.

Ein Screenshot dieser Applikation ist in Abbildung 12 zu sehen.

Man steckt also die PCI-Karte, die die Hardware inkl. analogem Frontend (analoges Telefoninterface) enthält, in seinen Rechner und verbindet sie mit der Telefonleitung (natürlich nur, wenn die Karte für den Anschluß an das öffentliche Telefonnetz zugelassen ist) und erhält so zunächst die fertige Hardware-Konfiguration.

Diese Applikation kann natürlich nur soviel an Funktionalität leisten, wie im Treiber implementiert ist. Die eigentliche Funktionalität und Intelligenz liegt also demnach in diesem Treiber.

Abbildung 13 zeigt die gesamte Umgebung der Anwendung.

Wie man in Abbildung 12 erkennen kann, ist es mit Hilfe der Software zunächst möglich, eine Telefonnummer zu wählen. Und dies sowohl mit DTMF- als auch mit Pulswahl. Weiterhin erkennt die Applikation, wenn das Board „angerufen" wird - dies ist die sogenannte Ring-Detection. Im Treiber sind zwei Puffer implementiert, deren Zustand man an den Level-Anzeigen erkennen kann. Ein Puffer ist für die Sendedaten und einer für die Empfangsdaten vorgesehen.

Auch ein „Underrun" von Daten (wenn also der Empfangspuffer leerläuft) sowie ein „Overrun" (wenn die Applikation keine Daten vom Treiber-Puffer abholt und dieser so überläuft) werden angezeigt.

Mit dem Button „Hook" „hebt" das Modem ab bzw. „legt auf".

Über „Play" und „Record" kann man Daten auf der Festplatte speichern und bei Bedarf wiedergeben. Mit dieser Applikation ist also schon ein reichlich komfortabler Telefonanrufbeantworter realisierbar.

Um eigene Anwendungen zu erstellen, ist die Applikation in Quellcode auf der CD enthalten, die den Einsatz des Treibers verdeutlicht und als Beispiel für die Nutzung des PCI-Reference-Board-Pakets dienen kann, das wir ja bereits in Teil 2 dieser Artikelserie ausführlich beschrieben haben.

Der Treiber bietet jedoch nicht nur die Möglichkeit, das analoge Frontend anzusteuern, sondern mit ihm ist auch das serielle und parallele Interface der Karte zu betreiben, über das andere Hardwarekomponenten ansprechbar sind.

Somit kann die Schaltung der PCI-Karte auch als Basis für eigene Entwicklungen genutzt werden, die eine schnelle Umsetzung eines Projektes ermöglicht.

Wer über genügend Ehrgeiz, Zeit und Geld verfügt, kann anhand dieser Hard- und Software ein vollständiges funktionsfähiges Software-Modem entwickeln, wobei dafür ca. 1 Jahr einzuplanen ist und Lizenzgebühren für die verschiedenen Protokolle zu entrichten sind.

All diese Funktionalität sowohl von Anwendungsprogramm als auch der Hardware wäre nicht möglich ohne einen gut durchdacht programmierten Treiber. An dieser Stelle kann man die hohe Bedeutung eines solchen Programms sicher sehr gut erkennen.

Im nächsten Teil dieser Serie gehen wir näher auf das bereits angesprochene PCI-Board ein.

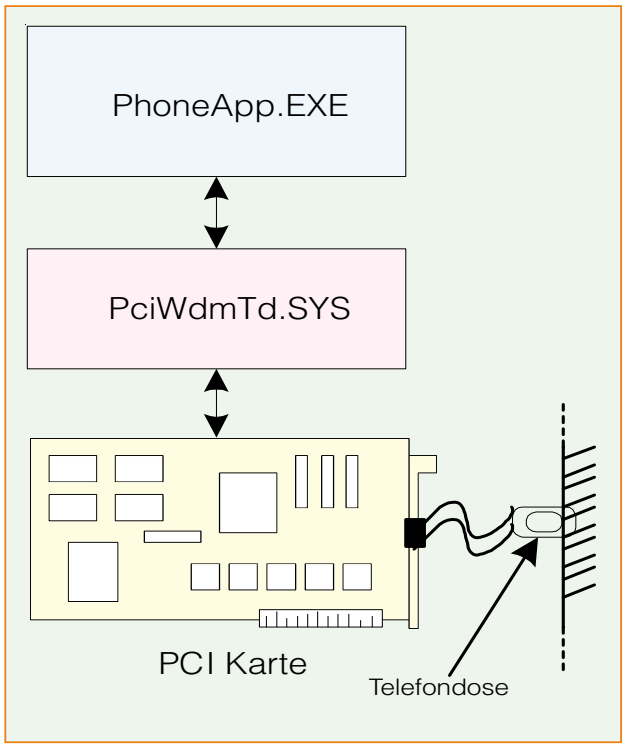

Bild 13: Die gesamte Umgebung (Hard- und Software) der Anwendung