Anhang 2: Der PCI-Bus

Inhaltsverzeichnis

Von ISA bis PCI

Historisches

Im Jahre 1985 stellte die Firma IBM den ersten AT Computer der Weltöffentlichkeit vor. Das damals verwendete Bus-System war der ISA-Bus (Industry Standard Architekture). Kurze Zeit nach der Markteinführung stellte sich jedoch sehr schnell heraus, dass das hier verwendete Bus-System nicht mehr dem aktuellen Stand der Technik entsprach. Man musste mit erschrecken feststellen, dass 286-Prozessoren bereits ausgebremst wurden (Beispiel: Graphikkarte oder Netzwerktechnologie mit hohen Datenraten). Schuld an diesem enormen Performance-Verlust war zum einen die Abwärtskompatibilität zu alten 8-bit-Systemen zum anderen der doch recht langsame Bustakt von 6 und später 8 MHz. Daraus ergab sich bei einer Busbreite von 16 Bit eine theoretische Datentransfer-Rate von 8 MB/s. In der Praxis wurden allerdings nur maximale Datenraten von 4 bis 6 MB/s erreicht.

Der MCA-Bus
Durch diese Umstände entschied sich damanls die Firma IBM zur Entwicklung eines neuen 32-bit-tauglichen Bussystems. 1987 wurde der MCA (Micro Channel Arichtecture) auf dem Markt eingeführt. Bei diesem Bus handelte es sich um einen 32-bit-Daten und Adressbus, der multimaster-fähig war und mit Datenübertragungsraten von bis zu 16 MByte/s aufwartete. Doch leider wurden diese neuen Rechner nicht mehr mit den alten ISA-Bus Slots ausgestattet, so dass sich dieser Bus letztendlich auf dem Markt nicht durchsetzen lies.

Der EISA-Bus
Bei der Entwicklung des EISA-Bus (Extended Industry Standard Architekture) spielten viele Erfahrungswerte des gescheiterten MCA-Bus eine große Rolle. Dieses neue Bus-System arbeitet mit 8 MHz auf 32-bit-Technologie. Aus der Busbreite von 32 bit ergab sich eine Datentransferrate von 16 MByte/s im Standardmode und von 33 Mbyte/s im Burstmode. Später folgten zwei weitere Burstmode-Versionen, der EMB-66 und der EMB-133, die eine Datenrate von bis zu 133 Mbyte/s. erreichten. Diese Leistung hatte jedoch auch ihren Preis und wurde deshalb nur in aufwendigen Serversystemen eingesetzt.

Der VLB-Bus
Der nun sensibilisierte Markt und die unterschiedlichen Hersteller von Peripheriegeräten suchten nach einem billigen und dennoch leistungsfähigem Bus-System. Hieraus entstand auf Umwegen der VLB-Bus (VESA-Local-Bus). Dieser hatte ebenfalls eine Busbreite von 32 bit und eine Taktfrequenz von 25…60 MHz. Dieser Bustyp erschien in drei Versionen, der Version VLB1.0 (Taktrate 25…40 MHz), VLB2.0 (Taktrate 25…50 MHz) und der VLB-64-bit (Taktrate 25…60 MHz). Das Hauptproblem dieser Bus-Technologie lag in der Nichteinhaltung der unterschiedlichen Spezifikationen. Die Spezifikation des VLB-1.0 erlaubte die Verwendung von Slots nur bis zu einem maximalem CPU-Takt von 40 MHz. Einige Hersteller bauten jedoch Systeme mit bis zu 3 Slots bei 50 MHz Taktfrequenz. Dadurch kam es zu erheblichen Problemen mit der Stabilität der einzelnen Systeme.

Der PCI-Bus
Im Jahr 1991 wurde der PCI-Bus (Peripheral Component Interconnect-Bus) von der Firma Intel konstruiert. Die Hauptgründe für eine neue Busentwicklung waren:

- Eine höhere Datenrate als bei dem 16-bit ISA-Bus-System
- Eine größere elektromagnetische Verträglichkeit (EMV) zu allen Vorgängersysteme
- Zukunftssicherheit für folgende Prozessorgenerationen

1992 war es dann soweit, der erste PCI-Bus-System wurde der Weltöffentlichkeit vorgestellt. Im späteren Verlauf der weiteren Entwicklung dieses Bus-Systems ergaben sich unterschiedliche Spezifikationen (Version 1.0, 2.0, 2.1 und aktuell 2.2). Aus den 1991 gesetzten hohen Vorgaben entstand ein 32-bit breites Bus-System, das Daten und Adressen im Zeitmultiplex-Verfahren überträgt und sogar Burst-Zyklen von unterschiedlichen Längen zulässt. Die Firma INTEL konnte eine vollständige, realistische und gut strukturierte Definition aller wichtigen Bus-Signale als Grundlage für den PCI-Bus durchsetzen. In der Entwicklung dieses aufbauenden, nach oben hin offenen Bus-Systems sollten nicht die gleichen Fehler wie bei den Vorgängersystemen gemacht werden.

Eigenschaften PCI-Bus

Das PCI-Bussystem besteht aus drei unterschiedlichen Modulen:

- Der Data-Path-Unit
- Das Expansion-Bus-Interface
- Der Host-Bridge mit einem Cache-DRAM-Controller

Mit Hilfe der Data-Path-Unit wird eine 32-bit Verbindung zu den einzelnen im System befindlichen Komponenten hergestellt. Über das Expansion-Bus-Interface können andere Bus-Systeme angeschlossen werden (Bsp. PCI oder AGP). Erst durch das Expansion-Bus-Interface und weiteren System-Bridges ist beispielsweise eine Abwärtskompatibilität zum ISA-Bus-System möglich oder auch zu neuen Technologien wie AMR.

Die Host-Bridge stellt den zentralen Baustein des PCI-Bussystems dar. Mit ihrer Hilfe kann eine Verbindung zwischen PCI-Bus und CPU hergestellt werden. Desweiteren setzt sie PCI-Zyklen in CPU-Zyklen um und umgekehrt. Dadurch wird der PCI-Bus im Gegensatz zu anderen Bussystemen prozessorunabhängig, da für die Anbindung des PCI-Bus an den jeweiligen Prozessortyp (INTEL, Alpha, AMD usw.) nur unterschiedliche Host-Bridges verwendet werden müssen. Diese Eigenschaft macht das PCI-Bussystem auch für zukünftige Prozessorgenerationen relativ unabhängig und erweiterbar.

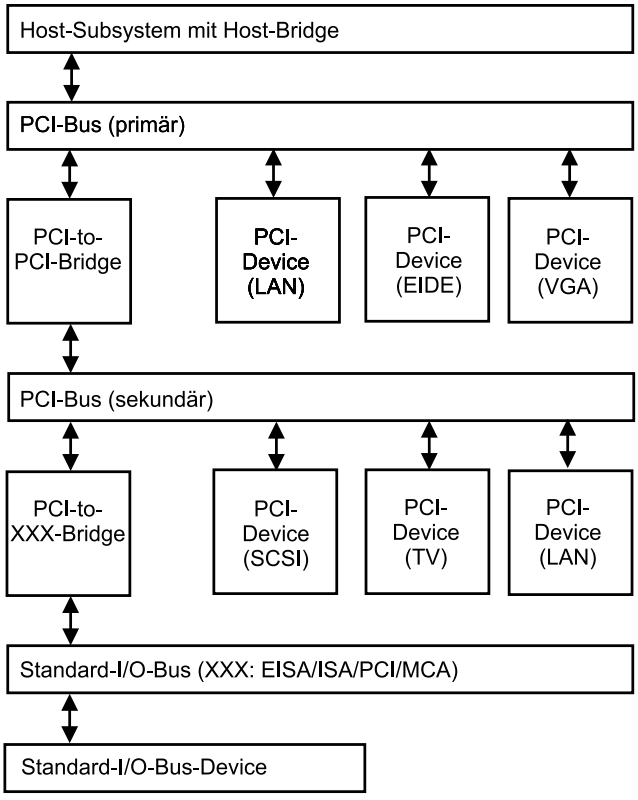


Bild 1: Hierarchisches PCI-Bus-System

PCI-Spezifikationen

Die PCI-Spezifikation erlaubt es insgesamt 10 Devices an einem PCI-Bus zu betreiben. Da die Host-Bridge aus Sicht des PCI-Bus als ein PCI-Device erkannt wird, stehen allerdings real nur 9 Devices zur Verfügung. Diese können zum Beispiel für On-Board-Komponenten (SCSI, EIDE, LAN, usw.) beziehungsweise für PCI-Slots zum Anbinden von PCI-Karten verwendet werden. Dabei muss jedoch beachtet werden, dass ein Slot zwei Devices benötigt und daher nur vier Slots über einen PCI-Bus angesteuert werden können. Die Anzahl der

Slots kann jedoch durch Anbinden eines zweiten PCI-Bussystems an das Expansion Bus Interface weiter erhöht werden. Insgesamt können so bis zu 256 Busse zusammengeschaltet werden, wobei die ersten 255 PCI-Busse sind und der letzte Bus ein ISA-, EISA-, VL- oder sogar ein MCA-Bus sein kann. Auf diese Weise können selbst große Systeme realisiert werden (siehe Bild 1).

Eine andere Möglichkeit der Sloterweiterung findet man auf einigen aktuellen Mainboards (zum Beispiel Abit KT-7, Gigabyte ZX, Epox EP-8KTA+). Hier werden mit Hilfe des IRQ-Sharings bis zu sechs Slots auf einem Mainboard angesteuert. Dabei teilen sich zum Beispiel der 5. und 6. Slot einen IRQ. Dies geht jedoch, bei Verwendung von mehr als vier PCI-Karten, zu Lasten der Performance. Es kann zudem nicht garantiert werden, dass grundsätzlich alle PCI-Kartenkombinationen einwandfrei funktionieren.

Über die Slots können unterschiedlichste PCI-Karten angeschlossen werden. Folgende Karten werden unterschieden:

- **Single-Function PCI-Karten**: Single-Function PCI-Karten können zum Beispiel Grafikkarten oder SCSI-Host-Karten sein.
- **Multi-Function PCI-Karten**: Ein PCI-Device kann bis zu acht verschiede I/O-Funktionen haben, woraus sich die Multi-Function PCI-Karten ergeben.
- **Multi-Device PCI-Karten**: Die Multi-Device PCI-Karten können aus mehreren der oben genannten PCI-Karten bestehen. Dabei werden auf der Multi-Device PCI-Karte die unterschiedlichen Devices über eine PCI-to-PCI-Bridge angebunden. Somit stellt die Multi-Device PCI-Karte einen komplettes PCI-Bussystem dar.

Die Plug-and-Play (PnP) Funktion des PCI-Bus

Das BIOS findet nach dem Einschalten des Computers alle PCI-Devices und fragt jedes Device nach den benötigten Ressourcen. Die benötigten Ressourcen bestehen aus I/O Adressen, IRQ-Nummern, DMA-Kanälen und verwendeten Speicherbereichen. Überschneiden sich bei unterschiedlichen Karten die jeweils benötigten Ressourcen, dann versucht das BIOS durch umkonfigurieren einer PCI-Karte die Funktion beider in einem System zu ermöglichen. Ist dies einmal nicht der Fall, schaltet das BIOS eine der beiden Karten einfach ab. Dies kommt aber relativ sehr selten vor. Die verwendeten Ressourcen jeder PCI-Karte werden in der ESCD-Datenbank des BIOS dem verwendetem Betriebssystem zur Verfügung gestellt. Desweiteren werden diese Informationen auch in einem bis zu 256 Bytes großem Speicherbereich auf der PCI-Karte abgelegt, dem Configuration Space (siehe Bild 2). Aus diesem Speicherbereich hat sich auch das BIOS die Informationen über die benötigten Ressourcen geholt und stellt es anderen Anwendungen über spezielle BIOS-Interrupt-Aufrufe zur Verfügung. Da diese BIOS-spezifischen Parametrierungsregister in jedem System dynamisch geführt werden und kein einheitlicher, fester Speicherplatz (beispielsweise MEM-Adresse) vorgegeben wird, kann das Lesen dieser Information auch nur durch einen für alle Motherboard-Hersteller vorgegebenen Software-Interrupt erfolgen.

Über **Status** und **Command** können Informationen über die verwendete Karte ausgelesen und gegebenenfalls auch gesetzt werden. Die Vendor-ID ist die Nummer des Herstellers der verwendeten Karte. Diese Nummer wird vom PCI-SIG (PCI Special Interest Group) vergeben. Die PCI-SIG ist ein Konsortium, welches alle Spezifikationen des PCI-Busses kontrolliert. Über diese Institution können auch die genauen Spezifikationen zum PCI-Bus bezogen werden, in denen auch detaillierte Erläuterungen zu den Register stehen.

31	16	15	0	
Device-ID		Vendor-ID		00h
Status		Command		04h
Class Code			Revision-ID	08h
BIST	Header Typ	Latency Timer	Cache Line Size	0Ch
Base Address 0				10h
Base Address 1				14h
Base Address 2				18h
Base Address 3				1Ch
Base Address 4				20h
Base Address 5				24h
Cardbus CIS-Pointer				28h
Subsystem-ID		Subsystem Vendor-ID		2Ch
Expansion ROM Base Address				30h
Reserved				34h
Reserved				28h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

Bild 2: Configuration Space einer PCI-Karte.

Die PCI-Devices unterteilen sich in zwei Gruppen, Initiator (Master) und Target (Slave). Der Initiator startet eine Datenübertragung, indem er die Kontrolle über die Steuersignale übernimmt und die genauen Informationen zur Datenübertragung festlegt (Adressen, Beginn, usw.). Das Target erzeugt ein Handshake-Signal für den Initiator, wenn es seine Adresse auf dem PCI-Bus erkannt hat (Initialisierungsphase). Desweiteren kann es die Verfügbarkeit von Daten, beziehungsweis die Bereitschaft zum Empfangen von Daten oder die Bitte um Wartezyklen signalisieren. Somit können über diese beiden Gruppen sämtliche Funktionen eines PCI-Bussytems genutzt werden.

Es besteht jedoch die Möglichkeit, dass alle PCI-Devices einen Master für den PCI-Bus darstellen. Ist dies der Fall muss eine Arbitierungslogik entscheiden welcher Master den PCI-Bus nutzten darf (Arbitierungsphase). Hierbei müssen die beiden PCI-Devices festlegen in welchen Modus die Daten übertragen werden sollen, im Burst- oder im Non-Burst-Mode. Im Burst-Mode (siehe Bild 3) werden normalerweise Daten ab vier DWORDs (32 bit) übertragen. Dabei wird vor den Daten die Adresse gesendete und danach die dazugehörigen Daten.

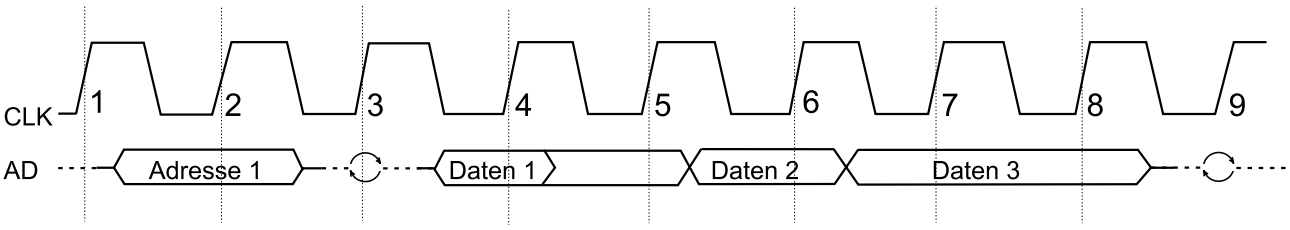


Bild 3: Der Burst-Mode.

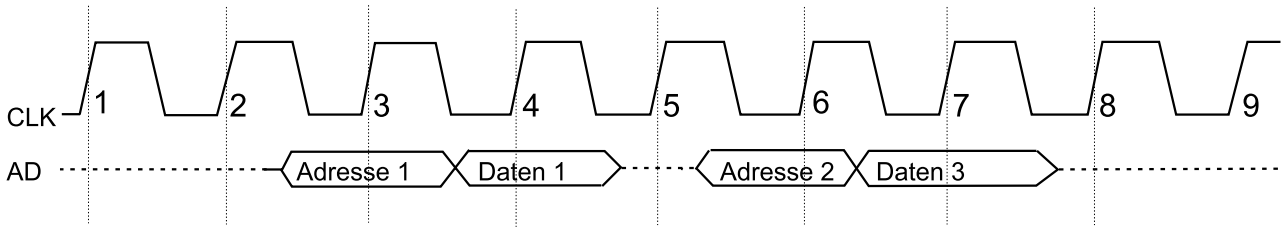


Bild 4: Impulsdiagramm zum Non-Burst-Mode.

Im Non-Burst-Mode (siehe Bild 4) wird vor jedem Daten-DWORD die jeweilige Adresse gesendet. Hierdurch ergeben sich unterschiedlich Datenraten für die jeweiligen Modi. Im Non-Burst-Modus ergibt sich eine Datenrate beim Lesen von 44 MByte/s (drei Takte / DWORD) und beim Schreiben 66 Mbyte/s (zwei Takte / DWORD). Da im Burst-Mode für jedes weitere DWORD ein Takt benötigt wird, kann mit zunehmender Burstlänge (Anzahl der DWORDs) eine Übertragungsrate von annähernd 117 Mbyte/s erreicht werden. Laut der PCI-Bus Spezifikation 2.1 (PCI-Bus 32 bit bei 33 MHz) können Datentransferraten bis zu 117 Mbytes/s erreicht werden. In der PCI-Bus Spezifikation 2.1 (PCI-Bus 32 bit bei 66 MHz) kann es zu einer Datentransferrate von bis zu 234 Mbytes/s kommen und in der Spezifikation 2.1 (PCI-Bus 64 bit bei 66MHz) können die Datentransferraten auf einen maximalen Wert von bis zu 468 Mbytes/s ansteigen (siehe Bild 5).

	PCI-Bus 32 bit 1.0	**PCI-Bus 64 bit 2.0**	**PCI-Bus 32 bit 2.1**	**PCI-Bus 64 bit 2.1**
Busart	Synchronbus	Synchronbus	Synchronbus	Synchronbus
Taktfrequenz in MHz	33	33	66	66
mit Slots	33	33	66	66
Datenbusbreite in Bit	32	64	32	64
Adressbusbreite in Bit	32	32	32	32
Anzahl Devices	10	10	10	10
Anzahl Slots	4	4	4	4
max. Burst-Länge	nicht begr.	nicht begr.	nicht begr.	nicht begr.
Datenraten bei 33 MHz, ohne Extra-Wait in MByte/s				
Non-Burst-Read	44	88	44	88
Non-Burst-Write	66	132	66	132
Burst-Read	106	211	106	211
Burst-Write	117	234	117	234
Datenraten bei 66 MHz, ohne Extra-Wait in MByte/s				
Non-Burst-Read			88	172
Non-Burst-Write			132	264
Burst-Read			211	423
Burst-Write			234	468
Autokonfiguration			ja	ja
Concurrency			ja	ja
Interrupt-Sharing			ja	ja

Bild 5: Die PCI-Spezifikationen und ihre unterschiedlichen Datentransferraten.

Das PCI-Bus Timing Diagram

Der PCI-Bus ist ein synchrones Bussystem, das alle Datenübertragungen relativ zu einen Systemtakt (CLK) sendet, beziehungsweise empfängt. Ein Beispiel hierfür ist der Read-Befehl. An ihm soll nur verdeutlicht werden, wie die einzelnen Signale untereinander im Zusammenhang stehen (siehe auch Bild 6).

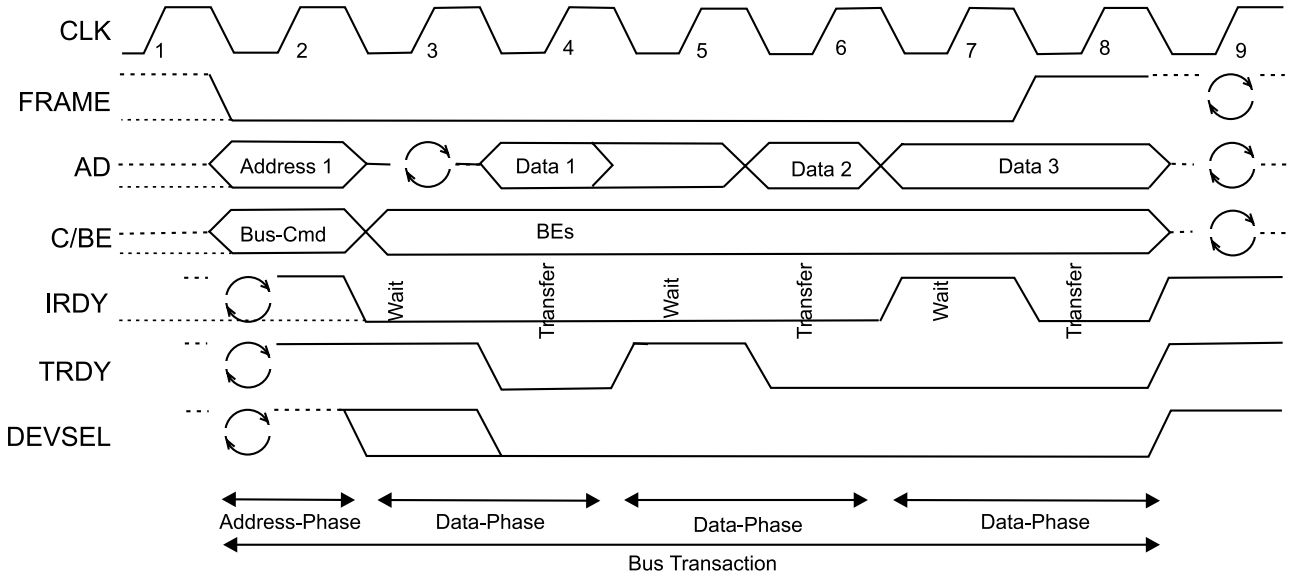


Bild 6: PCI-Timing-Diagramm eines Read-Befehls.

Takt 1: Der PCI-Bus ist im Ruhezustand.

Takt 2: FRAME# wird vom Master aktiviert um den Beginn einer Transaktion anzuzeigen
AD enthält eine Adresse
C/BE# enthält ein Bus-Command
IRDY#, TRDY# und DEVSEL# sind im Turn-Around-Zyklus, da ein Treiber-Wechsel erfolgt

Takt 3: AD führt einen Turn-Around durch, da die Kontrolle von Master auf Target wechselt
C/BE# wird mit Byte-Enables getrieben
IRDY# ist aktiv, da der Master die Daten lesen kann
TRDY# ist noch daktiviert, da noch kein Target gefunden ist

Takt 4: AD enthält die ersten Daten
TRDY# ist aktiviert, da die ersten lesbaren Daten auf dem Bus sind
DEVSEL# ist aktiviert, da das Target sich an der Adresse von Takt 2 erkannt hat

Takt 5: AD enthät noch immer die Daten aus Datenphase 1
TRDY# ist deaktiv, da das Target (Datenquelle) eine Pause einlegen will

Takt 6: AD enthält die Daten der Datenphase 2
TRDY# signalisiert, dass die Daten gelesen werden können

Takt 7: AD trägt die Daten für Datenphase 3
IRDY# wurde vom Master deaktiviert, da er einen Waitstate verlangt

Takt 8: FRAME# wird vom Master deaktiviert, da der letzte Datenblock nun übertragen wird

Takt 9: FRAME#, AD, C/BE# legen Turn-Arounds ein, da der Treiber wechseln kann
IRDY#, TRDY# und DEVSEL# sind deaktiviert, da keine Transaktion erfolgt
Der Bus ist aufgrund von FRAME# = 1 und IRDY# = 1 im Idle Zustand

Zur Verdeutlichung der einzelnen Signalleitung ein Überblick über die verwendeten Signalleitungen eines PCI-Device (siehe Bild 7).

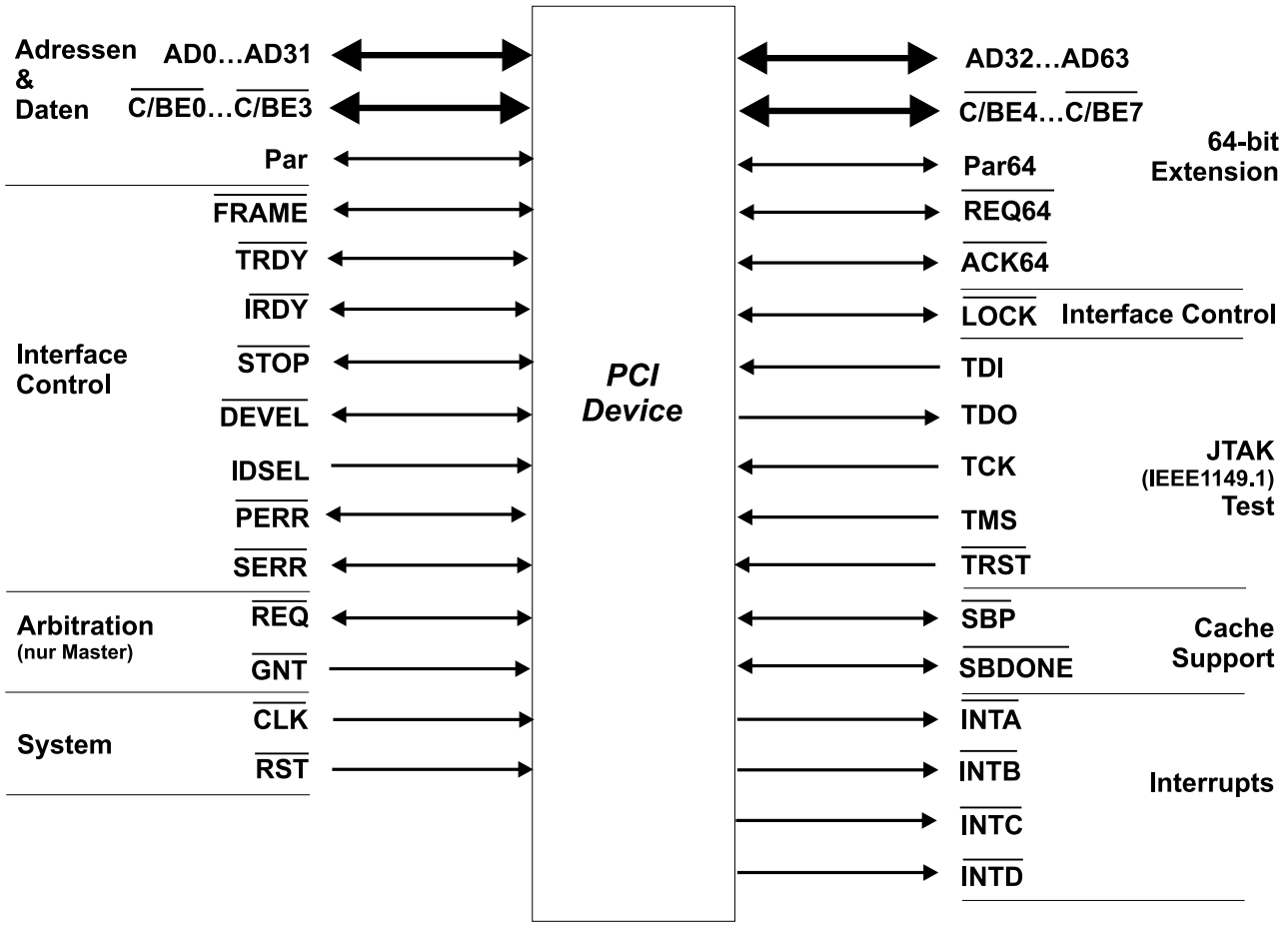


Bild 7: Die Signalleitungen eines PCI-Device.

PCI-Geschwindigkeit

Durch den PCI-Bus, der von der Prozessorgeschwindigkeit unabhängig ist, ergeben sich neue Möglichkeiten für Multimedia-Applikationen, Netzwerke, Messtechnik und vieles mehr. Dies liegt hauptsächlich an den enormen Datentransferraten (besonders des PCI-Bus 64 bit 2.1), so dass Daten von PCI-Karten zur CPU und umgekehrt schneller verarbeitet werden können. Mit einer „alten“ ISA Soundkarte in kombination zu zum Beispiel einer PCI-3D Grafikbeschleunigerkarte können Sie unter Umständen Ihr System um bis zu 10…20 % ausbremsen, bei voller Ausnutzung der zu Verfügung stehenden Performance der PCI-Karte. Wichtig für die Stabile Performance des Systems sind auch die einzelnen Einstellung im BIOS des Rechners, womit die Stabilität des Systems ganz erheblich beeinflußbar ist. Bedingt durch den breiten Datenbus und der höheren Taktung moderner Motherboards, arbeitet der PCI-Bus mit einer größeren Geschwindigkeit als beispielsweise der Standard-ISA-Bus. Für fehlerfreien Datenverkehr ist daher oftmals eine Verzögerung des PCI-Busses notwendig, da das Motherborad keine Informationen über die Grenzwerte der eingestecken Hardware hat. Die folgende Option stellt die Dauer der Verzögerung ein, die der PCI-Bus bei einer Transaktion zwischen dem angegebenen PCI-Steckplatz und der CPU wartet. Der Wert ist unter anderem von der benutzen PCI-Master-Einheit abhängig. Eine der wichtigsten Einstellungen ist somit die PCI-

Latency-Timer Zeit, die mindestens 32 CPU-Clock-Zyklen dauern sollte. Wird dieser Wert zu klein gewählt, kann die CPU eventuell schneller auf die Hardware zugreifen, als die Daten im Decoder verarbeitet werden können. Das macht zwar den Rechner und die Karte schön schnell (besonders beim Wert 0) oft jedoch bleibt das nicht ohne unangenehme Folgen, die sich eventuell auch noch erst nach längerer Betriebszeit zeigen (beispielsweise Überhitzung mit anschließendem Defekt usw.).

Fehler wie Bootprobleme, falsche I/O-Adresse, falsche Device-Angaben oder Zugriffprobleme auf die Hardware sind nur ein paar der Fehler, die dann auftreten können, wenn die BIOS-Werte falsch oder für die Slot-Hardware zu schnell eingestellt sind.

Zweitwichtigster BIOS-Eintag ist die Aktivierung der PCI-Delay-Transaction, welche auf Enable stehen sollte. Damit wird lediglich festgelegt, dass die verwendete Hardware mindestens der PCI-Spezifikation 2.1 entspricht. Interne Busprotokolle und spezielle Datenpakete werden entsprechend der Spezifikation übertragen und erhöhen die Daten-Sicherheit zwischen Rechnereinheit und PCI-Slot-Hardware. Natürlich spielt sich bei diesen Transactions eine ganze Menge mehr auf dem PCI-Bus ab. Das zu erläutern würde aber an dieser Stelle zu weit führen.

Weitere BIOS-Einstellungen wie PCI-Master 0 WS write (= Disable), PCI-Dynamic Bursting (= Disable) und CPU-to-PCI buffer (= Enable) sind ebenfalls wichtig, um eine saubere Ansteuerung der Hardware zu gewährleisten. Gerade die Einstellung „CPU to PCI buffer“ gilt es nicht zu unterschätzen. Ist der Buffer auf Enable gesetzt, wird der Datenfluss ohne Unterbrechung der CPU durch einen „intelligenten“ Puffer geleitet, der zusätzlich die Koordination von Read/Write-Zyklen zum PCI-Device ordnet, ohne das es zu Datenkonflikten kommen soll. Über dieser Einstellung wird beispielsweise in einem Prozess mit vier DWORD-Datenwörtern zur PCI-Karte geschrieben. Umso wichtiger ist die bereits oben beschriebene PCI-Latency-Zeit, die quasi den Abstand der Datenblöcke vorgibt. Bei der Einstellung OFF beziehungsweise Disable wird der Datenpuffer nicht benötigt und der Prozessor-Zugriff erst dann abgeschlossen, wenn der PCI-Bus dem Prozessor ein Signal mit der Meldung sendet, dass der Bus zur Datenaufnahme bereit steht.

PCI-Einstellungen im BIOS

Alle oben genannten Einstellungen für das BIOS lassen sich über das BIOS-Menü einstellen. Dieses Menü wird im Allgemeinen dadurch aufgerufen, indem Sie beim Booten des Rechners die „Entf“-Taste (oder „Del“-Taste) gedrückt halten. Bei einigen Systemen kann auch eine andere Taste zu den Einstellungsseiten des BIOS führen. Leider findet man selten eine ausführliche Dokumentation der BIOS Einstellungen im Lieferumfang eines Motherbords gescheige denn bei den Unterlagen, die man beim Kauf eines Komplett-Systems erhält, so dass man sich die nötigen Informationen aus anderen Quellen beschaffen muss.

Die für PnP wichtigen Einstellungen findet man im zum Beipsiel im Award-BIOS-Menü unter dem Eintrag: PNP AND PCI SETUP. Das PnP and PCI Setup ist für die Vergabe von Systemressourcen wie IRQs und DMAs zuständig. Bei den anderen BIOS-Herstellern (zum Biespiel AMI, Phoenix) kann dieses Einstell-Menü allerdings eine leicht abweichende Bezeichnung haben.

Funktioniert das System einwandfrei, sollten hier keine Änderungen vorgenommen werden. Im Falle eines Interruptkonflikts liegt jedoch in diesem Menü oft des Rätsels Lösung.

- **Ressources Controlled By / PnP OS Installed / Plug and Play Aware O/S:**
  Hier läßt sich einstellen, ob die PnP-Daten per Setup oder automatisch zugewiesen werden. Sind keine Probleme vorhanden, sollten man die Option Auto wählen.

- **Reset Configuration Data oder Clear NVRAM on every Boot:**
  Diese Option hilft, wenn neue Komponenten einen Systemstart verhindern. Wählt der Anwender »enabled«, werden die Informationen des ESCD neu geschrieben. Beim nächsten Start schaltet sich die Option ab.
- **IRQ-X/DMA-X assigned oder DMA Channel X:**
  Hier läßt sich festlegen, ob der DMA/IRQ vom PnP-Bios zugewiesen werden soll oder für ISA reserviert wird.
- **PCI IDE IRQ Map to:**
  Regelt die Nutzung von IRQ14 und 15. Bei Einsatz der Onboard- IDE-Kanäle muß hier Auto stehen, beim Einsatz eines ISA-IDE-Controllers ISA.
- **PCI IDE Prefetch Buffers:**
  Vermeidet bei Deaktivierung Schreib/Lese-Fehler auf der Festplatte, verringert jedoch die Performance.
- **PCI IDE BusMaster:**
  Aktiviert den IDE- Busmasterbetrieb und damit die DMA-Funktion einer Festplatte. Die Option sollte »enabled« sein.
- **PCI Latency Timer:**
  Gibt an, wie lange ein Gerät den PCI-Bus belegen darf. Ist der Wert zu hoch, kann es zu Problemen mit Sound- oder Netzwerkkarten kommen, die sich einen Interrupt teilen. Empfehlenswert sind 40 bis 50 Takte. Award steht defaultmäßig auf 32, AMI auf 66 und Phoenix gibt 40 Takte vor.
- **PCI Bus Parking:**
  Ermöglicht für eine PCI- Komponente besondere Reservierungszeiten. Diese Option kann bei Problemen mit ISDN- Karten sehr hilfreich sein.
- **Assign IRQ to PCI VGA Card:**
  Hier wird der VGA-Karte ein IRQ zugewiesen. Diese Option muß beim Einsatz von 3D-Beschleunigern eingeschaltet werden.
- **PCI SlotX IRQ Priority:**
  Verändert die Priorität des PCI-Slots. Normalerweise hat Slot 1 die höchste Priorität.

Probleme mit PCI-Karten

Entspricht eine PCI-Karte genau dem definierten PCI-Spezifikationen, wird es im Normalfall bei der Installation keine Problem geben. Dank Bios-Identifikation und automatischer Hardware-Erkennung in Windows 95/98 ist die Konfiguration nach kurzer Zeit erledigt. In der Praxis allerdings kommt es manchmal zu unerwarteten Phänomenen: Da werden neue Karten schon vom BIOS nicht erkannt oder falsch angesprochen, oder es treten nach der Hardware-Erkennung Konflikte auf, die es laut Plug and Play eigentlich nicht geben dürfte. Die Folge ist schlimmstenfalls ein zerstörtes Windows, das dann - wenn überhaupt - nur noch im abgesicherten Modus startet. In diesem Fall bleibt dem Anwender oft keine andere Möglichkeit, als über die Versuch-und-Irrtum-Methode alle installierten PCI-Komponenten zu testen, bis das Problem lokalisiert ist. Besonders ältere PCI-Karten oder ISA-Adapter, die auf die Schnelle auf PCI portiert wurden, verfügen oft über Jumper oder andere Einstellmöglichkeiten. Das sollte gemäß den PCI-Vorgaben eigentlich verhindert werden. Im Prinzip funktionieren derartige PCI-Karten wie ältere ISA-Karten, bei denen es dem Anwender überlassen bleibt, die Ressourcen (DMAs, IRQs, I/O- und Memory-Adressen) richtig zu vergeben. Daher kann weder das Plug-and-Play-Bios noch ein Plug-and-Play-fähiges Betriebssystem wie Windows 98 erkennen, ob irgendwo Konflikte entstehen.

Wer eine solche Karte einbauen muss, sollte zunächst über den Gerätemanager die aktuelle Konfiguration des Systems auf freie IRQs und Speicheradressen prüfen. Erst dann kann die Karte auf eine konfliktfreie Adresse eingestellt werden. Hilfreich ist in diesem Fall auch, im PCI-Teil des Bios den gejumperten Interrupt dem entsprechenden PCI-Slot explizit zuzuweisen. Allerdings wird diese Option gerade bei neuen Bios-Versionen kaum noch unterstützt. Dann bleibt nur das manuelle Ermitteln einer geeigneten Adresse.

PCI-Kopmonenten erkennen
Ein weiteres Problem sind zwar erkannte, aber nicht spezifizierte PCI-Devices. Normalerweise zeigt das Bios beim Systemstart alle PCI-Komponenten namentlich an. Es kann aber auch vorkommen, dass ein Eintrag lediglich die Bezeichnung „Unknown PCI Device“ enthält. In diesem Fall hat der Kartenhersteller auf die Vergabe einer spezifizierten ID verzichtet. PCI definiert einen Konfigurationsbereich (Configuration Space) von 256 Byte, der alle Angaben zur automatischen Erkennung enthält. Die Angaben werden während des Bootvorgangs aus einem EEPROM auf der PCI-Karte gelesen. Dabei unterteilt sich der Konfigurationsadressraum in einen Header und einen geräteabhängigen Bereich. In den PCI-Spezifikationen sind lediglich Angaben für den 64 Byte umfassenden Header festgelegt. Die verbleibenden 192 Byte enthalten geräteabhängige Informationen. Hier finden sich bei einem PCI-Motherboard beispielsweise Register für die Cache- und DRAM-Speichersteuerung sowie für die Bridges (ISA, Eisa). Bei der Fehlersuche ist jedoch nur der Header von Bedeutung. Das Auslesen der hier gespeicherten Daten erfolgt über drei unterschiedliche Verfahren zur Geräteerkennung:

- mit Hilfe des Bios-Interrupts 1Ah, Funktionsnummer B1h,
- über zwei 32-Bit-Adressen im I/O-Adreßraum (0CF8h: address, 0CFAh: data),
- durch Einblendung eines Configuration Space in einen 4 KByte großen Adreßbereich zwischen C000h-CFFFh.

Erkennungsverfahren
Üblich ist das erstgenannte Verfahren, da es auch hardware-nahen Applikationen, sprich Treibern, die direkt mit PCI-Devices kommunizieren, zur Verfügung steht. Nach der Erkennung werden die Geräte als PCI Device Listing auf dem Bildschirm angezeigt. Wichtig zur Konfiguration ist hier die Vendor ID (Herstellercode) und die Device ID, die beide auch in der Geräteliste erscheinen. Jedes Erkennungsverfahren beruht darauf, alle bekannten Vendor- und Device-IDs vom Bios abzufragen. Die Vendor-ID wird wie die Device-ID durch die PCI-Special-Interest-Group (PCISIG) vergeben, wobei eine Vendor-ID nur auf Herstellerantrag ausgestellt wird. Besonders Hersteller von PCI-Karten mit kleinen Stückzahlen verzichten oft auf diesen Antrag, da dies mit einigen Kosten verbunden ist. Die Bios-Hersteller übernehmen die Listen und legen sie fest im System ab. Aus diesem Grund ist es nicht ungewöhnlich, daß ein relativ neues PCI-Device von einem älteren PCI-Bios nicht erkannt wird.

Plug and Play
Allerdings ist das kein Grund, daß das Device nicht unter Windows 95/98 verwendet werden kann. Denn hier kommt die interne Plug-and-Play-Funktionalität zum Tragen, die neben den internen Karteninformationen auch Treiberdaten ausliest. Daher kann Windows 95/98 auch dann mit Plug-and-Play-Komponenten umgehen, wenn es sich um ein älteres Bios handelt oder sich die Karte nicht nach den Vorgaben richtet. Besonders ISA-Karten mit Plug-and-Play-Unterstützung profitieren davon. Weniger schön ist diese Vorgehensweise der Hersteller für NT- und Linux-Nutzer: Wird die Karte nicht vom Bios erkannt, scheitert hier die Installation. Dann helfen nur entsprechende Test-Utilities weiter.

Wesentlich seltener kommt es vor, dass zwar das BIOS die PCI-Devices korrekt erkennt, die Hardware-Erkennung unter Windows 95/98 jedoch nicht fündig wird. Dies lässt auf eine generelle Fehlkonfiguration im Betriebssystem schließen. In diesem Fall sollte man das falsch erkannte Gerät im Gerätemanager löschen und bei einem Systemstart erneut einlesen. Funktioniert auch dies nicht, kommt man oft damit weiter, dass die Karte in einen anderen Slot eingebaut wird. Dies sollte auf jeden Fall dazu führen, dass das neues PCI-Device von Windows 95/98 erkannt wird.

BIOS-Probleme

Aber nicht nur PCI-Karten, sondern auch das BIOS selbst kann Probleme verursachen. Wird eine PCI-Karte beim Booten nicht angezeigt, obwohl sie den Spezifikationen entspricht, liegt das oft am BIOS. Besonders ältere Versionen können oft nur bis zu fünf PCI-Devices verwalten. Dabei geht es nicht nur um die Zusatzkarten, sondern um alle PCI-Einheiten, sprich auch die auf dem Mainboard wie beispielsweise die PCI-ISA-Bridge, die Host-Bridge oder auch das IDE-Interface. Es kann daher sein, dass nach dem Einbau einer dritten PCI-Komponente diese nicht erkannt wird. Ob ein solches BIOS-Problem vorliegt, lässt sich schnell kontrollieren: einfach einige PCI-Devices des Mainboards im BIOS abschalten und prüfen, ob die nicht erkannte Komponente jetzt erfasst wird. Ist dies der Fall, sollte man sich ein Bios-Update besorgen und installieren. Ansonsten kann unter Umständen die Hardware-Erkennung von Windows 95/98 das Gerät finden, wenn entsprechende Treiber beiliegen. Dazu sind die Funktion unter Systemsteuerung/Hardware manuell aufzurufen und dann die Treiber zu installieren. Danach sollte der Zugriff funktionieren.
Unter NT 4.0 und Linux hingegen ist ein Bios-Update die einzige Möglichkeit, die Karte ordnungsgemäß im System anzumelden.

Besondere Installationsroutinen

Einige Hersteller beschreiten bei der Installation ihrer PCI-Karten etwas undurchsichtige Wege über ein eigenes Installationsprogramm. Mit Hilfe des Setuptools wird das PCI-Device unter Windows 95/98 eingerichtet, obwohl das BIOS zunächst keine Daten übermittelt. Dabei geht der Erkennung meist die Meldung Updating ESCD (Extended System Configuration Data) voraus, was gleichbedeutend mit einem Update des PCI-BIOS ist. Strenggenommen widerspricht diese Vorgehensweise dem vorgeschriebenen PCI-Plug-and-Play-Vorgaben mit der Folge, dass andere Betriebssysteme wie etwa Windows NT 4.0 oder Linux die Karte nicht akzeptieren, da sie nur über die interne Konfigurationsfunktion von Windows 95/98 angesprochen wird. Wer also einen Umstieg auf NT oder Linux plant, sollte von diesen Karten generell Abstand nehmen beziehungsweise vorher prüfen, ob entsprechende Treiber verfügbar sind. Im übrigen kommt es durchaus vor, dass Einträge in den INI-Dateien von Windows 95/98 für PCI-Karten vorgenommen werden, was laut Microsoft seit Existenz der Registry nicht mehr praktiziert werden sollte. Hardware-relevante Eintragungen wie verwendete IRQs in der Datei system.ini sind ein deutliches Warnzeichen dafür, daß es mit der korrekten Plug-and-Play-Funktionalität der PCI-Karte nicht weit her ist.

Der letzte Ausweg

Wenn nach dem Einbau einer Karte gar nichts mehr funktioniert, ist es oft hilfreich, die gespeicherten Daten im BIOS zurückzusetzen. Plug-and-Play-Konfigurationsangaben liegen im ESCD-Bereich. Dieser befindet sich nicht im normalen CMOS-RAM, sondern in einem Flash-PROM. Der ESCD-Bereich enthält Informationen über die im System verwendeten Plug-and-Play-Devices, wobei auch die ISA-Plug-and-Play-Einheiten erfasst sind. Bei einem BIOS-Update mit Hilfe eines Flash-Writer-Programms (liegt auf den Internet-Seiten des jeweiligen Mainboard-Herstellers bereit), sind oft Menüpunkte wie *Clear PNP ESCD Parameter Block* und auch *Update BIOS Including Boot Block and ECSD* zu finden. Die letztgenannte Option sorgt für eine komplette Neuprogrammierung des BIOS-ROMs, sorgt also für das komplette BIOS-Update.

Im Falle einer nicht erkannten PCI-Karte ist die Option *Clear PNP ESCD Parameter Block* hilfreich: Sie löscht nur die gespeicherten Plug-and-Play-Parameter. Dadurch werden beim nächsten Systemstart alle Daten neu eingeschrieben. Diese Aktion ist immer dann anzuraten, wenn sich der PC aufgrund eines Problems mit einem PCI- oder auch ISA-Plug-and-Play-Device allen anderen Konfigurationsmaßnahmen (Bios-Einstellungen, Treiberkonfiguration) entzieht. Um das Problem besser lokalisieren zu können, bietet es sich an, vor dem Löschen der PCI-Parameter kritische PCI-Devices zu entfernen, um sie dann Schritt für Schritt wieder einzubauen. Auf diesem Weg lässt sich die problembehaftete Komponente schnell während des Systemstarts ermitteln. Dabei kann es sinnvoll sein, den ESCD-Parameterblock nach dem Hinzufügen der einzelnen Karten immer wieder zu löschen. Allerdings sollte nicht bei jedem Neustart Windows geladen werden, da nach jedem BIOS-Reset die Hardware-Erkennung startet.

Geräteerkennung über Class Codes

Fehlen die notwendigen IDs, lässt sich eine PCI-Karte auch über Class Codes konfigurieren. Die Vendor- und die Device-ID werden als Funktionswerte mit Hilfe des PCI-Interrupts über das CX- (Device-ID) und das DX-Register (Vendor-ID) übergeben. Kennt der Anwender diese IDs nicht, müsste er im Falle eines Fehlers alle möglichen Kombinationen durchprobieren und stets überprüfen, bei welcher Kombination sich ein PCI-Device meldet - ein bei der Fehlersuche nicht durchführbares Verfahren. Aber es gibt noch eine zweite Möglichkeit für die Erkennung von PCI-Devices: Die sogenannten Class Codes. Dabei wird über das ECX-Register ein bestimmter Class Code zum BIOS gesendet, woraufhin sich alle PCI-Karten melden, die diesem Code zugeordnet sind. Der Haken dabei ist, dass sich auch hier nur diejenigen Karten melden, denen ein Class Code vom Hersteller explizit zugewiesen wurde. Und dies ist nicht immer der Fall. Einige Hersteller von spezielleren Karten sehen ihre Geräte in keinem der vorgegebenen Klassifizierungen (siehe Tabelle) repräsentiert und weisen daher keinen Code oder einfach nur FFh (keine Gerätezuordnung) zu. Dies hat zur Folge, dass selbst ausgeklügelte Testprogramme das PCI-Gerät nicht erkennen. Gleichwohl belegt es natürlich Ressourcen und kann somit für Konfigurationsprobleme verantwortlich sein.
Die Class Codes werden innerhalb einer Klasse in Sub Classes unterteilt. Beispielsweise entspricht ein Gerät mit dem Code 0401h einem Multimedia Device (Class Code: 04h), wobei die 01h auf ein Audio Device (Sub Class) hinweist.

Class Code	Beschreibung
00_{Hex}	Rückwärtskompatibilität
01_{Hex}	Laufwerks-Controller
02_{Hex}	Netzwerk-Controller
03_{Hex}	Display-Controller
04_{Hex}	Multimedia-Device
05_{Hex}	Memory-Controller
06_{Hex}	Bridge-Device
07_{Hex}	Communication-Controller
08_{Hex}	Systemperipherie
09_{Hex}	Input-Device
$0A_{Hex}$	Docking-Stations
$0B_{Hex}$	Prozessoren
$0C_{Hex}$	Serielle Buscontroller
11_{Hex}	Data Aquisition-Device
FF_{Hex}	Keine Zuordnung

Bild 8: Zusammenstellung der Class Codes.

Allgemeines zum PCI-Bus

Als vor Jahren die ersten „Peripheral Component Interconnect“- Spezifikationen (PCI) herauskamen, haben sich viele Computer- und vor allem Messtechnik-Zusatzkarten-Hersteller diesem schnellen 32-Bit-Standard zugewandt. Sein Vorteil: Im Vergleich zum „alten“ ISA-Bus lassen sich deutlich höhere Bus-Transferraten erzielen, was in der modernen Multimedia-Technik natürlich einen entscheidenden Vorteil bedeutet. Und obwohl es unter anderem schon viele interessante Messtechnik-Boards für den PCI-Bus gibt, so steht der richtige Boom diesem Marktsegment erst noch bevor. Mittlerweile ist der PCI-Bus in jedem neuen PC als Haupt-Bus-Architektur zu finden. Der wesentliche Vorteil des PCI-Buses ist die um den Faktor 20 bis 100 schnellere Datenübertragung im Vergleich mit dem ISA-Bus. Hinzu kommt die 32-Bit-Architektur, die der heutigen Windows-Betriebssystem- und -Messtechnik-Software entgegenkommt und eine Leistungssteigerung ermöglicht. Diese Leistungssteigerung wird jedoch für den größten Anteil an Messapplikationen kaum benötigt.

Der ISA-Bus hat hervorragende Dienste als Universal-PC-Bus geleistet, aber für die modernen CPUs, die schnellen Netzwerke, Grafikkarten, SCSI-Platten und Speichermodule sind 2,5 MByte pro Sekunde einfach zu langsam. PCI verspricht deutlich über 100 MByte/s, was natürlich auch für messtechnische Anwendungen manchmal von entscheidender Bedeutung sein kann. „Plug and Play“ (PnP) vereinfacht die Installation und Inbetriebnahme. Der meist auf den PCI-tauglichen Messdatenerfassungs- oder Multifunktions-Boards vorhandene digitale Signalprozessor schafft weitere Leistungsreserven und entlastet den Host-PC. Hinzu kommt, dass schnelle A/D-Wandler nun auch Abtastraten erlauben, die um den Faktor 3...10 höher liegen als noch vor einigen Jahren und durchaus schon mit Antastraten von 100 MS/s (Megasamples/Sekunde), was einer Abtastfrequenz von100 MHz entspricht, aufwarten können.

Grundsätzlich erlaubt der Hardware- und Protokoll-Aufbau des PCI-Bus die Übertragung von 32-Bit-Datenwörtern mit einer Taktfrequenz von 33/66 MHz. Dies führt zu einem optimalen Datendurchsatz nicht nur für messtechnische, sondern beispielsweise auch für Video- oder High-speed SCSI-Anwendungen. Ein weiterer Vorteil ist das „Plug&Play“-Konzept, bei dem man die installierte PCI-Zusatzkarte nicht mehr umständlich mit DIP-Schaltern bezüglich Adressraum und Interrupts konfigurieren muss. Vielmehr wird beim Booten des Rechners das PCI-BIOS aktiviert, das alle im Rechner installierten Karten abfragt und ihre Adressbelegungen sowie Speicherplatz-Bedürfnisse festlegt. Ein eigener PCI-Chipsatz auf dem Motherboard erledigt diese Aufgaben. Auch das Bus-Mastering-Konzept erweist sich bei der schnellen Datenübertragung als nützlich: Eine Karte kann – entweder dauernd oder nur für einen bestimmten Zeitraum – die Kontrolle über den Datentransfer übernehmen und zielgerichtet die Weiterleitung der Daten steuern. Dies ist besonders für messtechnische Aufgaben sinnvoll, da dadurch die CPU (die ihrerseits natürlich auch als Bus-Master fungieren kann) entlastet wird.

Da es sich bei den Datentransfers sowohl im messtechnischen Bereich wie auch bei anderen Applikationen hauptsächlich um Übertragungen in bestimmte Speicherbereiche handelt, sind zwei Betriebsarten von besonderer Bedeutung: der „Memory-Mode“ und der „Real-Time-Mode“ (hier nur für messtechnische Applikationen betrachtet). Im „Memory-Modus“ digitalisieren die A/D-Wandler der Messdatenerfassungskarte die Analogwerte und schreiben die Digitalworte in ein eigenes RAM, das sich auch auf dieser Karte befindet. Dieser Speicher kann nun von der CPU (oder einer anderen PCI-Karte) abgefragt werden – das Ganze allerdings erst nach Abschluss der A/D-Umsetzung. Mit diesem Verfahren kann man die höchsten Speicher-Transferraten erreichen, die durchaus bei über 500 MBit pro Sekunde liegen können. Selbstverständlich ist die Länge der aufzuzeichnenden Datensätze abhängig von dem zur Verfügung stehenden On-Board-Speicher auf der PCI-Karte. Für viele Applikationen ist diese Art der Datenerfassung weitaus übertrieben. Die Regel in der Messtechnik zeigt, dass Datenraten von unter 1 MHz völlig ausreichend sind.

Im „Real-Time-Modus“ schreiben die A/D-Wandler ihre digitalisierten Messdaten nicht auf ein On-Board-Memory, sondern über den PCI-Bus entweder in den Motherboard-Speicher oder in das Memory einer anderen PCI-Karte im Rechner. Da hier die Übertragung auf dem PCI-Bus korreliert werden muss, lassen sich keine so hohen Transferraten wie im „Memory-Modus“ erreichen, dafür nutzt man aber die Flexibilität der beliebigen Übertragung in alle dem Rechner zur Verfügung stehenden Speicherbereiche – auch Festplatten. Immerhin erreicht man bei einer 8-Bit-A/D-Wandlung in diesem Modus noch eine effektive PCI-Bus-Übertragungsrate von 100 MS/s. Übrigens: Den eben beschriebenen „Real-Time-Modus“ sollte man nur verwenden, wenn die Abtastrate kleiner ist als die maximale PCI-Bus-Übertragungskapazität von 100 MS/s für 8-Bit-Abtast-Auflösung. Wenn sehr lange Datenströme aufzuzeichnen sind, ist der „Real-Time-Modus“ im Zusammenspiel mit einer schnellen SCSI-Festplatte die einzige Alternative. Will man mehrere PCI-Datenerfassungskarten in einem Rechner betreiben, ist zu berücksichtigen, dass sich die maximale Bus-Bandbreite dann auf die einzelnen Karten aufteilt, was bedeutet, dass beispielsweise beim Einsatz von zwei PCI-Erfassungskarten in einem Rechner für jedes der beiden Boards nur die Hälfte der Übertragungskapazität zur Verfügung steht.

Bustakt und Datenbreite

Gegenüber den herkömmlichen Bussen weist der PCI-Bus eine ganze Reihe von Vorteilen auf. Wesentlich ist, dass er mit Busfrequenzen von 33, 66 und > 66 MHz sowie einer Datenbreite von 32 Bit arbeitet und somit schneller Daten zu den I/O-Karten übertragen kann. Besonders vorteilhaft ist dies zum Beispiel bei den neuen A/D und D/A- Karten, obwohl diese oft nur im kHz-Bereich arbeiten. Unsere PCI-Karten arbeiten alle mit einem PCI-Takt von 33 MHz.

Automatische Adress-Vergabe

Alle PCI-Karten besitzen die sogenannte Plug&Play-Funktion (kurz PnP), was soviel bedeutet wie „einstecken und loslegen“. Dadurch erübrigt sich eine Einstellung der I/O-Adresse von Hand (zum Beispiel über Jumper). Dies hat den Vorteil, dass man mögliche Adresskonflikte ausschließen kann. Beim Hochfahren des Rechners vergibt das PCI-BIOS allen eingesteckten Karten automatisch eine freie I/O-Adresse. Durch eine im Lieferumfang enthaltene Software (die mit jeder PCI-Karte ausgeliefert wird) kann die automatisch vergebene I/O-Adresse abgefragt beziehungsweise verändert werden. Hierzu stellen wir einige Tools bereit, die Sie auf der KOLTER-CD im Verzeichnis:

X:\PCI\UTILS_PCI\PCI_TOOLS\EXE

finden.

Mitgelieferte Treiber/Utilities

- Abfrage aller eingesteckten KOLTER-PCI-Karten
- Abfrage der I/O-Adressen von KOLTER-PCI-Karten
- Windows NT, 2000, XP-Treiber
- Windows 95/98/ME-Treiber
- Utils unter DOS
- Linux-Beispiel-Source

Software-Identifikation

Alle Karten können softwaremäßig identifiziert werden, da diese neben einer fest eingestellten Hersteller-ID (zum Beispiel für KOLTER = 0x1001) auch über eine Produkt-ID verfügen.

Funktionsweise der Adressvergabe

Alle PCI-Karten haben einen 64 Byte großen Konfigurationsheader, in dem die Hersteller-ID, die Produkt-ID, sowie Informationen darüber enthalten sind, ob die PCI-Karte Memory und/oder I/O-Bereiche benötigt- und wie groß dieser Bereich sein sollte. Beim Hochfahren des Rechners (Booten), checkt das BIOS des Motherboards alle PCI-Karten durch und verteilt systematisch freie I/O- bzw. Memory-Adressen auf den Karten. Die Adresslage jeder im PC befindlichen Karte wird also vom Motherboard bzw. BIOS vorbestimmt und nicht, wie sonst üblich, per Jumper oder Switch auf der Steckkarte festgelegt. Die zugeteilten Adressen werden in den dafür vorgesehenen Registern auf den PCI-Karten selber redundant zwischengespeichert. Anschließend werden die PCI-Karten entsprechend aktiviert und dekodieren sich auf den eingestellten bzw. zugewiesenen Adressraum.

Wie bekommt man die Kartenadresse heraus?

Jede PCI-Karte kann durch die Hersteller-ID und Produkt-ID identifiziert werden. Unseren Karten liegt dazu ein spezielles Programm bei: PCI_INF.EXE im CD-Verzeichnis:\PCI\Utils_PCI\pci_tools\exe, dass die Informationen sichtbar macht.

Das Programm gibt folgende Informationen auf dem Monitor aus:

Nr.	**Nummer des PCI-Slot (nicht sichtbar)**
Hersteller-ID	**Name des Herstellers**
Produkt-ID	**Name bzw. Nummer der Karte**
I/O-Adresse	**Hardware-Adresse, unter der die Karte angesprochen wird**
MEM-Adresse	**Basisspeicheradresse (nicht sichtbar)**
Produkt	**Name der PCI-Karte**

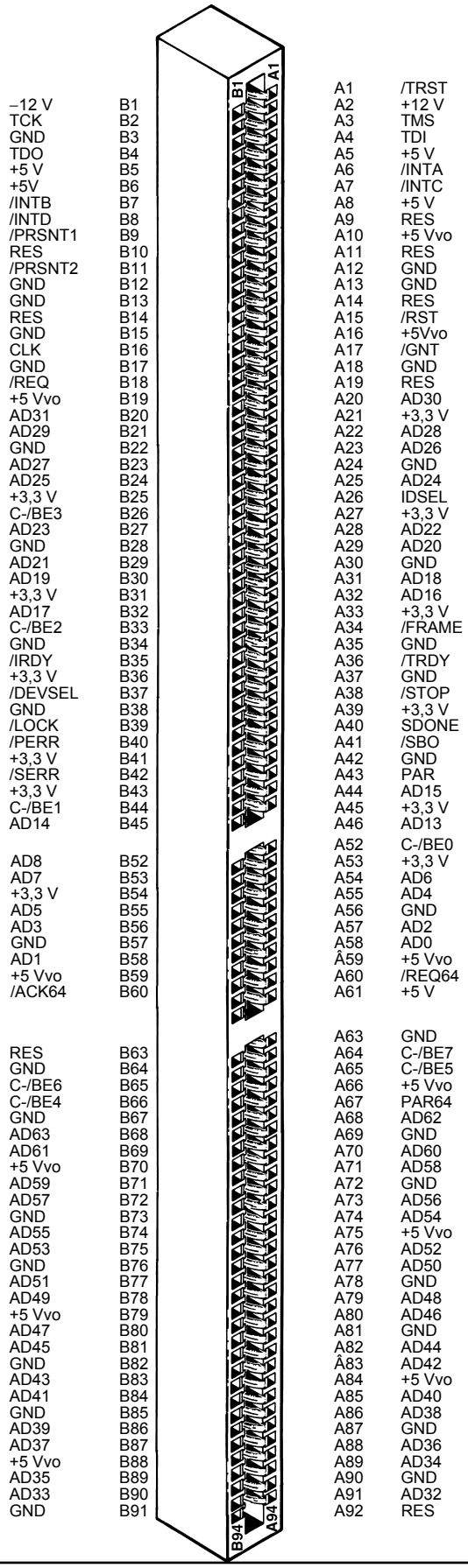
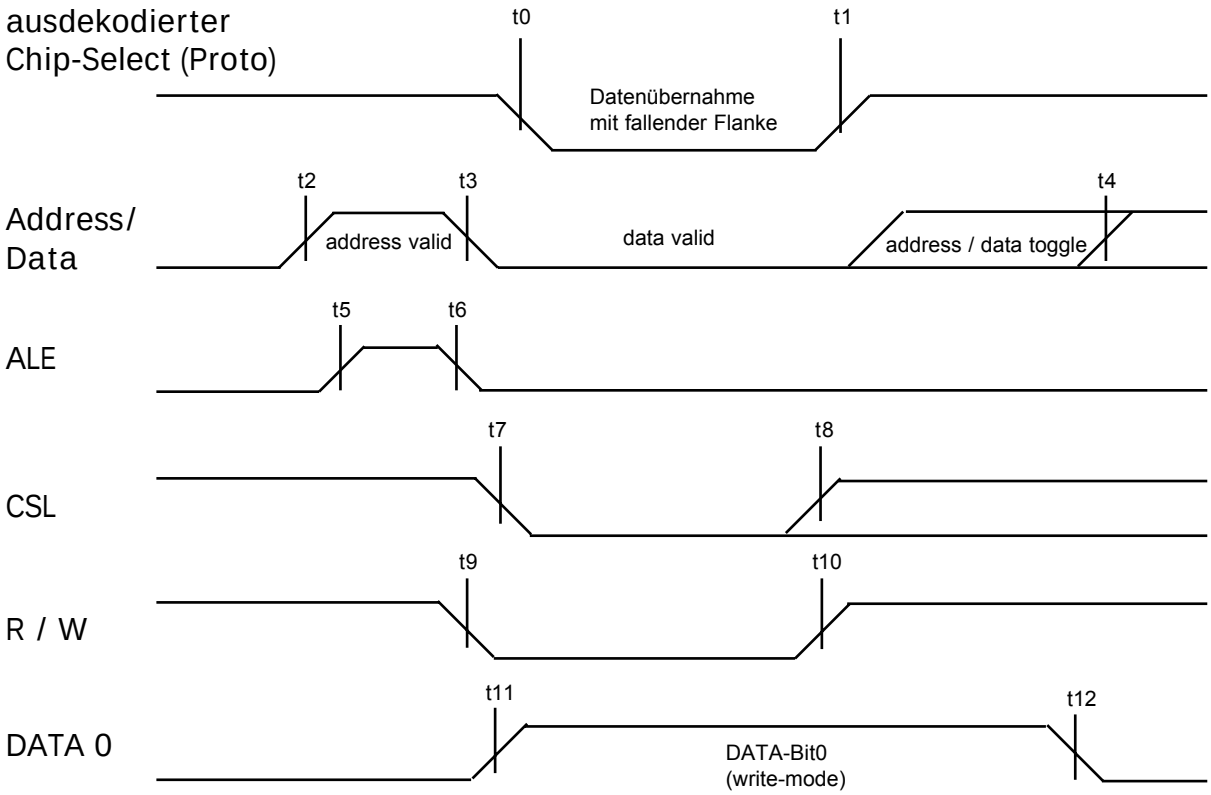


Bild 9. Die Signalbelegung am PCI-Slot.

Signalbeschreibung und Timing des Target-Controllers
PCI-Decoder (ispLSI1032E mit Kolter PCI-Core rev. 03)

PIN	Signal	Beschreibung / Ereignis
D0...D15	high	Adress- und Datenbus, wird über PCIALE gemultiplext (data/address) verfügbar sind 16 Datenleitungen und 8 Adressleitungen
PCI ALE	high	Latch-Signal auf Flanke um Adressen freizugeben (latch address)
PCI ALE	low	D0...D15 Daten-bits floaten (data access)
PCI RW	high	Daten werden gelesen (I/O-read)
PCI RW	low	Daten werden geschrieben (I/O-write)
PCI CSL	low	unteres Datensegment D0...D7 aktiv (data-low-byte enable)
PCI CSH	low	oberes Datensegment D8...D15 aktiv (data-high-byte enable)

Timing des User-Bus (neuer TCB rev. 03) nur für ispLSI1032E - code Exxxxxxx

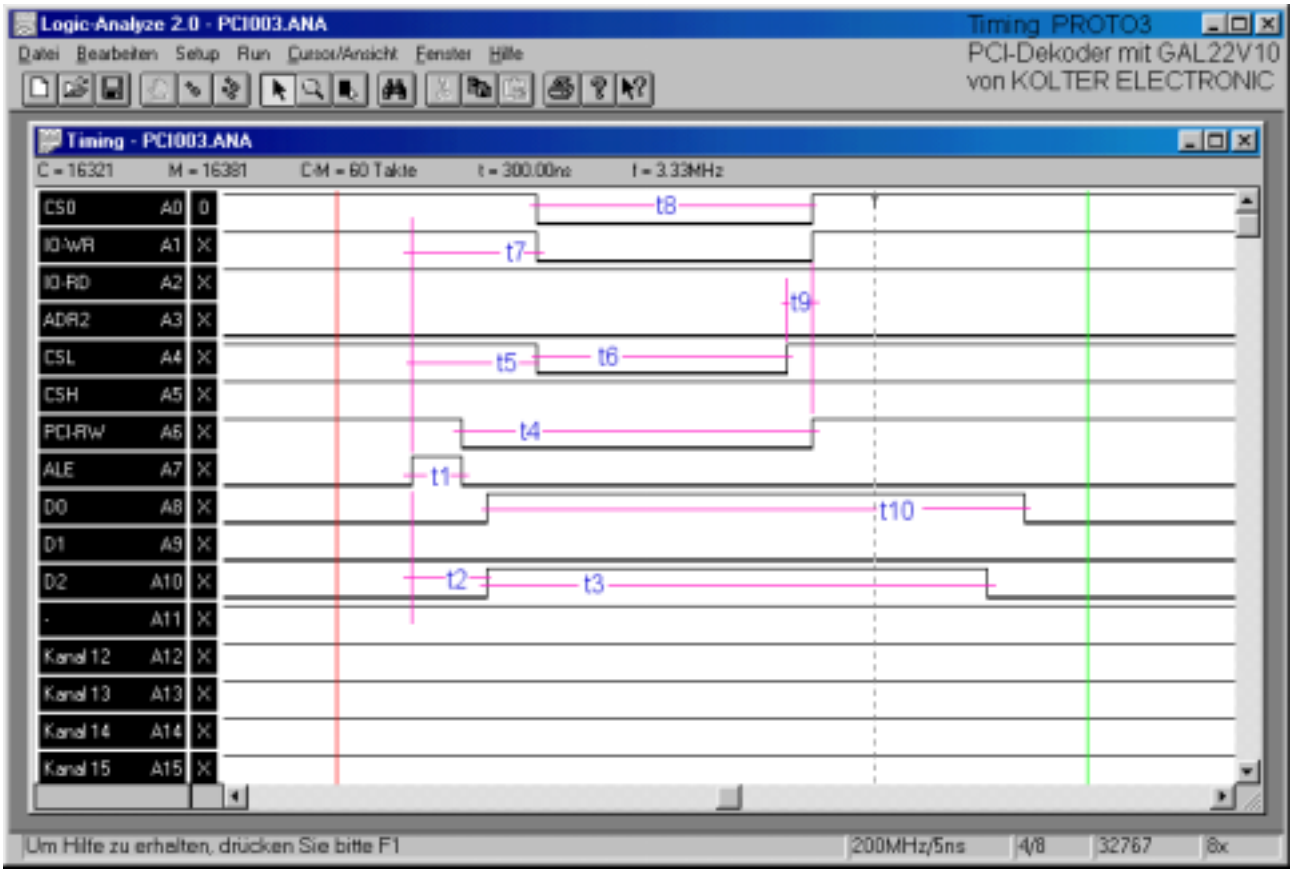


Zeit-Tabelle Anmerkung: ~ von GAL-Laufzeit abhängig; # von PCI-Bus abhängig

t0	-	t1	#~	100 ns	chip select (decode by GAL22V10 /CS1)
t2	-	t3	#	33 ns	address valid
t3	-	t4	#	160..200 ns	data valid
t5	-	t6	#	20 ns	address latch enable PCI-ALE
t7	-	t8	#	105 ns	chip select low PCI-CSL
t9	-	t10	~	140 ns	read / write PCI-RW (write = low)
t10	-	t1	~	5...10 ns	
t7	-	t0	~	5...10 ns	
t8	-	t1	~	5...10 ns	
t6	-	t3	~	1 ns	ALE < adress valid
t6	-	t9	~	0 ns	
t5	-	t7	~	66 ns	
t11	-	t12	#	180 ns	

Änderungen vorbehalten

Das original Timing des PCI-Dekoders mit nachgeschaltetem Dekodier-GAL sieht folgendermaßen aus:



Folgende Zeitzusammenhänge konnten bei einem 8-bit write-port mit einem 200 Ms/s. Logic-analyzer real ermittelt werden:

t1 = 20 ns ALE
t2 = 30 ns ready for data
t3 = 175 ... 200 ns (data-valid siehe t10)
t4 = 140 ns RW
t5 = 45 ns ready for CS
t6 = 105 ns chip-select low (low byte D0..D7)
t7 = 45 ... 50 ns ready for write
t8 = 100 ... 110 ns chip-select 8255
t9 = 10 ns CSL to chip-select

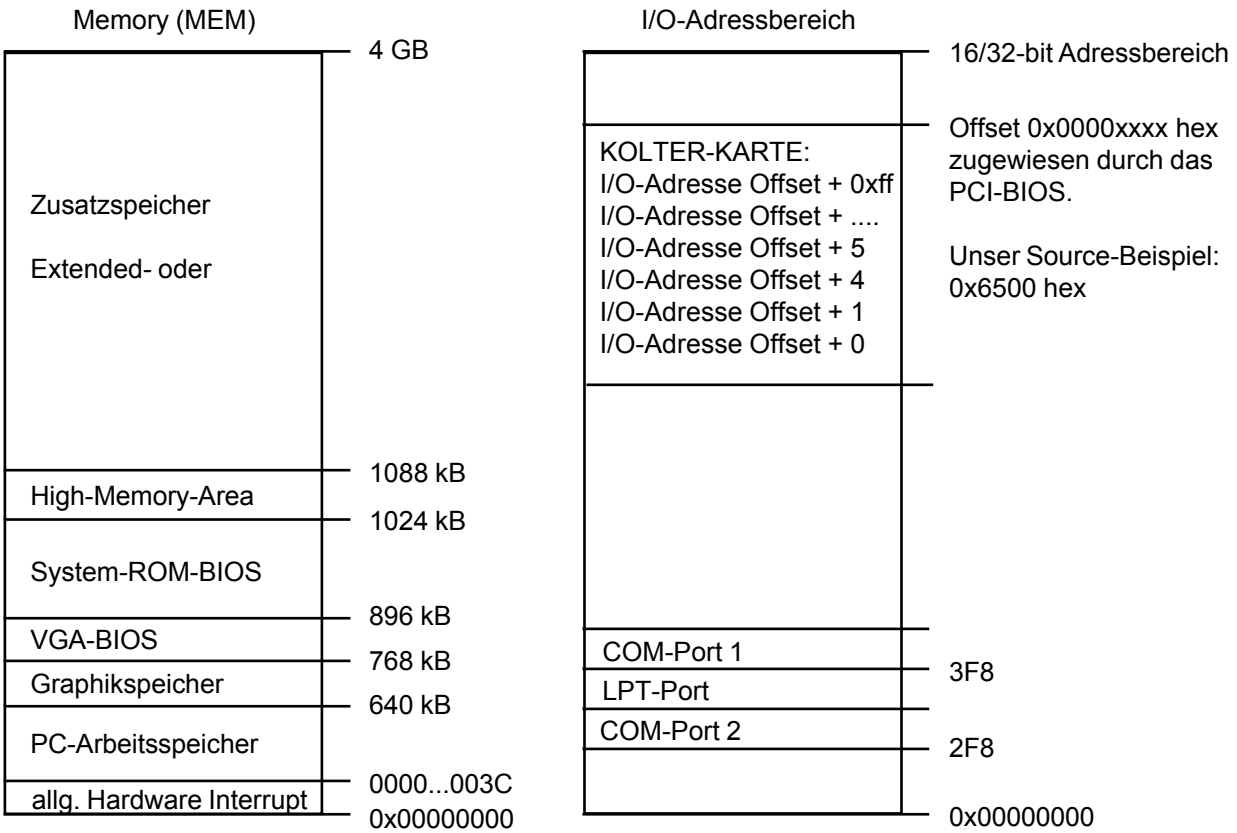
Der TCB-Block (im core) verlängert die Datengültigkeit um einen PCI-Clock-Zyklus (+33 ns), damit auch auf ältere Chips wie den 8254, 8255 u.a. ein zeitunkritischer Zugriff erfolgen kann. Eine verbindliche Zusage zur Einhaltung des Timings ist leider nicht möglich, da nicht jeder Motherboard-Hersteller sich genau an die PCI-Spezifikation 2.1 oder 2.2 hält. Es sind teilweise Abweichung um bis zu 50 % bei verschiedenen Boards gemessen worden. Die ursprüngliche Timing-Vorgabe können Sie aus den PCI-Spezifikationen der PCI-SIG Gruppe auf der Homepage www.pcisig.com beziehen.

Adressenverteilung der PCI-Basis.- und Offsetadressen

Im Gegensatz zu ISA ist **P**eripheral **C**omponent **I**nterconnect-Bus (PCI-Bus) Plug & Play-fähig. Für alle PCI-Einheiten ist ein Konfigurationsbereich von 256 Byte vorgesehen, um beim Booten dem System kartenspezifische Eigenschaften mitzuteilen. Mit Hilfe dieses Bereiches und dem erweiterten BIOS (PCI-BIOS) wird eine automatische Konfigurierung der einzelnen PCI-Devices (PCI-Einheiten) erreicht. Zwingend ist jedoch ein mindestens 64 Byte großer Konfigurationsbereich (Configuration Space) der die wichtigsten Karten-Daten beinhaltet, damit das System eine erfolgreiche Inbetriebnahme des Device sicherstellen kann.

Die Aufteilung der Adressbereiche innerhalb der PCI-Dekodierung (Chip) ist linear. Der Offset ist dagegen von der verwendeten Rechnerkonfiguration abhängig und unterscheidet bzw. verändert sich, je nach Ausbaustufe der Zusatzkomponenten, da das PCI-BIOS die Zuteilung der I/O-und Memory-Resourcen beim Boot-Vorgang zwischen dem System und den PCI-Dekodern selbstständig abstimmt. Um dennoch auf eine absolute physikalische I/O-Adresse programmieren bzw. zugreifen zu können, bedient man sich diverser PCI-Tools, die den Konfigurationsheader des jeweiligen PCI-Dekoders im System auslesen (über INT) und diese Daten der jeweiligen Anwendung entsprechend mitteilen, bevor dann auf dieses Device zugegriffen wird. Diese PCI-BIOS-Daten sind in jedem System dynamisch angeordnet und können nur über bestimmte Interrupt-Routinen ausgelesen werden, da jeder Motherboard-Hersteller ein anderes BIOS bzw. Memorymodell verwendet und somit keine absoluten Speicheradressen für die Konfigurationsdaten zur Verfügung gestellt werden können. Zu diesem Thema gibt es reichlich Literatur, weswegen wir auch jetzt nicht weiter auf diese Programmierweise eingehen möchten.

Die Klassifizierung einer PCI-Karte wird durch die im Konfigurationsheader enthaltenen Datensätze wie Vendor-ID und Produkt-ID erreicht. Über diese Parameter lässt sich eine Karte erkennen und entsprechend einbinden. Jeder Hersteller weltweit hat eine Vendor-ID, über die sich seine Produktserie darstellen lässt. Damit auch eine klare Zuweisung seiner Produkte im System möglich wird, besitzt jede PCI-Karte eine eigene Produkt-ID. Über weitere Datensätze, die in jedem PCI-Dekoder enthalten sind, lässt sich die jeweilige Karte noch tiefer klassifizieren.



Die interne Aufteilung des PCI-Dekoders hinsichtlich der linearen Adressverteilung ist wie folgt spezifiziert:

KOLTER-KARTE:
I/O-Adresse Offset + 0xff
I/O-Adresse Offset + ...
I/O-Adresse Offset + 5
I/O-Adresse Offset + 4
I/O-Adresse Offset + 1
I/O-Adresse Offset + 0

Der PCI-Dekoder besitzt eingangsseitig einen 32-bit PCI-Adress/Datenbus und am Ausgang einen 16-bit breiten User-Bus, der Ähnlichkeiten mit einem normalen ISA- oder Z80-Standard-Bus aufweist. Neben der State-Machine für den Datenverkehr zwischen Configaration-Space, PCI-Interface und Steuerlogik besitzt unser PCI-Dekoder noch einen TCB-Block (Timing-Correktion-Block) der das User-Bus-Timing auf ein normales 80er-Nievau konvertiert und somit jeden Peripherie-Baustein am PCI-Bus zulässt.

Der Adressbereich des PCI-Dekoders umfasst immer eine feste Blockgröße von 256 Bytes (00...FFh). Da jedoch ausgangsseitig ein 16-bit-Datenbus verwendet wird, können nur insgesamt 128 Adressen ausdekodiert werden, was jedoch völlig ausreichend sein dürfte. Alle dazwischenliegenden I/O-Adressen werden ignoriert und können bzw. dürfen nicht verwendet werden. In der fortlaufenden Adressierung bilden sich somit Lücken von jeweils 2 Byte, nach jeweils 2 Byte, die Verwendung finden. Dazu ein Beispiel:

```
var   offset     : word;
      a          : byte;

offset := 0x6500;              // nicht redundant, vom PCI-BIOS vergeben

out(offset + 0,byte);          // scheibt auf die erste Adresse der PCI-Karte
out(offset + 1,byte);          // scheibt auf die zweite Adresse der PCI-Karte
out(offset + 2,byte);          // Fehler !!!
out(offset + 3,byte);          // Fehler !!!

a := inp(offset + 4,byte);     // liest auf der vierten Adresse der PCI-Karte
a := inp(offset + 5,byte);     // liest auf der fünften Adresse der PCI-Karte
a := inp(offset + 6,byte);     // Fehler !!!
a := inp(offset + 7,byte);     // Fehler !!!

u.s.w.
```

Der Grund, warum nur ein 16-bit-Daten-Bus verwendet wurde, liegt in der begrenzten Anschlussmöglichkeit des LATTICE-Bausteins ispLSI1032 (72 I/O-Pins) sowie in der Gatterzahl von nur 6000 PLD-Gates bzw. 128 Makrozellen.

Da der cPLD-Baustein im PLCC-84-Gehäuse zur Verfügung steht, ist er ideal für schnelle und einfache PCI-Entwicklungen, die nur eine geringe Einarbeitung in das Thema PCI fordern. Eines der wichtigsten Kriterien ist die Losung: “time-to-market“, die dieser Baustein für sich reklamiert, da Prototyping und vorhandene Schaltpläne, Layouts und Know-How eine rasche Einführung neuer Produkte ermöglicht.

Bei Neuentwicklungen oder beim Re-Design von ISA-Karten auf PCI-Bus sind wir Ihnen gerne behilflich.

Allgemeine Adressierung unter PCI

In selbstgeschriebener Software kann beispielsweise auch die folgende Routine eingesetzt werden. Es handelt sich um einen allgemeinen Beispiel-Source zur Ansteuerung von Relais oder Optokopplern oder TTL-Signalen. Der Source-Code ist auf keine besondere Karte zugeschnitten und dient lediglich zur Erläuterung.

Um das Programm einzubinden ist vorher ein Make-File zu bilden. MS VC++ erledigt das für Sie.

Das folgende C-Programm-Beispiel zeigt die Adressabfage einer PCI-Karte unter Windows 9x/ME:

```
// Test-Programm

#include <stdio.h>

void main()
{
unsigned long ret;
unsigned int port;

int i;
long int j;
unsigned int wert;

port=ret=PciGetIO(0x010);

printf("Die Port-Adresse ist:%lx\n",ret);       // Adresse holen
if(port==0) exit(0);                            // Abbruch wenn nicht o.k.

while(!kbhit()) {                               // Wiederholen bis Taste
for(i=0;i<16;++i) {
  wert=1<<i;

  outp(port,wert&0xff);                         // Auf Port schreiben
  outp(port+4,(wert>>8)&0xff);                  // Auf Port schreiben

  for(j=0;j!=200000;++j);                       // Oder Sleep(500);
  printf("%x %x \n",inp(port),inp(port+4));     // Auf Monitor mitschreiben
}
}
}
```

Im Verzeichnis `PCI\UTIL_PCI\Visual_BASIC\Sourcen` auf der KOLTER-CD finden Sie zudem einen VB6-Projekt-Source mit Device-Treibern, zum Auslesen des Config-Headers. Auch hierüber lassen sich die gültigen Adressen ermitteln.

Einbindung in eigene Programme

Programmierbeispiel zur Karteneinbindung

Um die entsprechende Vergabe der I/O-Adresse braucht sich als Anwender bei PCI-Karten nicht zu kümmern. Wohl aber um Installation der Device-Treiber und die Einbindung der Adresse in sein Programm.

Die Device-Treiber werden, über ein Setup-Programm, von der CD aus in „C:\Programme“ installiert. Die CD liegt jedem KOLTER-Produkt bei kann aber auch gegen eine Schutzgebühr separat bezogen werden.

Zur Einbindung in eigene Projekte beziehungsweise zur Weitergabe einer Anwendungssoftware mit den Device-Treibern, können die entsprechenden Dateien direkt in den Projekt-Ordner kopiert werden (aus dem Verzeichnis \Pci\Utils_PCI\Visual_BASIC\Sourcen). Das Setup befindet sich auf der aktuellen CD im Pfad \Pci\Utils_PCI\Visual_BASIC\Program.

Damit das eingene Programm weiß, wo sich die neue Karte im Adressraum befindet, müssen Parameter übergeben werden. Das Programm PCIGETIO.C (im CD-Ordner:\PCI\Utils_PCI\pci_tools) ist in der Lage, dem nachzukommen. Bei Aufruf liefert es einfach die I/O-Adresse zurück.

Das Programm PCIGETIO.C

```
/*  Beispiele:
Die folgende Funktion zeigt eine Möglichkeit auf, wie die I/O-Adresse einer KOLTER-Karte
ermittelt werden kann. Als übergabewert der Funktion ist die Device ID der entsprechenden
Karte anzugeben (Beispiel: PCI16/16 = 0x0010). Der Rückgabewert der Funktion beinhaltet die
I/O-Adresse der Karte oder 0, wenn keine Karte gefunden wurde bzw. keine Plug’n Play-
Adresse vergeben wurde.
*/

#define VENDOR_KOLTER 0x1001

extern unsigned long far pascal INT_1A( unsigned long reax,
                          unsigned long rebx,
                          unsigned long recx,
                          unsigned long redi);

unsigned long int PciGetIO(Device) {

unsigned long reax;
unsigned long rebx;
unsigned long recx;
unsigned long redi;
unsigned long ret_ecx;

unsigned int slot_no;
unsigned long io_adr;
unsigned long ven_dev;
unsigned long K_ven_dev;

io_adr = 0;

/* Device und VendorID werden verknüpft */
K_ven_dev = ((long) Device << 16) | VENDOR_KOLTER;

for (slot_no = 0;(slot_no <= 0x00F8) && (io_adr == 0);slot_no += 8) {
     ven_dev=INT_1A(0xb10aL,(long) slot_no,0L,0L);

        if (K_ven_dev == ven_dev)              /* Karte von KOLTER wurde gefunden*/
        io_adr=INT_1A(0xb10aL,(long) slot_no,0L,0x0015L);
}

io_adr &= 0xFFFFFFF0;                           /* I/O-Adresse wird maskiert */
return(io_adr);                                 /* Rückgabe der I/O-Adresse  */

}
```

In eigener Software kann diese Routine wie im folgenden Beispiel eingesetzt werden:

Um das Programm PCIGETIO einzubinden ist es allerdings nötig, vorher ein Make-File zu bilden. MS VC++ erledigt das für Sie.

Das Quick-C Programm:

```
#include <stdio.h>

void main()
{

unsigned long ret;       // Variable definieren

ret=PciGetIO(0x10);      // in der long-Variable "ret" steht die I/O-Basis-Adresse

printf("%lx\n",ret);     // Adresse nur auf Monitor anzeigen

}
```

Das Programm PCIUTILS liefert über die Produkt-ID und Vendor-ID die Adresse zurück, die die Karte belegt. Das Programm befindet sich ebenfalls auf dem beigelegten Datenträger.

PCIUTIL.C

```
extern unsigned long far pascal INT_1A(      unsigned long reax,
                                             unsigned long rebx,
                                             unsigned long recx,
                                             unsigned long redi);


unsigned long pci_rd_cfg (unsigned int einheit, unsigned int adresse)
{
unsigned long inhalt;

 inhalt=INT_1A(0xb10aL,(long) einheit,0L , (long) adresse);
 return (inhalt);
}

void pci_wr_cfg(unsigned int einheit, unsigned int adresse, unsigned long wert)
{
 INT_1A(0xb10dL, (long) einheit, (long) wert, (long) adresse);
}
```

Andere Vendor-IDs

```
#if 0
PCICODE.H

  Created automatically from the web using the following URL:
    http://www.halcyon.com/scripts/jboemler/pci/pcicode
  Software to create and maintain the PCICODE List written by:
    Jim Boemler (jboemler@halcyon.com)
  This is header number 547, generated 08-21-96.

Too many people have contributed to this list to acknowledge them all, but a few have provided the majority of
the input and deserve special mention:
  Frederic Potter (frederic@cao-vlsi.ibp.fr), who maintains a list for Linux.
  Chris Aston (caston@madge.com) at Madge Networks, who furnished his companys list.
  Thomas Dippon of Hewlett-Packard GmbH, who added a huge list of vendors
  and devices _after_ it seemed the list was nearly complete.
#endif

typedef struct _PCI_VENTABLE
{
        unsigned short  VenId ;
        char *  VenShort ;
        char *  VenFull ;
} PCI_VENTABLE, *PPCI_VENTABLE ;

PCI_VENTABLE    PciVenTable [] =
{
        { 0x003D, „M-M“, „Martin-Marietta Corporation“ } ,
        { 0x0E11, „Compaq“, „Compaq“ } ,
        { 0x1000, „SYM“, „Symbios Logic Inc.“ } ,
        { 0x1001, „KOLTER“, „KOLTER ELEC. Germany“ } ,
        { 0x1002, „ATI“, „ATI Technologies“ } ,
        { 0x1003, „ULSI“, „ULSI“ } ,
        { 0x1004, „VLSI“, „VLSI Technology“ } ,
        { 0x1005, „Avance“, „Avance Logic Inc.“ } ,
        { 0x1006, „Reply“, „Reply Group“ } ,
        { 0x1007, „NetFrame“, „Netframe Systems“ } ,
        { 0x1008, „Epson“, „Epson“ } , { 0x100A, „“, „“ } ,
        { 0x100A, „Phoenix“, „Phoenix Technologies Ltd.“ } ,
        { 0x100A, „“, „“ } ,
         { 0x100B, „NSC“, „National Semiconductor“ } ,
        { 0x100C, „Tseng“, „Tseng Labs“ } ,
        { 0x100C, „“, „tseng labs“ } ,
        { 0x100D, „AST“, „AST Research“ } ,
        { 0x100E, „Weitek“, „Weitek“ } ,
        { 0x1010, „VLogic“, „Video Logic Ltd.“ } ,
        { 0x1011, „DEC“, „Digital Equipment Corporation“ } ,
        { 0x1012, „Micronics“, „Micronics Computers Inc.“ } ,
        { 0x1013, „Cirrus“, „Cirrus Logic“ } ,
        { 0x1014, „IBM“, „IBM“ } ,
        { 0x1015, „LSIL“, „LSI Logic Corp of Canada“ } ,
        { 0x1016, „ICL“, „ICL Personal Systems“ } ,
        { 0x1017, „Spea“, „Spea Software AG“ } ,
usw.
```