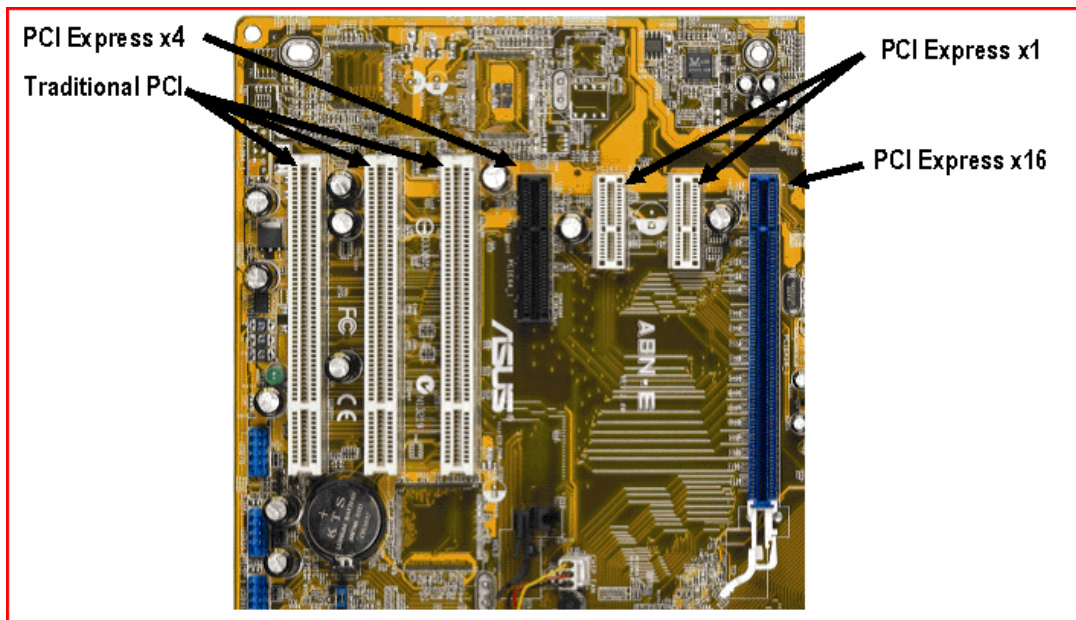


### 3. PCI-Bus (Peripheral Component Interconnect)

[www.pcisig.com](http://www.pcisig.com), [www.plxtech.com](http://www.plxtech.com), [www.plda.com](http://www.plda.com)

#### 3.1. Einführung

Die Rechnerperipherie ist direkt oder über Peripherie-Schnittstellen wie z.B. den SCSI-Bus am internen PCI-Bus des Computers (Host) angeschlossen, siehe **Bild "Busstruktur/Schnittstellen des PC"**



**Bild: Platine mit PCI-Bus Steckplätzen (Slots)**

3 Steckplätze PCI Conventional (Traditional), 4 Steckplätze PCIe (e=Express)

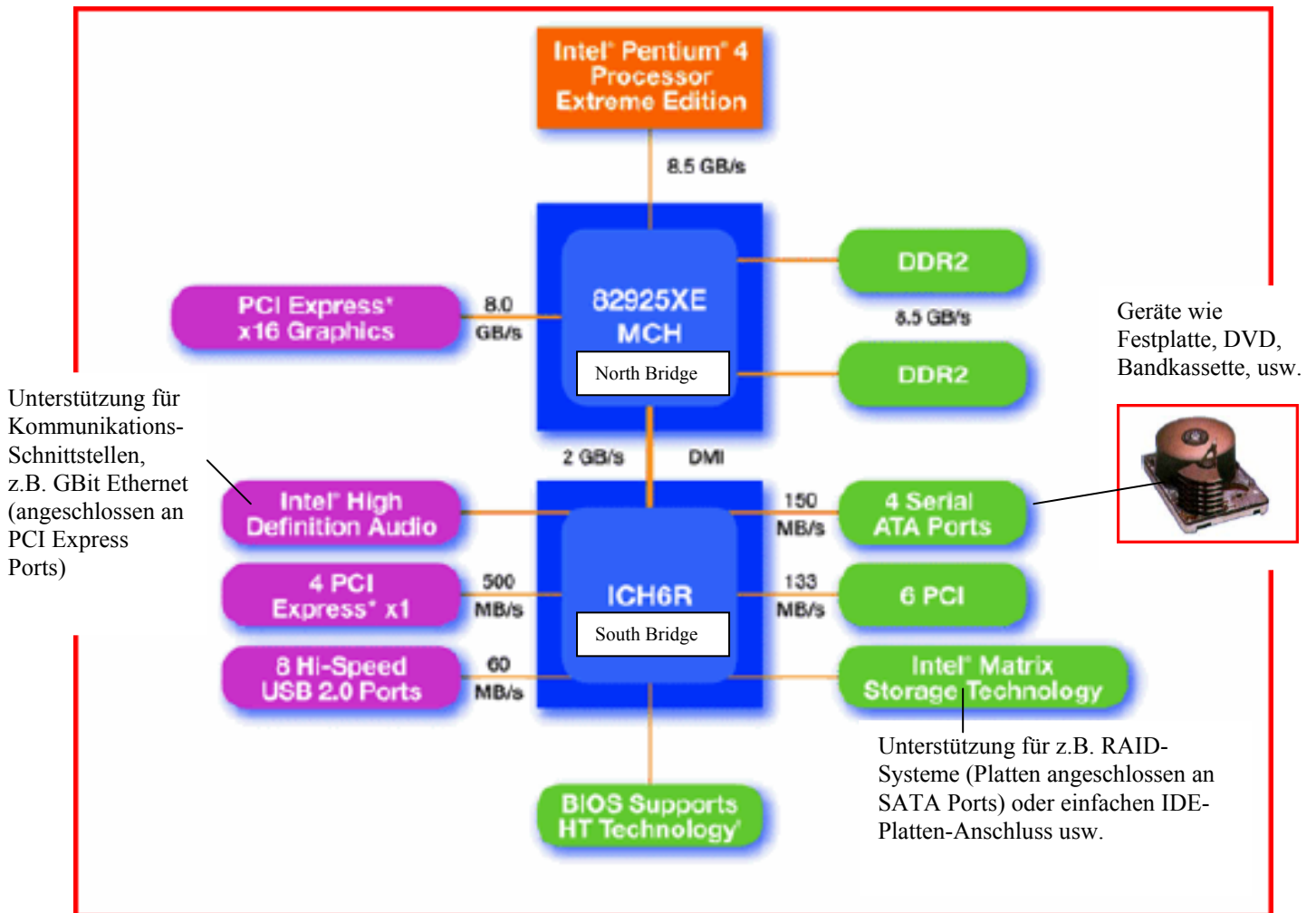


**Bild: Platine mit PCI-Bus Steckplätzen (Slots)**

*Frage: Wie ist die genaue Bezeichnung der 4 PCI-Steckplätze?*

*Frage:*

*Wieso nicht einfach die Geräteperipherie am uP-Bus anschließen? .....*



MCH Memory Controller Hub (Memory Bridge)

DMI Direct Media Interface

ICH6R I/O Controller Hub (I/O Bridge, I=Input, O=Output)

("... 8 multipurpose DMA engines, 4 input and 4 output, support of simultaneous independent data streams")

### Bild: PC-Architektur (Fa. Intel)

Über die Schnittstellen PCI, SATA und USB sind alle gängigen Peripherie-Geräte (Funktionen) anschließbar.

Weitere Schnittstellen wie PCI-X, Firewire, SAS, SCSI, RS232, Enhanced Parallel Port EPP usw. sind nicht eingezeichnet.

*Fragen: Was versteht man unter "Bus-Bandbreite"? Zahlenwerte abschätzen für PCI, USB und die Geräte!*

Der übliche Chipsatz auf dem Motherboard teilt sich in die North- und die South-Bridge auf. Eine physikalische Trennung in zwei Bausteine ist nicht unbedingt nötig, d.h., sie können in einem Chip integriert sein.

Die North-Bridge regelt die Datenströme zwischen Prozessor, Cache und Arbeitsspeicher. Dort angeschlossen sind auch der AGP-Slot für die Grafikkarte und das gesamte Bussystem.

An der South-Bridge sind die Schnittstellen, die der Chipsatz anbietet, angeschlossen. Dazu gehören USB, serielle und parallele Schnittstelle usw. .

Hersteller von Chipsätzen sind Intel, AMD, Cyrix usw.

Der PCI-Bus verbindet im Host (PC) Prozessor und Arbeitsspeicher mit der Geräteperipherie.

*Aufgabe: Bitte Tabelle ausfüllen!*

	„Traditional (conventional)“	„Express“

Der PCI-Bus ist genormt und damit herstellerunabhängig (PCI-Standard, Normierungsgremium: PCI Special Interest Group PCISIG).

PC-Karten mit der älteren, langsameren ISA-Schnittstelle (die in gewissem Umfang noch auf dem Markt sind) können am PCI-Bus über einen PCI/ISA-Adapter (PCI/ISA-Bridge) angeschlossen werden.

### 3.2. Bus-Varianten

#### PCI Conventional (alt):

- 33MHz, 32 Bit Datenbus: 132MByte/s "Basismodell"
- 66MHz, 64 Bit: 528MByte/s

→ Norm Conventional PCI 3.0

#### PCI-X:

Weiterentwicklung in Richtung höherer Frequenzen (Busbandbreite) unter Beibehaltung von Architektur, Protokoll, Signalen und Stecker des konventionellen PCI-Busses (→ Abwärtskompatibilität)

- 133MHz, 64 Bit: 1 GByte/s
- 266MHz
- 533MHz

Zusätzlich:

- 16Bit-Interface, spezifiziert für platzsparende Anwendungen.
- Fehlerkorrektur ECC mit 1Bit-Korrektur und 2Bit-Erkennung.
- größerer 4kByte-Konfigurationsbereich
- Device ID Message Transaction für Punkt-zu-Punkt-Verbindungen (z.B. zu einer Bandkassette)

→ Norm PCI-X 2.0

#### PCI Express:

-- Serieller PCI-Bus mit differentieller Übertragung der Bits als Weiterentwicklung des derzeitigen Parallelbusses. Eine 8Bit/10Bit-Codierung der Daten ermöglicht ein im Datenstrom integriertes Taktsignal

-- Pro Bitleitung 2 Adern (D+, D-) in Hin- und zwei Adern (D+, D-) in Rückrichtung = 1 Lane

Lane bezeichnet jeweils 2 Adernpaare.

Ein Link umfasst 1 oder mehrere (2x, 4x, 8x, usw.) Lanes. Über einen zentralen elektronischen Schalter (Switch) werden jeweils 2 Links verbunden und ergeben so eine Punkt-zu-Punkt-Verbindung. Maximal sind 32 Lanes pro Link möglich.

-- Datenraten:

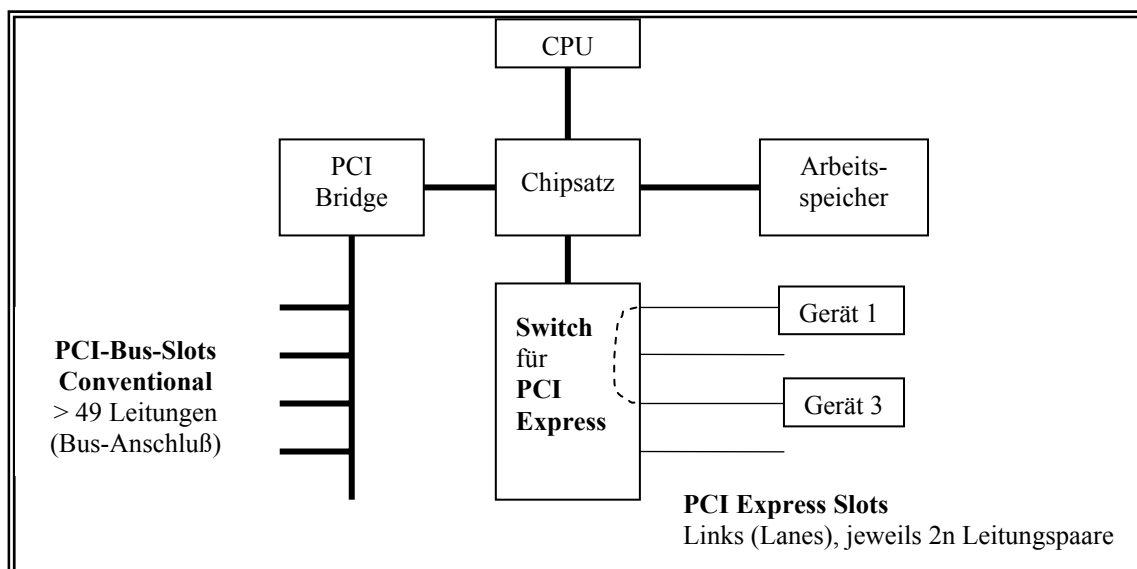
- 1 Lane (PCIex1) 250 MByte/s (2,5 GBit/s x 8/10 x 1/8 → ca. 250 MByte/s)
- 32 Lanes (PCIex32) 8 GByte/s

-- Die Daten werden paketweise übertragen (Packet-Nr, CRC-Sicherung).

-- 4 Adressräume: Memory-, I/O-, Configuration- und zusätzlich Message-Address Space (z.B. Message signalled Interrupts).

→ Norm PCI Express 1.1

#### PCI-basierte Verbindungsstruktur mit PCI-Bus und PCI-Express-Links:



Slot PCI-X: mehr Signale als PCI Conventional Slot (wegen 64 Bit Bus)

Slot PCI Ex1: weniger Signale als PCI Conventional Slot (da nur 1 Link)

### **3.3. PCI Conventional**

**Viele Aussagen gelten unverändert auch für den neuen seriellen Bus PCI Express.**

#### **3.3.1. Adressierung**

##### **Bus-Teilnehmer:**

"**PCI Device**" = Busteilnehmer ("Agent").

Ein Busteilnehmer ist vom Typ "Initiator Device" oder "Target Device" oder "Bridge Device".

Ein Busteilnehmer kann eine PCI-Steckkarte sein, z. B. ein PCI/SCSI-Adapter, oder eine auf dem "Mother Board" realisierte Funktion (z.B. USB Controller).

"**Initiator**" Device: beginnt einen Bustransfer (auch "**Bus Master**" genannt).

"**Target**" Device: vom Initiator adressierter Teilnehmer (auch "**Bus Slave**" genannt).

Jeder Bus-Teilnehmer kann bis zu 8 verschiedene funktionelle Einheiten besitzen (**Functions**, → "Multi-functional Devices").

Nach dem Einschalten ermittelt ("scanning") der Host-Prozessor über die Konfigurations-SW (BIOS, PCI Bus Enumerator) die am Bus angeschlossenen "PCI-Devices" und über deren Konfigurations-Register die Anforderungen an die Konfiguration. Mit "Anforderungen" ist gemeint, wieviel Adreßraum ein PCI-Device benötigt, ob eine Interruptleitung notwendig ist, welcher Device-Typ vorliegt und wer der Hersteller ist.

Mit den Angaben Geräteklasse (Typ) und Hersteller lädt der Host-Prozessor das Treiberprogramm. Manuelle Einstellungen über Schalter oder Steckbrücken entfallen ("plug and play").

##### **PCI-Adresse (32 Bit):**

Die Interpretation der Adresse hängt vom jeweiligen Befehlstyp im Zylus ab. Bei Konfigurationsbefehlen (Lesen bzw. Schreiben Konfigurationsregister) wird die Adresse wie gezeigt als Bus|Device|Function-Adresse interpretiert. Die Host/PCI-Bridge dekodiert die Device-Adresse (AD[15..11]) und aktiviert das entsprechende IDSEL-Signal für den dazugehörigen Slot:

HOST→Bridge: Device-Adresse -----Bridge→PCI-Bus: IDSEL-Signal

##### **Konfigurationsadresse:**

AD[23..16]=	<b>Sekundärer PCI Bus (Kaskadierung)</b>	{ IDSEL-Signal notwendig, da Konfig-Adressraum
AD[15..11]=	<b>PCI Device (max. 32 Geräte am Bus)</b>	{ aller Teilnehmer Parallel
AD[10..8]=	<b>PCI Funktion (max. 8 Funktionen innerhalb eines Geräts)</b>	
AD[7..2]=	<b>Konfigurations-Register (1 aus 64), nur für Konfigurationstransfers</b>	
AD[1..0]=	Adressierung eines Bytes in einem 4Byte-Wort	

Bei „normalen“ Memory-Read/Write-Befehlen wird die ganze Adresse als Speicheradresse interpretiert (Adressraum: Arbeitsspeicher+PCI-Speicherraum)

Signal IDSEL:

IDSEL wird realisiert mit AD[31..11] → 21 Signale, daher theoretisch 21 Geräte am Bus möglich.

Signal DEVSEL:

Alle Zugriffe, die nicht von einem PCI-Device des Busses beantwortet werden (mit Signal DEVSEL), werden auf den Erweiterungs-Bus geleitet („subtraktiver Dekodiermodus“)

### 3.3.2. Konfiguration

Die Konfigurations-SW stellt fest, wieviel Speicher- und wieviel I/O-Adressraum die angeschlossenen PCI-Device's benötigen und programmiert danach deren Speicher- und I/O-Adressdekoder so, daß keine Mehrfachbelegungen von Adressen stattfinden. Damit ist sichergestellt, daß bei Speicher- und I/O-Zugriffen vom Host-Prozessor immer nur ein Bus-Teilnehmer angesprochen wird. Sind im Konfigurations-Register Interrupts angezeigt, programmiert die SW die notwendige Routing-Information (Verbindung der PCI-Interruptleitungen INTA#..INTD# zu den Host-Prozessor-Interruptleitungen IRQ15..IRQ0). Dies kann in Abhängigkeit von der Device Address erfolgen.

Es entfallen manuelle Einstellung über Schalter oder Steckbrücken betreffend Adreßraum und Nummer der Interruptleitung.

Befehlscodes für Konfigurationszyklen:

- „Lesen Konfigurations-Register“
- „Schreiben Konfigurations-Register“

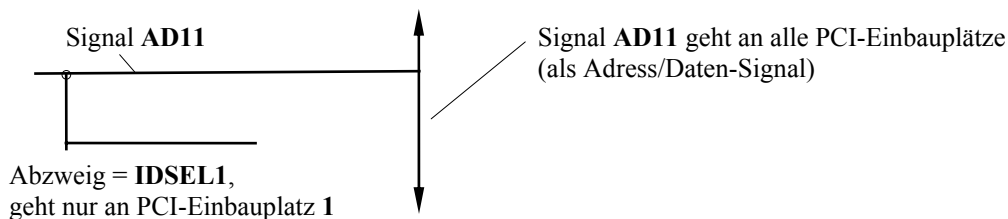
Daher existieren am PCI-Bus 3 Adressräume, die über den Befehlscode (4 Bit) im Buszyklus unterscheidbar sind:

- Configuration-Adressraum,
- Memory-Adressraum und
- I/O- Adressraum.

#### a) Abfrage der Bus-Einbauplätze (Slots)

Zu Beginn der Konfiguration wird jeder Bus-Einbauplatz über sein Signal **IDSEL<sub>x</sub>** (Initialization Device Select) adressiert, d.h., die Software fragt mit Hilfe der IDSEL<sub>x</sub>-Signale die PCI-Einbauplätze der Reihe nach ab, ob ein PCI-Device vorhanden ist (= eine PCI-Karte steckt) und liest dann die Konfigurations-Register des PCI-Device aus.

Da während der Konfigurationszyklen in Taktphase T1 die höherwertigen Adressen AD11..AD31 nicht benötigt werden können sie als IDSEL<sub>x</sub>-Signale verwendet werden (Abzweig, an den IDSEL-Pins angeschlossen), z.B.



## b) Konfigurationsregister

Die Konfigurationsdaten sind in den **Konfigurations-Registern** der Bus-Teilnehmer abgespeichert. Je nach Inhalt sind diese Register

- Nur-Lese-Register: **ROM**-Zellen, vom **Hersteller** programmiert, enthalten .....
- Schreib/Lese-Register: **RAM**-Zellen, vom **Host** beschrieben, übernehmen .....

Die **Konfigurations-Register** einer Funktion umfassen maximal 256 Bytes (= 64 doublewords DW = 64 x 4 Bytes) oder erweitert 4 kByte und sind aufgeteilt in:

- DW 00..15 **Configuration Header** (Die **schattierten Felder** sind obligatorisch)
- DW 16..63 (1k) **Device specific data**

### Konfigurations-Register, Teil "Configuration Header":

Byte 3	Byte 2	Byte 1	Byte 0	DW
Device ID (Geräte-Typ, Vergabe d. Hersteller)		Vendor ID (fest) (Hersteller, Vergabe durch PCI SIG)		00
Status Register (Gerätestatus)		Command Register Bit2=Master Enable (Bus Master)		01
Class Code (fest) (Geräte-Klasse z.B. Storage Controller/SCSI-Interface)			Revision (Geräte- Version)	02
BIST (Built-In Self Test)	Header Type <sup>3)</sup>	Latency Timer (Master, Zeit am Bus)	Cache Line Size	03
Base Address 0				04
Base Address 1				05
Base Address 2				06
Base Address 3				07
Base Address 4				08
Base Address 5				09
				10
				11
Expansion ROM Base Address (ROM des PCI-Device mit BIOS, Interrupt service routines, self-test. Enthält Funktionen für die Steuerung des PCI-Devices, die nicht im zentralen BIOS-ROM des Hosts enthalten sind).				12
				13
				14
Max_Lat (Master, max. Zeit am Bus)	Min_Gnt (Master, min. Zeit für GRANT-Signal)	Interrupt Pin <sup>1)</sup> (fest, ROM-Zelle)	Interrupt Line <sup>2)</sup> (zugewiesener Host-Interr., RAM-Zelle)	15

**fest**="hard-wired", d.h., diese Informationen sind fest im Konfigurationsregister des PCI Device eingetragen (ROM).

<sup>1)</sup> Zeigt an, welchen Interrupt (INTA/B/C oder D#) das PCI Device benötigt.

<sup>2)</sup> Die Konfigurations-Software verbindet das Interruptsignal des PCI Device (INTA/B/C oder D#) mit einem Interrupt-Eingang des Hosts (Prozessor). Der Vorgang heißt "routing" und wird durch Programmieren der "Programmable Interrupt Router"-Schaltung in der Host/PCI Bridge erreicht. Der zugewiesene Host-Interrupt (IRQ0..IRQ15) wird hier von der Konfigurations-SW eingetragen.

(Zusätzlich zu den Interruptsignalen des PCI-Busses müssen auch Interruptsignale z.B. des ISA-Busses an den Host weitergereicht werden).

<sup>3)</sup> Im Register Header Type wird die Art des Device festgelegt, ob Single- oder Multifunktion, PCI-to-PCI Bridge usw. Dadurch entscheidet sich das Aussehen des Config. Space am Register 10h .

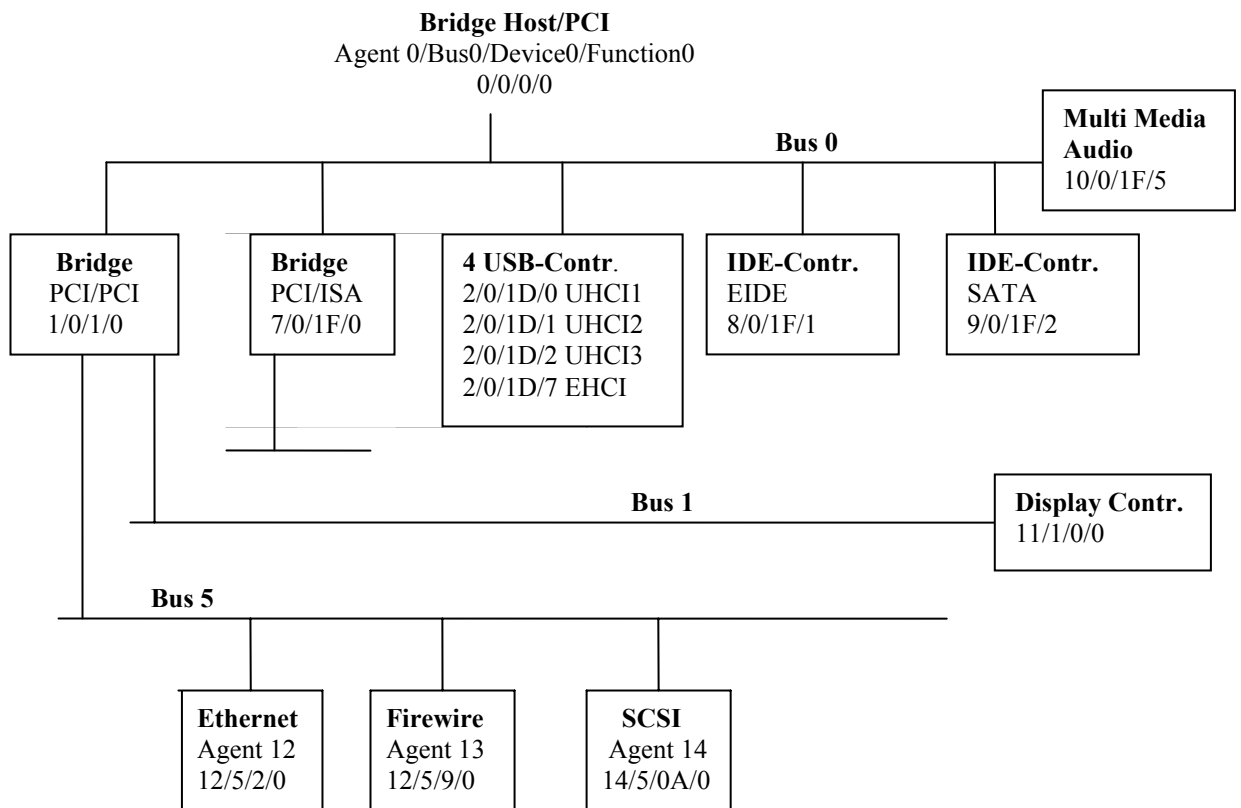
**DEMO:** Abfrage eines PCI-Busses und einzelner PCI-Devices mittels „PCI Tools“

**Diskussion**

**1) Pcmng (PCI Manager): 15 Agents, Bild an Tafel: Busstruktur**

Busstruktur Demo PC5=10.27.201.68: 4 Lasten am Bus (ohne Host-Bridge)

Agentd#	Bus#	Device#	Function#	
0	0	0	0	Host/PCI Bridge
1	0	1	0	PCI/PCI Bridge
2	0	1D	0	USBContr. UHCI#1
3	0	1D	1	USBContr. UHCI#2
4	0	1D	2	USBContr. UHCI#3
5	0	1D	7	USBContr. EHCI
6	0	1E	0	PCI/PCI
7	0	1F	0	PCI/ISA
8	0	1F	1	IDE-Contr. EIDE
9	0	1F	2	IDE-Contr. SATA
10	0	1F	5	MultiMedia Audio
11	1	0	0	Display Contr.
12	5	2	0	Ethernet
13	5	9	0	Firewire
14	5	0A	0	SCSIUltra160





Das Status Register gibt den aktuellen Zustand des PCI Device an.

Einige Bits sind reine Lese-Bits, andere Lese/Schreib-Bits, die bei Schreibzugriff gelöscht werden.

Die Bits 4 ..10 sind vom Hersteller fest verdrahtet (=Vorgaben des Entwicklers).

Die Bits 11 ..15 werden bei entsprechenden Bus-Ereignissen hardwaremäßig gesetzt.

-- Das „Capabilities List Bit“ gibt an, ob das Device einen erweiterten Konfigurationsadressraum besitzt.

-- Fast Back-to-Back Capable:

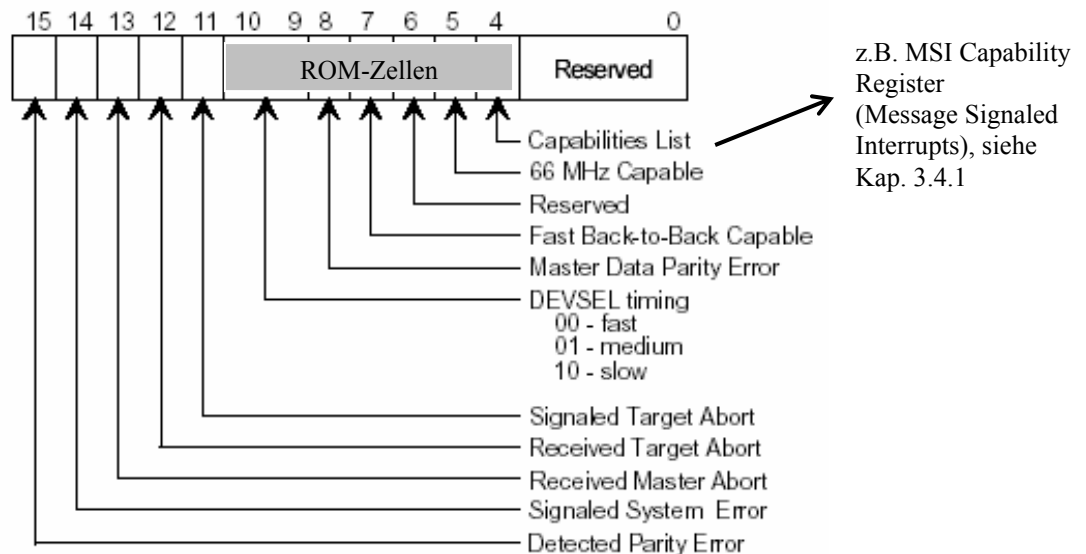
Fast Back-to-Back Capable ist '1', falls das Device diese Funktionalität unterstützt:

Fast Back-to-Back Capable: optional für Bus Master. Ist die Fähigkeit, 2 aufeinanderfolgende Busbelegungen (transactions) zu 2 verschiedenen Targets ohne idle state (1 Takt Pause) durchzuführen.

Status Reg.=1: Target ist "Fast Back-to-Back Capable" fähig

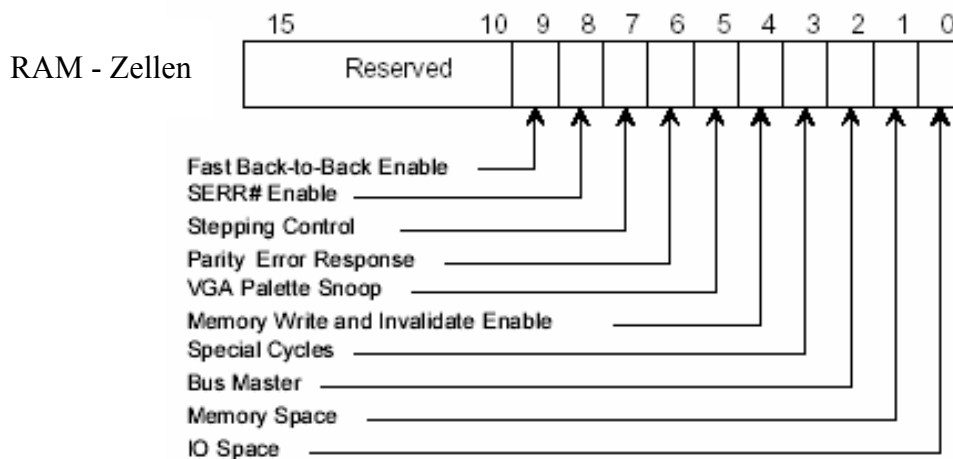
Command Reg.=1/0: Eigenschaft ist ein/ausgeschaltet

-- DEVSEL timing: Target sendet DEVSEL, wenn es angesprochen wird. Initiator bricht ab, falls kein DEVSEL innerhalb einer bestimmten Zeit kommt.



**Bild: Status Register**

Das Command Register legt das Verhalten des Device am Bus und dessen Eigenschaften fest.



**Bild: Command Register**

Bei gesetztem Bit sind die im Bild genannten Merkmale vorhanden. Die Bits lassen Lesen und Schreiben zu.

(Nach dem Reset haben z.B. die Bits „Memory Space“ und „I/O Space“ den Wert 0, d.h., der Baustein reagiert nur auf Konfigurationszyklen).



#### **d) Konfigurationszyklen**

Die CPU greift über die **HOST/PCI-Bridge** (North/South Bridge) auf den PCI-Bus zu, siehe **Bild "PCI - Architektur"**.

Für die Erzeugung der Konfigurationszyklen hält die Bridge zwei I/O-Ports bereit (I/O-Adressen im I/O-Adreßraum der CPU):

- 32 Bit **Configuration Address Port** (belegt die Adressen 0CF8..0CFB hex )
- 32 Bit **Configuration Data Port** (belegt die Adressen 0CFC..0CFF hex).

##### Schritt 1: Schreibzyklus

Die CPU schreibt die **Adresse (=Schreibdaten)** des gewünschten Konfigurationsregisters auf den I/O Configuration Address Port:

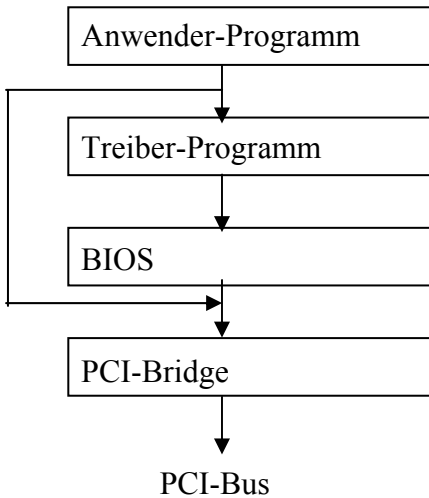
**Adresse** = PCI-Bus-Nummer (falls mehrere Busse)/  
Device-Nummer/  
Funktions-Nummer (innerhalb eines Devices)/  
DW-Adresse innerhalb des Konfigurations-Registers.

##### Schritt 2: Lese- oder Schreibzyklus

Die CPU greift mit einem **I/O-Lese- oder Schreibzyklus** auf den Configuration **Data** Port der HOST/PCI-Bridge zu. Dies veranlaßt die HOST/PCI Bridge, einen "Konfigurationszyklus für Lesen" am PCI-Bus durchzuführen und die Lesedaten im Configuration Data Port für die CPU abzuspeichern (oder einen Schreibzyklus mit den im Configuration Data Port enthaltenen Daten am PCI-Bus durchzuführen).

*Aufgabe: PCI-Bus*

### 3.3.3. Software-Schnittstelle



C-Funktionen  
Variable (Standardparameter-Übergabe)

BIOS-Funktionen  
Register-Übergabe

**2 I/O Ports für Übergabe von Zyklus-Adresse und Daten**

#### PCI-Bridge:

Wie schon beschrieben, bietet die die Bridge der CPU für die Erzeugung der **Konfigurationszyklen** zwei I/O-Ports (I/O-Adressen im I/O-Adreßraum der CPU). Greift man auf diese Ports zu, generiert die Bridge die entsprechenden Konfigurationszyklen und übermittelt eventuelle Übergabewerte zum Prozessor.

**Memory-** und **I/O-Zyklen** der CPU in den Adressraum von PCI-Devices werden von der Host/PCI-Bridge über die Adresse erkannt und auf den PCI-Bus durchgeschaltet, siehe Kap. „Buskommunikation“.

#### BIOS:

Das **BIOS** (Ein/Ausgabe-Routinen des PC-Betriebssystems) stellt für den Zugriff auf den PCI-Bus **Funktionen** zur Verfügung, z.B.

<code>pci_bios_present</code>	Abfrage: PCI-Bus vorhanden
<code>find_pci_device</code>	Karte mit VENDOR-ID und DEVICE-ID gesucht
<code>read_configuration_area</code>	
<code>write_configuration_area</code>	usw.

Die Funktions-Nummer und sonstige Parameter werden in CPU-Registern übergeben. Alle BIOS-Funktionen werden über den Interrupt 1ah aufgerufen.

#### Treiber-Programm:

<code>init-Funktion</code>	initialisiert eine Karte am Bus
<code>config_read_byte/word/dword-Funktion</code>	
<code>config_write_byte/word/dword-Funktion</code>	usw.
(C-Funktionen)	

### 3.3.4. Bus-Kommunikation

Die Kommunikation über den PCI-Bus wird über **Kommandos** eingeleitet. Ein Initiator informiert über ein Kommando andere am Bus angeschlossene Teilnehmer über den Typ des folgenden Datentransfers. Die Übertragung eines Kommandos erfolgt während der Adressierungsphase innerhalb eines Buszyklusses auf den Leitungen C/BE#[3..0]. Gleichzeitig mit dem Kommando überträgt der Initiator die Zieladresse für den folgenden Datentransfer auf den Leitungen AD[31..00], siehe Bild Lesezyklus.

Der Datentransfer kann aus einem einzigen Datenwort oder aus mehreren Datenworten bestehen (Burst-Mode). Im Burst-Mode ist das adressierte Target Device selbst für die Bereitstellung der jeweiligen Folgeadresse verantwortlich.

Zum Anschluß eines Target Device am PCI-Bus sind mindestens 47 Signale notwendig.

Ein Initiator (Master) braucht zwei zusätzliche Signale REQ# und GNT# für die Belegung (Arbitrierung) des Busses.

#### a) Bus-Kommandos:

Folgende Kommandos stehen einem Initiator zur Verfügung, wenn er einen PCI-Buszyklus durchführt (codiert in den Signalen C/BE#[3..0] während der Adressierungsphase). Durch das Kommando ist festgelegt, welcher Adreßraum (Speicher, I/O oder Konfigurationsreg.) mit der Adresse angesprochen wird.

C/BE#[3::0]	Command Type
0000	Das <b>Interrupt Acknowledge</b> Kommando zeigt einen Zugriff auf den Interruptcontroller. Dann ist der Zustand der Adressleitungen ohne Bedeutung.
0001	<b>Special Cycle</b> zeigt einen Broadcast an (geht an alle Teilnehmer)
0010	Bei <b>IO Read</b> wird von einem PCI Device, dessen Adressraum sich im I/O Bereich befindet, gelesen.
0011	<b>IO Write</b> wie IO Read jedoch mit schreibendem Zugriff.
010X	Reserved
0110	<b>Memory Read</b> leitet einen lesenden Zugriff auf ein Device ein.
0111	<b>Memory Write</b> wie Memory Read jedoch mit schreibendem Zugriff
100X	Reserved
1010	<b>Configuration Read</b> erzeugt einen lesenden Zugriff auf den Konfigurations- Speicher.
1011	<b>Configuration Write</b> erzeugt einen schreibenden Zugriff auf den Konfigurations-Speicher.
1100	<b>Memory Read Multiple</b> entspricht dem Memory Read, kündigt aber an, dass mehr als ein Datum gelesen wird.
1101	<b>Dual Address Cycle</b> wird zu Übertragung einer 64-Bit Adresse verwendet.
1110	Bei <b>Memory Read Line</b> wird eine komplette Cache Zeile übertragen.
1111	<b>Memory Write and Invalidate</b> entspricht dem Memory Write. Es wird jedoch eine komplette Cache Zeile übertragen.

**b) Bustransfers (PCI Transaction Models):**

Darstellung der Schreib- und Lesezyklen im Zeitdiagramm siehe Unterkapitel e).

**„Programmed I/O“:**

CPU ← Interrupt PCI-Device:	Meldung „Daten abholen“
CPU ← PCI-Device:	Lesezyklus der CPU (Daten in CPU-Register)
CPU → Arbeitsspeicher:	Schreibzyklus der CPU (Daten in Arbeitsspeicher)

Vor/Nachteile: Bus-Master =

**„DMA (Direct Memory Access)“:**

PCI-Device → Arbeitsspeicher	Schreibzyklus des PCI-Device
------------------------------	------------------------------

(genauer: PCI-Device → North Bridge: Schreibzyklus des PCI-Device (in Bridge-Register), von der North Bridge über die Adresse erkannt.  
North Bridge → Arbeitsspeicher: Schreibzyklus der North Bridge auf Speicherbus)

CPU ← Interrupt PCI-Device:	Meldung „Transfer beendet“
-----------------------------	----------------------------

Vor/Nachteile: belastet CPU nicht (Datentransfer ohne CPU)

Bus-Master = PCI-Device

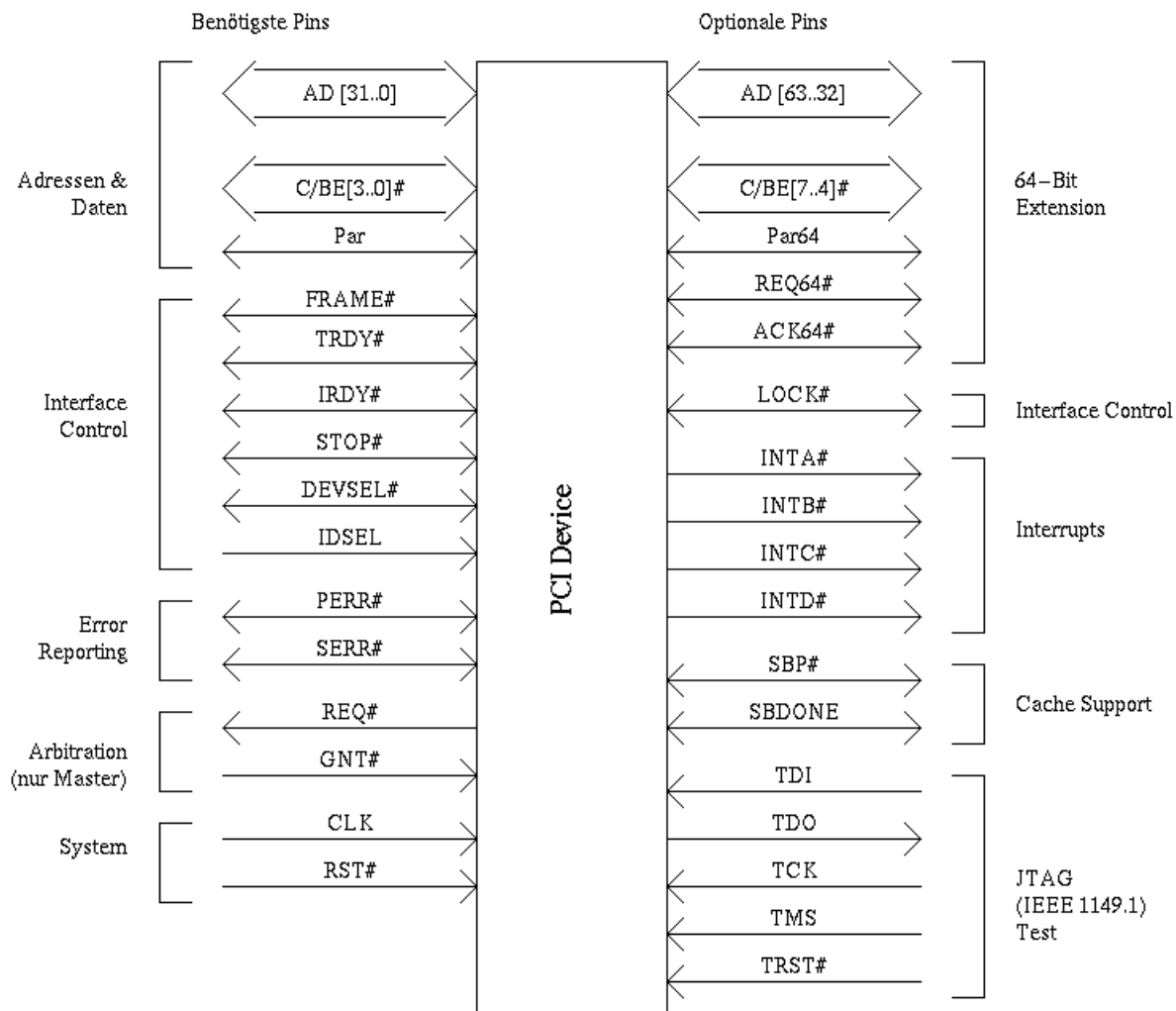
**„Device-Device (Peer to Peer)“:**

PCI-Device1 → PCI-Device2:	Schreibzyklus von PCI-Device1 mit Speicheradresse von PCI-Device2
----------------------------	-------------------------------------------------------------------

Vor/Nachteile: nur 1 Zyklus, belastet CPU nicht

Bus-Master = PCI-Device1

**c) Signale:**



- FRAME Initiator Start Signal
- TREADY Lesedaten gültig oder Schreibdaten übernommen
- IRDY bereit, Lesedaten zu übernehmen oder
- STOP Target signalisiert dem Initiator, dass er die Transaktion beenden soll
- DEVSEL Target signalisiert mit DEVSEL, dass es seine Adresse auf dem Bus erkannt hat

Beim Target Device (Slave) drehen sich die Richtungen folgender Signale um:  
C/BE#[3..0], FRAME#(nur Eingang), TRDY#, IRDY#, STOP#, DEVSEL#, PERR#(nur Ausgang)

Beim Target Device (Slave) entfallen:

REQ#, GNT#

Zusätzlich sind für Target und Initiator noch optional:

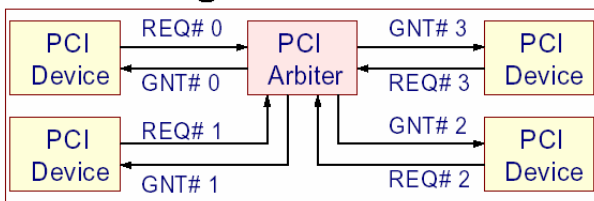
JTAG-Signale, Signale für Erweiterung auf 64Bit-Bus usw.

*Fragen:*

*Wieso sind 2 Ready-Signale IRDY# und TRDY# vorhanden ? .....*

*Erklärung und Vergleich mit uP-Bus. ....*

**d) Bus-Belegung (Arbitrierung):**



PCI-Arbiter: Bussteuerung in der PCI-Bridge  
PCI-Device: z.B. PCI-Board im PCI-Slot (Steckplatz)

Ein Initiator fordert mit dem Signal **REQ#** den Bus an und bekommt ihn, falls er frei ist, mit dem Signal **GNT#** von der Bus-Arbiter-Schaltung zugeteilt. Der Bus ist frei (**IDLE state**), wenn die Signale **FRAME#** und **IRDY#** "nicht aktiv" sind. Jeder Initiator hat eine eigene REQ#- und GNT#-Leitung zum Arbiter. Der **Bus-Arbiter** ist Bestandteil der **Host/PCI-Bridge** (der Arbiter sollte programmierbar sein hinsichtlich der Teilnehmer-Priorität).

Eine Busbelegung kann beendet werden durch den Initiator (Wegnahme des Signals FRAME#) oder das Target (Signal STOP# an Initiator)

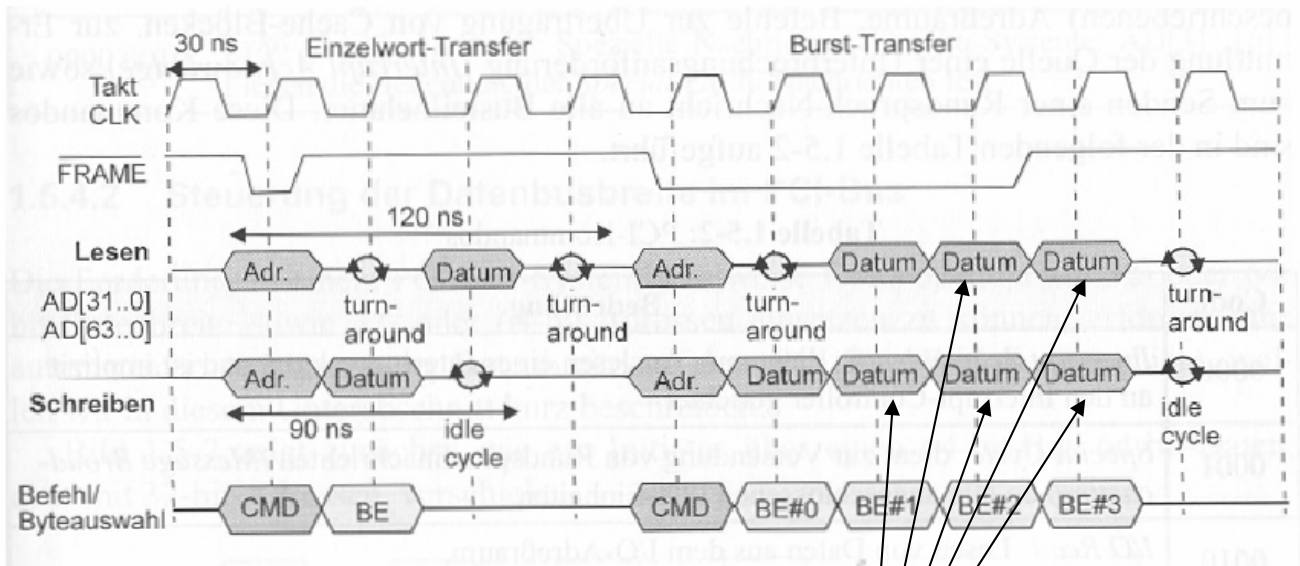
Das Konfigurationsregister des Initiators enthält eine maximale Busbelegungszeit (master latency time).

**Zeitdiagramm: Pegel-Darstellung**

Die Bussignale sind low-aktiv (#): Signal = 0 V → Signal aktiv

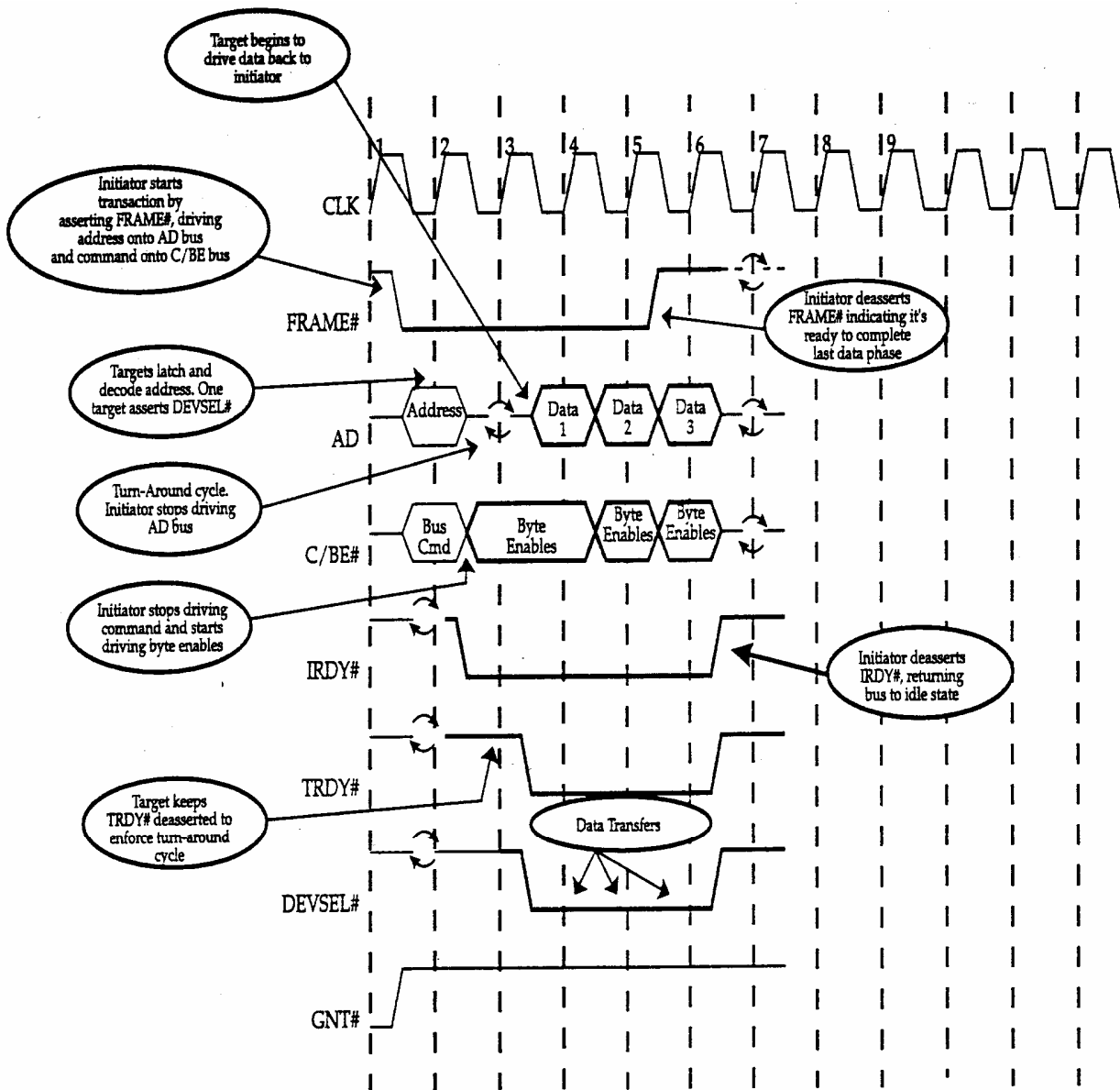
Bussignale	Aktivität Initiator	Aktivität Target
FRAME# =1 IRDY# =1	Bus frei (Idle state)	
GNT# =0	Bus wird dem Initiator zugeteilt (vom Arbiter)	
FRAME# =0 ..... AD[31..2] ..... C/BE#[3..0] .....	<b>T1:</b> Initiator startet Buszyklus durch Anlegen von FRAME# -Signal (=Start), Adresse und Kommando	Target speichert und dekodiert: Adresse und Kommando
C/BE#[3..0] ..... IRDY# =0 ..... TRDY#=1 .....	<b>T2:</b> Initiator meldet über "Byte Enable" welche Bytes gültig sind Initiator meldet: bin bereit mit IRDY#	Target erzwingt "turn around cycle" für Kontrolle des AD[31..0]-Bus (bidirektional)
DEVSEL# =0 ..... AD[31..0] ..... TRDY# =0 .....	<b>T3:</b>	Target meldet "bin ausgewählt" legt Daten auf den Bus meldet "Daten gültig"
	<b>T4:</b> Initiator usw.	Target inkrementiert die Adresse für die nächsten Daten (Burst) usw.
FRAME# =1 .....	<b>T5:</b> Initiator zeigt dem Target die letzte Datenphase an	
IRDY# =1 .....	<b>T6:</b> Initiator gibt Bus frei	

**e) Datenzyklen: vereinfacht (nicht alle Signale), Pegeldarstellung**  
**Lesen, Schreiben, Einzel- und Burst-Transfer**  
Schaltzeiten für 33MHz



Target generiert Folgeadressen selbst

**Lesezyklus** (alle Signale): **Pegeldarstellung** (# = low-aktives Signal)



Frage:  
IRDY# = 1 → ?

TRDY# = 1 → ?

### 3.3.5. Hardware

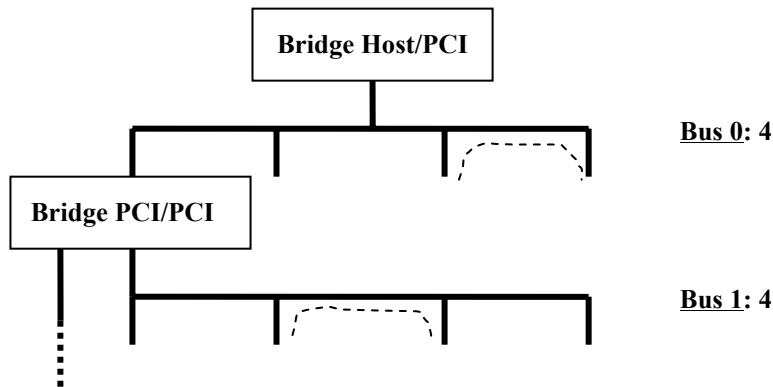
#### Bus-Takt:

Alle Aktivitäten auf dem Bus sind synchron zum Bustaktsignal PCI CLK (spezifiziert mit maximal 33MHz, maximal 66MHz usw.). Steckkarten, die z.B. für die Timer-Funktion eine genaue Taktfrequenz benötigen und die in verschiedenen Computern zu Einsatz kommen können, sollten deshalb wegen der nicht genau festgelegten Taktfrequenz eine vom PCI-Takt unabhängige Taktquelle besitzen.

#### Zahl PCI-Einbauplätze (Slots) am Bus:

Bei 33MHz und 32 Bit Busbreite sind max. 10 elektrische Lasten („loads“) anschließbar. Eine PCI-Einsteckkarte zählt 2 Lasten (Steckverbinder = 1 Last, Gatter-Ein/Ausgang = 1 Last). Da der Chipsatz (Bridge Host/PCI) auf dem Motherboard auch eine Last darstellt, sind normalerweise nur 4 Einbauplätze am PCI-Bus verfügbar. Sind mehr Einbauplätze notwendig, ist der PCI-Bus (BUS 0) mit einer Bridge zu erweitern (Bus 1 ..).

Datentransfers innerhalb von PCI-Bus 1 können gleichzeitig zu Datentransfers innerhalb von Bus 0 ablaufen.



**Bild: Kaskadieren von PCI-Bussen**

#### Compact PCI:

Der Bus ist für verschiedene Steckkarten und Stecker spezifiziert, z.B. für Europakartenformat.

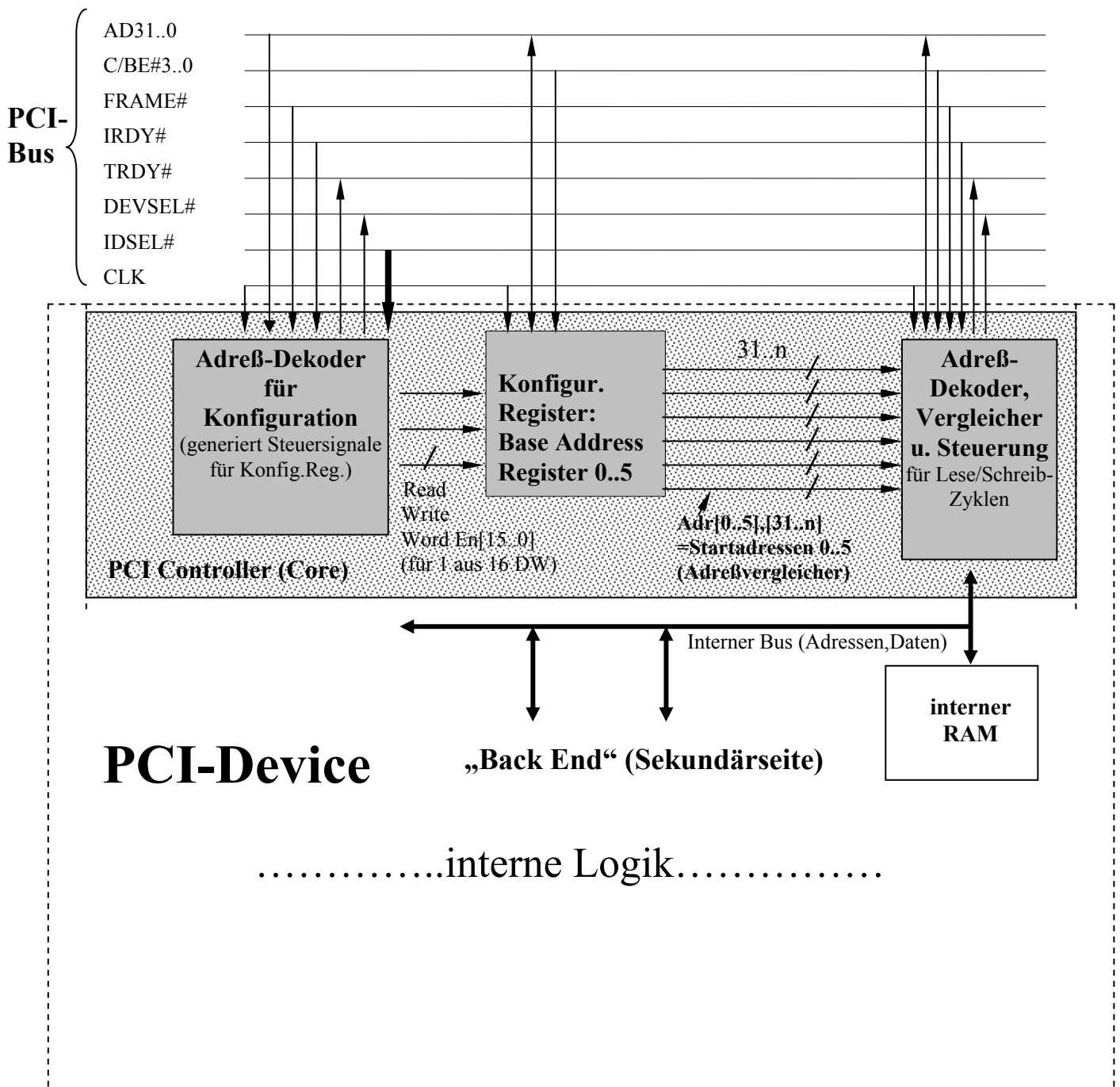
Weitere Themen:

Interrupt-Funktion / Hardware: kein Leitungsabschluß (Unterschied zu SCSI-Bus) / Adressen sind in AD31..2 enthalten. (=Grenze eines Doublewords=4Byte)

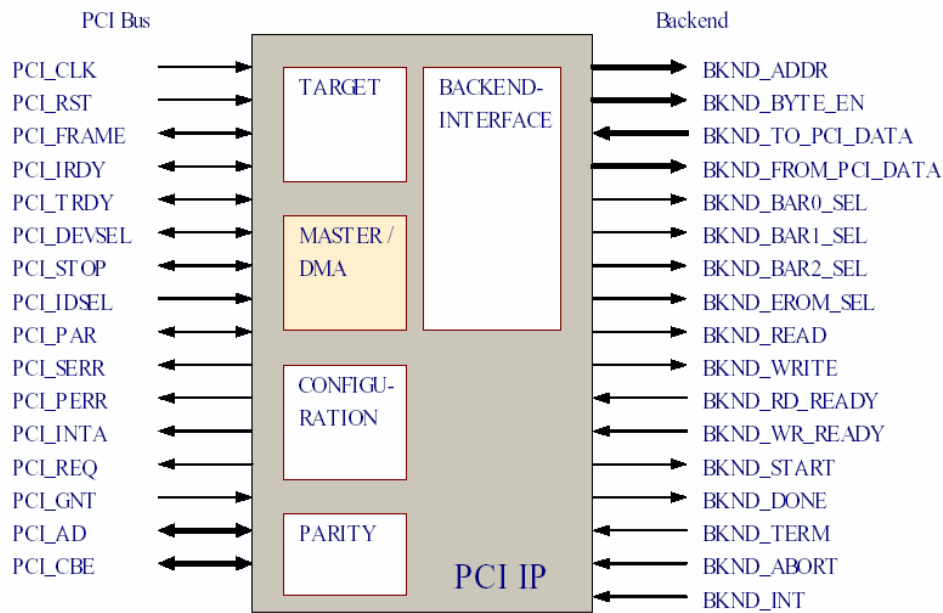
**Realisierung:**

Bei der Entwicklung einer PCI-Karte kommen derzeit zwei Methoden zur Anwendung:

- a) bei kleinen Stückzahlen werden PCI-Controller-Chips verwendet, die auf der Primär-Seite das PCI-Bus-Protokoll abwickeln und auf der Sekundär-Seite Signale (Address-, Daten-, Dekoder- und Steuersignale) für die Anwenderfunktionen zur Verfügung stellen.
- b) bei größeren Stückzahlen wird die PCI-Bus-Schnittstelle in programmierbaren Logikbausteinen wie z.B. FPGAs implementiert. Dazu existieren z.B. VHDL-Makros (PCI-Cores) für die PCI-Schnittstellenfunktionen, die normalerweise nicht kostenfrei sind.



### Darstellung am Beispiel eines PCI-Core (Fa. IMG)



IP = Intellectual Property

Vorteile der Backend-Schnittstelle für den Anwender: .....

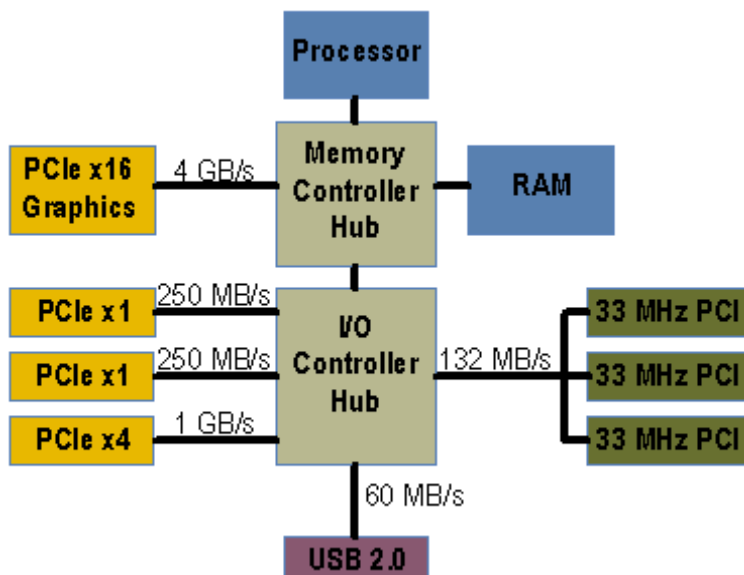
*Tabelle „Ihr Wissen über den PCI-Bus“ am Kapitel-Anfang ergänzen/korrigieren!*

### 3.4. PCI Express

Der PCI-Bus stellt das „Rückgrat“ im PC und in anderen Computern dar. Die steigenden Anforderungen an den Bus wie höhere Datenrate, begrenzte Pinzahl für den Bus, Datentransfer in Realzeit für isochrone Daten, Power Management (Umschalten eines PCI-Device in Power-Down-Mode) usw. sind mit den bisherigen Konzepten von PCI Conventional und auch von PCI-X nicht oder nur teilweise erreichbar. Deshalb wurde mit dem PCI Express ein neues Konzept entwickelt, das die obigen Forderungen für die nächste Zukunft erfüllen soll.

	Anforderung	PCI Conventional, PCI-X	PCI Express
<b>Datenrate</b>	hoch, aber skalierbar (wegen Pinzahl) Festplatte, Gigabit-LAN, Grafikkarte usw.	PCI Conv: 132 MByte/s  PCI-X: 1 GByte/s bei 64 Bit, 133 MHz	PCIe x1: 250 MByte/s bis PCIe x32: 8 GByte/s (auf Basis von 2,5 GBit/s Übertragungsrate)
<b>Pinzahl (Kabel)</b>	möglichst klein	PCI Conv: ca. 80 Pins (49 Sign.+GNDs+Vcc) PCI-X: ca. 110 Pins	PCIe x1: 8 Pins (4 Sign. +GNDs+Vcc) PCIe x8 : 40 Pins (32 Sign. +GNDs+Vcc) angepasst an Datenrate
<b>Datentransfer in Realzeit</b>	keine Verzögerung	alle PCI-Devices teilen sich die Busbandbreite → <u>kurzzeitige Verzögerungen</u>	Punkt-zu-Punkt-Verbindung, für Hin- und Rückrichtung getrennt → <u>keine Verzögerungen</u>
<b>Power Management</b>	reduzierte Stromaufnahme falls kein Datentransfer		

Der PCI Express ist software-kompatibel zum PCI Conventional (Spec. 2.3). Die Anwenderprogramme müssen nicht geändert werden.



**Bild: Busstruktur einer CPU (PC) betreffend PCI-Steckplätze (Kombination PCI Express und PCI Conventional)**

Jeder PCI-Express-Steckplatz verfügt über die angegebene Busbandbreite zum RAM-Arbeitsspeicher wogegen sich die 3 PCI-Conventional-Steckplätze (33MHz PCI) die Bandbreite von 132 MB/s des Busses „teilen“ müssen.

### 3.4.1. Physikalische Ebene

Im Gegensatz zu den Parallelbussen PCI Conventional und PCI-X ist der PCI Express ein serieller Bus mit differentieller Datenübertragung. Im Minimalfall PCIe x1 (250MByte/s) besteht der Bus aus nur 4 Signaladern. Er ist entsprechend den Anforderungen modular erweiterbar (skalierbar) auf PCIe x2/ x4/x8/ x12/ x16 und x32.

Damit bei einer Datenübertragung auf nur wenigen Adernpaaren ein hoher Datentransfer zustande kommt, ist eine hohe Datenrate (Übertragungsfrequenz) notwendig. Mit den bei den Gigahertz-Prozessorboards gemachten Erfahrungen sind jetzt Übertragungsraten pro Signal von 2,5 GBit/s und mehr möglich.

Weitere Signale wie z.B. Interrupt- Fehler-, Buszuteilungs- und Fehlersignale sind nicht vorhanden. Diese Funktionen werden von Nachrichten-Paketen (Messages) übernommen.

Beispiel Interrupt:

Da Interrupt-Signale nicht vorhanden sind erfolgt die Signalisierung über Message Signaled Interrupts MSI. Ein MSI ist eine Schreib-Transaktion. Sie unterscheidet sich von einer „normalen“ Schreib-Transaktion (Memory Write Transaction) nur durch die Ziel-Adresse (Target Memory Address, reserviert vom System für Interrupt-Meldungen) und den Datencode. Beide werden bei der Konfiguration vom Host in das MSI Capability Register geschrieben, siehe auch Kap.3.3.2 b) Konfiguration, Stausregister, Capability List.

#### Signale:

Nur differentielle Datensignale (D+, D-).

**Lane** = 2 Signalpaare: je ein differentielles Signalpaar (D+, D-) für jede Übertragungsrichtung

2,5 GBit/s pro Übertragungsrichtung ergibt  $2,5 \text{ GBit/s} * 8/10 * 1/8 = 250 \text{ MByte/s}$

(8/10 wegen 8Bit  $\rightarrow$  10 Bit Codierung, siehe auch Kapitel Serial Attached SCSI)

(5 und 10 GBit/s sind in Entwicklung)

**Link** = physikalische Verbindung zwischen 2 PCI-Devices, bestehend aus  $\geq 1$  Lane.

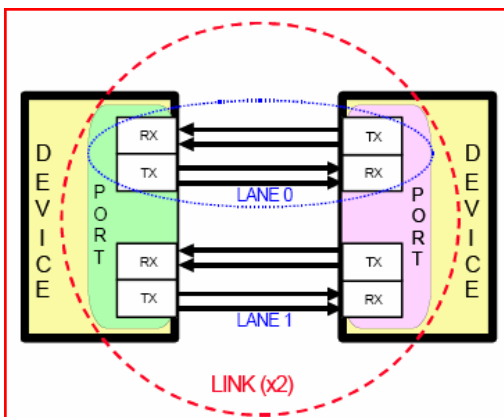
Da der Bus eine Punkt-zu-Punkt-Verbindung ist, gehört die Datenrate eines Links

ausschließlich zu dieser Verbindung und muss nicht mit anderen Geräten geteilt werden wie dies beim PCI Conventional der Fall ist.

#### Beispiele für PCI Express Links:

PCIe x1 = 250 MByte/s

max. PCIe x32 = 8 GByte/s



#### Bild:

**Link PCIe x2 = Lane 0 + Lane 1**

Besteht ein Link aus mehreren Lanes (x2, x4, x8 usw.), dann werden die zu übertragenden **Daten byte-weise auf die Lanes aufgeteilt**.

hier:

Lane 0: Byte 0, Byte 2, Byte 4, usw.

Lane 1: Byte 1, Byte 3, Byte 5, usw.

eines Datenblocks

**Switch (Schalter), PCI Express Fabric (PCI Netzwerk):**

verbindet zwei PCI-Devices miteinander (im Bild von Kap. 3.4 im Controller Hub enthalten)

PCI-Devices mit unterschiedlicher Lane-Zahl, z.B. PCIe x2 und PCIe x4, können miteinander verbunden werden auf Basis der kleineren Lane-Zahl (hier PCIe x2).

**Power Management:**

Es sind 4 Zustände Full on/ Light Sleep/ Deep Sleep und Full Off definiert (Zustandsübergänge: 200 us bis 10 ms).

**Power Budgeting:**

Bei der Konfiguration wird erkannt ob ein PCI Device hinsichtlich Stromaufnahme (und Kühlung) anschließbar ist.

Signalpegel: .....

Stecker mit Signalbelegung: .....

**3.4.2. Adressierung**

Es existieren wie bei PCI Conventional eigene Adressräume für Konfiguration, für Memory (4 GByte bei 32 Bit Adressierung bzw. 16 Exabyte bei 64 Bit Adressierung) und für I/O.

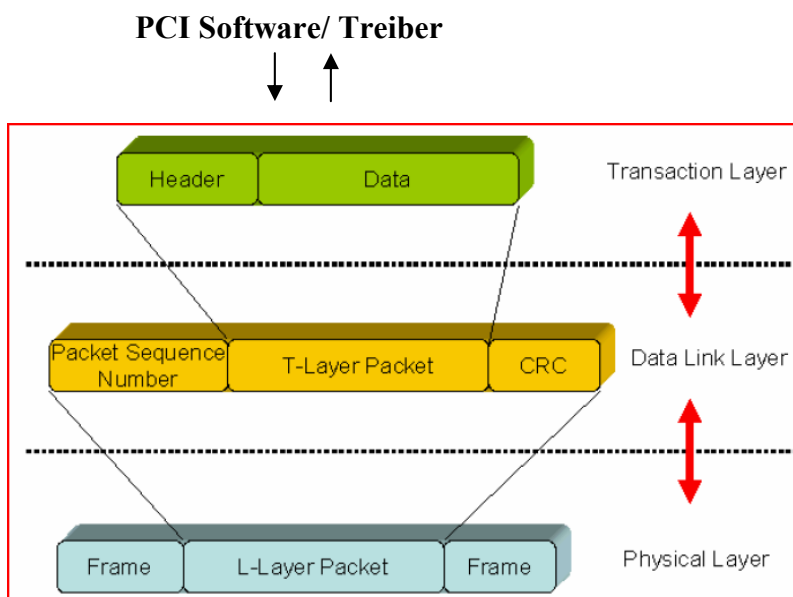
Der Konfigurationsraum wurde von 256 Byte auf 4 kByte erweitert. Die ersten 256 Byte des Konfigurationsregisters sind identisch zu PCI Conventional.

Zusätzlich gibt es einen Adressraum für Messages (z.B. Interruptsignalisierung, Fehlersignalisierung, usw.).

**3.4.3. Konfiguration**

Die Konfiguration ist kompatibel zum PCI Conventional (PCI-compatible Configuration Mechanism). Sie verwendet dieselben Ports (Configuration Address Port, Configuration Data Port) und dieselbe Adressierung. Daneben existiert ein PCI Express Enhanced Configuration Mechanism.

**3.4.4. Architektur und Buskommunikation**



**Bild:**  
**Schichtenmodell und Datenpakete**  
Die PCI Express Spezifikation definiert ein Schichtenmodell für die Entwicklung von PCI Devices

Die Schichten werden im PCI Device beim **Senden** von **Requests** von oben nach unten und beim **Empfangen** von **Requests** von unten nach oben durchlaufen.

**Transaction Layer:**

Erhält Lese- und Schreibenanforderungen (Requests) von der Software und generiert daraus sog. Request Packets (Header+Data) für die Busübertragung

**Data Link Layer:**

Fügt eine fortlaufende Packetnummer und eine CRC-Fehlersicherung hinzu.

**Physical Layer:**

Fügt eine Start-Kennung (Frame-Bitmuster links) und eine Ende-Kennung (Frame-Bitmuster rechts) hinzu.

**Header-Feld (max. 16 Byte):**

Die Inhalte hängen von der Transaktionsart (Memory-/ IO-/ Configuration-/ Message- oder Completion Request) ab und sind sehr vielfältig. Einige wichtige Inhalte sind:

- PCI Express Command
- Transaction Type (Read/ Write)
- Requester-ID (Bus/ Device/ Function Nummer des sendenden PCI Device)
- Completer-ID (Bus/ Device/ Function Nummer des empfangenden PCI Device)
- Transferclass TC des Pakets (Priorität beim Durchschalten des Pakets zum empfangenden PCI Device)
- Länge des nachfolgenden Datenfelds
- Paket-Nummer (dieses 8Bit-Tag wird benötigt zur Identifizierung ausstehender Pakete)
- alternativ :
- Memory-Adresse
- usw.

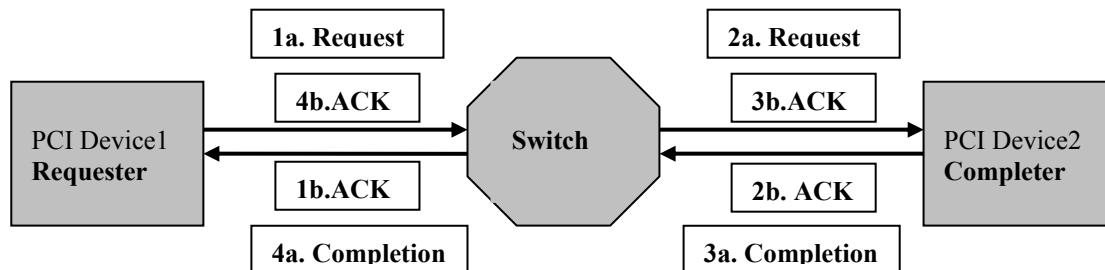
**Data-Feld (max. 1024 Byte):** Enthält die Daten.

**Transaction:**

Umfasst ein oder mehrere Pakete, um einen Informationstransfer durchzuführen.

### Split Transactions, Completions:

Wie schon beim PCI-X können Transaktionen über den Bus sog. Split-Transaktionen sein, d.h., bei einer Anforderung (Request), die vom Empfänger nicht sofort bearbeitbar ist, wird vom Empfänger nur der Erhalt der Anforderung mit einem Completion-Paket quittiert. Das Ergebnis der Anforderung (z.B. Lesedaten) wird erst später gesendet. Dadurch werden Verbindungswege und Datenpuffer nicht unnötigerweise blockiert.



**Bild: Split Transaction am Bspl. eines Memory Read (Device1 liest aus Speicher von Device2)**

Reihenfolge: 1a→1b→2a→2b→3a→3b→4a→4b

1a: Requester sendet Memory Read Packet MRd. Switch speichert MRd und überprüft CRC (LCRC)

1b: Switch quittiert mit ACK. Requester löscht Packet MRd aus seinem VC Buffer.

2a: Switch leitet Packet MRd auf grund der Memory-Adresse an den richtigen Port weiter. Completer erhält (speichert) MRd und überprüft CRC (LCRC).

2b: Completer quittiert mit ACK. Switch löscht Packet MRd aus seinem VC Buffer.

3a: Completer überprüft CRC (ECRC) und sendet ein Completion Packet CplD mit den Lesedaten. Switch speichert Packet CplD und überprüft CRC (LCRC).

3b: Switch quittiert mit ACK. Completer löscht Packet CplD aus seinem VC Buffer.

4a: Switch dekodiert Requester ID in CplD und leitet das Paket an den richtigen Port weiter. Requester erhält (speichert) CplD und überprüft CRC (LCRC).

4b: Requester quittiert mit ACK. Switch löscht Packet CplD aus seinem VC Buffer. Requester überprüft CRC (ECRC) in CplD. Requester vergleicht 8Bit-Tag des erhaltenen CplD Pakets mit dem 8Bit-Tag des ursprünglichen MRd Pakets.

### Traffic Class TC (0 bis 7):

Legt die Priorität eines Pakets beim Durchschalten durch das PCI-Netzwerk (PCI Switch, PCI Express Fabric) fest. TC7 ist die höchste Priorität. Ein hochprioriges Durchschalten (TC=7) ist besonders wichtig bei Interrupt-Messages und bei isochronen Daten (z.B. Daten von einer Filmkamera, die eine garantierte Bandbreite brauchen).

Normale Dateitransfers von/zum SCSI-Controller (Festplatte, DVD) besitzen in der Regel TC=0.

Die Traffic Class TC wird jedem Paket von der Anwendersoftware oder der Treibersoftware zugewiesen.

### Virtual Channel VC:

Die momentane Verbindung vom Datenpuffer des Senders (Requester) über die Datenpuffer des PCI-Netzwerks (PCI Switch, PCI Express Fabric) bis zum Datenpuffer des Empfängers (Completer) wird auch als Virtual Channel VC bezeichnet. Die im PCI Device und im PCI-Netzwerk enthaltenen Datenpuffer werden als **VC Buffer** bezeichnet. Es sind dies in einem PCI Device maximal 8 (VC Buffer0 bis VC Buffer7). Da im PCI-Netzwerk mehrere VC Buffer vorhanden sind können gleichzeitig mehrere Verbindungen (Virtual Channels VCs) mit unterschiedlicher Priorität geschaltet sein.

### **TC/VC-Mapping:**

Jedem VC Buffer ist ein Wert für eine Traffic Class TC zugeordnet. Datenpakete werden entsprechend ihrer TC# in VC Buffern mit derselben TC# zwischengespeichert (→ Virtual Channels mit unterschiedlichen Prioritäten).

### **Flow Control:**

Ein Paket mit Schreibdaten wird im VC-Datenpuffer des empfangenden PCI-Device abgespeichert. Der Empfänger versorgt den Sender periodisch mit Informationen, wie viel freie Pufferkapazität er zur Verfügung hat (Flow Control Mechanism Protocol). Somit ist sichergestellt, dass kein Transfer wegen zu kleiner Pufferkapazität wiederholt werden muss.

### **Switch mit Port:**

Über Multi-Port-Switches können PCI-Geräte direkt miteinander kommunizieren. Pro Port ist ein PCI-Device anschließbar, das maximal in 8 Funktions („Teilgeräte“) unterteilt sein kann. Eine Funktion muss function# = 0 besitzen. Ein Switch verwendet die prioritätsgesteuerte Arbitrierung.

**Ingress Port:** Port, der ein Paket empfängt.

**Egress Port:** Port, der ein Paket sendet.

**Hot Plug/ Unplug:** .....

### **Kommandos:**

Es sind prinzipiell die gleichen Memory-, I/O- und Configuration-Kommandos verfügbar wie beim PCI Conventional, siehe die dortige Tabelle. Zusätzlich gibt es Message-Request- und Completion-Kommandos. Completion-Kommandos steuern die Split Transactions, Message-Request-Kommandos übernehmen die Funktionen von (beim PCI Express fehlenden) Signalen wie Interrupt-, Fehler-, Buszuteilungs- oder Ready-Signal usw.

Die Kommando-Codes sind im Header eines Request Packets enthalten.

### 3.5. Aufgaben

#### **Aufgabe PCI-1: PCI-Bus, Varianten**

**Bitte nur kurze Antworten (Stichpunkte)**

Neben dem bisherigen „PCI Conventional“ gibt es 2 neuere PCI-Bus-Varianten. Nennen Sie die beiden Bezeichnungen. Vergleichen Sie beide mit dem PCI Conventional bei den wichtigen Merkmalen --Buskonzept(u.a. parallel/seriell), --Busbreite (Signaladern) und --Taktrate.

#### **Aufgabe PCI-2: PCI-Bus, Konfiguration**

1. Erklären Sie den Begriff "Bus-Bandbreite"! Welche Werte bietet der PCI-Bus?
2. Was ist ein "Target-Device" ?
3. *Beim Konfigurieren des PCI-Busses wird von der CPU über das Konfigurationsregister (Feld "Base Address") zuerst die Größe des Speicherbereichs eines PCI-Device ermittelt und dann die Beginnadresse festgelegt ("Mapping"). Beantworten Sie dazu folgende Fragen:*
  - a) Welche Speichergröße liegt vor unter den Annahmen:  
*Schreiben "Base Address" AD31..AD0 = FFFFFFFF*  
*Lesen "Base Address" AD31..AD0 = FFFF00X*
  - b) Welche Beginnadressen für den obigen Speicherbereich kann die CPU vergeben ?

#### **Aufgabe PCI-3: PCI-Bus, Konfiguration**

Gegeben ist ein PCI-Board mit 64 kByte Datenpuffer.

*Im 32Bit-Konfigurationsregister (Feld: Base Address) steht nach der Konfiguration: FC400004 hex.*

- a) Welche Bitpositionen (31 .. 00) sind als ROM-Zellen realisiert?
- b) Wie ermittelt der Host beim Zugriff auf das Konfigurationsregister (Feld: Base Address) die Größe des geforderten Speicherbereichs? (Antwort stichpunktartig).
- c) Welche Beginnadresse (Hex-Wert) wurde dem Speicherbereich zugewiesen?
- d) Was wäre die kleinstmögliche von 0 verschiedene Beginnadresse (Hex-Wert) bei obiger Konfiguration?

#### **Aufgabe PCI-4: PCI Express**

1. Nennen Sie 3 wichtige technische Vorteile des PCI Express Bus (gegenüber dem PCI Conventional):
2. Erklären Sie die Bezeichnung PCIe x2! Wie ist der Bus realisiert? Wie viele Signaladern sind vorhanden?

#### **Aufgabe PCI-5: PCI-Bus, Adressräume**

1. Was ermöglicht es dem Host bei der Konfiguration, die Steckplätze einzeln abzufragen? Sind dadurch mehr Bussignale notwendig?
2. Welche Adressräume gibt es am PCI-Bus und woran erkennt ein Target diese unterschiedlichen Adressräume bei einem Buszyklus?
3. Im Unterschied zu einem "normalen" Mikroprozessorzyklus gibt es beim PCI-Zyklus zwei Ready-Signale: TRDY# (Target-Ready) und IRDY# (Initiator-Ready)  
Erklären Sie die Funktion der Ready-Signale am Beispiel eines Lesesyklus. Wieso ist ein Initiator-Ready notwendig?
4. Was verbirgt sich hinter der Bezeichnung PCISIG?