Studienarbeit

Zum Thema:

PCI-to-PCI-Bridge mit sicherheitsrelevanten Eigenschaften

Christian Böhme

Technische Universität Dresden

Fakultät Informatik

Institut für Systemarchitektur

18. Juli 2005

PCI-to-PCI-Bridge mit sicherheitsrelevanten Eigenschaften

In heutigen Betriebssystemen zählen Geräte-Treiber zu den fehleranfälligsten Komponenten im System überhaupt. Eines der härtesten und ohne schaltungstechnische Unterstützung nur unter großem Laufzeitaufwand behebbaren Sicherheitsprobleme ist die Möglichkeit, mittels Busmaster-DMA unter Umgehung der virtuellen Speicherverwaltung direkt auf den Hauptspeicher zugreifen zu können.

Ausgehend von den Eigenschaften des PCI-Busses und den aus einer Diskussion abgeleiteten Anforderungen an die Sicherheit eines PCs mit dem PCI-Bus als dessen Systembus wird eine Schaltung in Form einer *Add-In Card* für den PCI-Bus vorgestellt, die die Möglichkeit bietet, Geräte nach *PCI Conventional Revision 2.3* in einem handelsüblichen x86-PC mit 32-Bit-PCI-Bus als Systembus mit nicht vertrauenswürdiger Software zu betreiben, wobei nur Zugriffe auf Bereiche innerhalb des Definitionsbereichs des PCI-Busses zugelassen werden, die vom Betriebssystem explizit dafür freigeschaltet worden sind.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Die überwiegende Mehrheit der Begriffe der digitalen Schaltungstechnik und speziell nahezu sämtliche Literatur, die sich mit den technischen Aspekten des PCI-Busses befaßt (allen voran natürlich die Standards), bedienen sich entweder der englischen Sprache in Lehnwörtern oder sind vollständig in ihr verfaßt. Zudem haben häufig auftauchende Begriffe im Kontext des PCI-Busses eine spezielle Bedeutung und somit den Charakter eines Eigennamens. Aus diesem Grunde wurden die entsprechenden Textstellen lediglich hervorgehoben und auf deren Übertragung ins Deutsche verzichtet, falls die Bedeutung eines Begriffes einerseits unmißverständlich aus dem Kontext hervorgeht und andererseits die sprachliche Verständlichkeit beibehalten wird. Zur eigentlichen Begriffsklärung sei jeweils auf den gleichnamigen Eintrag im Glossar hingewiesen.

Mein besonderer Dank gilt überdies meinem Betreuer Frank Mehnert, der die haarige Aufgabe der Betreuung eines Elektrotechnikers unter Informatikern mit meisterlichem Verständnis und ebenso viel Geduld auf sich nahm.

Kapitel 1. Motivation

Im Laufe der letzten Jahre hat sich auch in der Welt der PC-Nutzer das Bewußtsein durchgesetzt, daß stabil verfügbare, informationsverarbeitende Systeme einen nicht zu unterschätzenden Wert darstellen. Betrachtet man dazu, welchen Stellenwert der PC mittlerweile im alltäglichen Leben beispielsweise für so wichtige als auch vertrauliche Vorgänge wie die Verwaltung der privaten Konten über das Internet eingenommen hat, dann wird deutlich, daß Fragen der Sicherheit immer weiter in den Vordergrund rücken. Der Begriff *Sicherheit* innerhalb der Informationstechnik ist allerdings recht überladen, weshalb es in der Regel notwendig ist, ihn entsprechend des Kontextes, innerhalb dessen er benutzt wird, näher zu bestimmen.

Allgemein beschreibt der Begriff in Bezug auf Betriebssysteme in erster Linie ein Maß für die Systemstabilität und -integrität. Eine der wichtigsten Forderungen an ein informationsverarbeitendes System ist die, daß es möglichst nicht durch einen einzelnen Fehler an Stabilität oder sogar gänzlich seinen Gebrauchswert verliert. Beispiele dafür sind die allzu bekannten Software-Fehler, die von Prozessen im Nutzermodus verursacht, vom Betriebssystem detektiert und deren Wirkung schließlich nur auf den fehlerhaften Prozeß begrenzt wird. Da sich derartige Probleme lediglich mit softwareseitigen Fehlern innerhalb der virtuellen Speicherverwaltung offenbaren, sollen im Folgenden zunächst eine Diskussion die Eigenschaften des PCI-Busses in seiner Funktion als Systembus im PC und die Tragweite von möglichen Angriffen durch Treibersoftware sowie PCI-Hardware selbst näher beleuchten und schließlich Möglichkeiten für entsprechende Gegenmaßnahmen abgeleitet werden.

Identitätsprüfung

Die Adressierung von Teilnehmern am PCI-Bus wird über die Definition eines oder mehrerer nichtüberlappender Bereiche aus dem Adressraum des PCI-Busses während der PCI-Bus-Konfiguration durch die Systemsoftware vorgenommen, was einer surjektiven Abbildung von PCI-Bus-Adressen auf physisch präsente Geräte am PCI-Bus entspricht. Prinzipiell sieht der PCI-Standard keine Mittel für die eindeutige Ableitung der Identität eines Gerätes am PCI-Bus, beispielsweise aus einer PCI-Bus-Adresse über geeignete Verzeichnisdienste, vor. Um also die Identität eines Busteilnehmers zweifelsfrei feststellen zu können, bedarf es der Unterstützung durch das Bestriebssystem oder allgemein der Instanz, die die Adressraumverwaltung am PCI-Bus durchführt, da nur diese die entsprechende Kenntnis über die Abbildungsvorschrift besitzt. Der Angriffspunkt an diesem Zuordnungsmodell offenbart sich nun durch die implizite Adressierung beim Start einer Transaktion am PCI-Bus: Beabsichtigt ein *Master* einen Datentransfer, d.h. einen Zugriff auf einen Adressbereich innerhalb des Adressraumes des PCI-Busses, so kann er lediglich eine Startadresse bekanntgeben, mit der es ohne Kenntnis der o.g. Abbildungsvorschrift nicht möglich ist, die Identität des Zielteilnehmers, das physisch präsente Gerät, zu bestimmen. Zu diesem Zeitpunkt würde sich ein *Target*, in dessen Bereich die Adresse fällt, als der angesprochene Busteilnehmer melden und damit die Transaktion ermöglichen. Es ist nun leicht vorstellbar, daß ein nicht vertrauenswürdiges *Target* auf eine Transaktion, die eine Startadresse außerhalb seines von der Systemsoftware festgelegten Adressbereichs definiert, reagieren und somit eine Identität vortäuschen könnte, die es tatsächlich gar nicht besitzt.

Gleichsam hat ein *Target* keine Chance, die Identität des Initiators einer Transaktion über die zu Beginn der Transaktion bereitgestellten Informationen zu bestimmen. Es kann somit keine eindeutige Entscheidung darüber treffen, ob eine Transaktion mit einem potentiell nicht vertrauenswürdigen *Master* stattfindet. [1]

[1] Rein theoretisch gäbe es eine Möglichkeit, durch Sammeln von Daten und Bewerten von Zuständen sofort nach Beendigung der Resetphase über Heuristiken eine Idee zu erlangen, wer der Initiator einer Transaktion sein könnte. Der Aufwand wäre allerdings immens. Konfigurationstransaktionen sollen an dieser Stelle auf Grund der Trivialität der Frage bezüglich ihrer Herkunft nicht betrachtet werden.

Vertraulichkeit

In Umgebungen mit verschärften Anforderungen an die Geheimhaltung von Informationen wird man besonders darauf bedacht sein, jeden Zugriff auf datenübertragende Systeme zu begrenzen. Der galvanische Kontakt jedes Teilnehmers am PCI-Bus mit den Signalleitungen und die Forderung nach der Fähigkeit zur Detektion des Buszustandes durch jeden einzelnen Teilnehmer zu jedem Takt, um das „Verpassen“ einer Transaktion zu vermeiden und schlichtweg ansprechbar zu bleiben, erlauben andererseits die uneingeschränkte Möglichkeit, sämtlichen Verkehr, der über den Bus abgewickelt wird, von jedem Teilnehmer abzuhören.

Als Beispiel stelle man sich folgende Situation vor: Der Nutzer eines frisch erworbenen Netzwerkadapters auf einer *Add-In Card* unbekannter oder zweifehalfter Herkunft (was kaum als Problem angesehen wird; das Gerät samt Treiber waren schließlich ausgesprochen „preiswert“) verbaut in bestem Glauben ein Gerät und installiert die mitgelieferte Treibersoftware, deren tatsächliche Funktion ihm möglicherweise völlig unbekannt sind. Mit mäßigem Aufwand wäre es nun für die Hardware-Treiber-Combo möglich, Daten zu sammeln und bei Bedarf unbeobachtet „nach Hause zu telefonieren“. Da es sich um einen Netzwerkadapter handelt, würde niemand Verdacht schöpfen, wenn in begrenztem Umfang zusätzlicher Netzverkehr auf unterster Ebene im System erzeugt werden würde.

Ein Gerät, welches bereits sofort nach Beendigung der Resetphase voll betriebsbereit ist, könnte mit vergleichsweise geringem Aufwand über das Abhören der Transaktionen zur Konfiguration der übrigen angeschlossenen Teilnehmer ein vollständiges Bild von der Hierarchie von Bussen aufbauen, an deren Ursprung es angeschlossen ist. Dabei ist es von Vorteil, sich so nahe wie möglich am Ursprung der Hierarchie aller Busse im System zu befinden, um idealerweise alle Konfigurationstransaktionen abhören zu können. Mit den auf diese Weise gewonnenen Informationen wäre es nun möglich, ganz gezielt Angriffe auf Bereiche innerhalb des gesamten Adressraumes des PCI-Busses auszuführen, ohne daß man diese mit einem Eindringling assoziieren könnte. Angriffe dieser Art würden sich entweder nur auf Lesezugriffe beschränken oder sogar mit entsprechendem Wissen Schreibzugriffe ausführen, um durch darauffolgende Lesezugriffe die Preisgabe von noch mehr Informationen zu erzwingen.

Speicherschutz

Während Instrumente wie virtuelle Speicherverwaltung den softwareseitigen Schutz vor unberechtigten Zugriffen auf Speicherbereiche bieten, müssen für Problemstellungen, die sich prinzipiell dieser Kontrolle entziehen, andere Maßnahmen ergriffen werden. In Architekturen, in denen der PCI-Bus als Systembus betrieben und damit der Definitionsbereich des PCI-Busses direkt auf den Adressbereich des Hauptspeichers abgebildet wird, ist es denkbar, daß PCI-Treibersoftware, die direkt mit Hardwarekomponenten kommuniziert und daher potentiell den größten Schaden verursachen kann, entweder bewußt (als Eindringling) oder unbewußt (durch Software-Fehler) den Zustand eines Systems durch unberechtigte Speicherzugriffe in einem Umfang modifizieren kann, daß es unbrauchbar wird. Die Problematik des Speicherschutzes am PC leitet sich als Konsequenz direkt aus den unter„Identitätsprüfung“ und „Vertraulichkeit“ beschriebenen Angriffsmöglichkeiten ab.

Eine entsprechende Konstellation ließe sich durch folgendes Szenario beschreiben: Das Beispielsystem sei ein x86-basierter PC, in welchem ein standardkonformes PCI-Gerät, das *DMA*-fähig ist, mit einer Software betrieben wird, die fehlerhaft implementiert ist und Probleme bei der Berechnung von Hardwareadressen für Zeiger in den physikalischen Hauptspeicher aufweist. Der *DMA*-Automat des Geräts wird in der Folge mit Adressen für überlappende oder gänzlich außerhalb der ihm ursprünglich zur Verfügung gestellten Bereiche im physikalischen Speicher programmiert. Sobald das Gerät seine FIFOs leeren muß, wird sein *Master* beginnen, den Zugriff auf den PCI-Bus zu beantragen. Falls diesem Antrag stattgegeben wurde, beginnt der *Master* einen Schreibzugriff mit einer vom Treiber gelieferten Startadresse, die außerhalb der Bereiche aller anderen Geräte am PCI-Bus liegt, womit sich die *Host-Bus-Bridge* als Empfänger für diese Transaktion angesprochen fühlt. Wie alle PCI-Bus-Teilnehmer kann auch sie nicht ermitteln, von welchem *Master* diese Transaktion initiiert wurde und muß davon ausgehen, daß der Initiator vertrauenswürdig ist. Das Gerät leert nun seine FIFOs in einen Bereich des Hauptspeichers, der möglicherweise nicht dafür allokiert wurde und könnte damit den Speicherbereich eines anderen Prozesses ohne dessen Wissen überschreiben,

was im Falle von Letzterem zu einem nicht definierten Zustand führt. Im Extremfall könnte ein derartiger Fehler das gesamte System in den Abgrund ziehen. Ähnlich bedenklich im Hinblick auf die Wahrung der Vertraulichkeit, allerdings in ihrer unmittelbaren Wirkung auf die Stabilität weniger weitreichend, sind Lesezugriffe auf nicht dafür vorgesehene Bereiche im physikalischen Speicher, die Informationen preisgeben würden, welche bei korrekter Programmierung geschützt wären.

Probleme der EMV

Oft wird im Sinne der Qualitätsbeschreibung in der Informationstheorie über gestörte Kanäle von einer „sicheren“ Übertragung gesprochen, wenn damit eher ausgedrückt werden soll, mit welcher Wahrscheinlichkeit eine fehlerhafte Information ausgeschlossen oder mit welcher Stabilität eine geforderte Übertragungsqualität aufrecht erhalten werden kann. Gern unterschätzt, ist die elektromagnetische Verträglichkeit einer Baugruppe inhärent mit den schaltungstechnischen Details der beteiligten Komponenten verknüpft. Der PCI-Standard gibt für Systemhersteller und Produzenten von Komponenten und *Add-In Card*s entsprechende Regeln vor, deren Einhaltung für die stabile und nachvollziehbare Arbeitsweise des gesamten Busses Voraussetzung sind.

Durch die gleichzeitige Verwendung von 32 oder 64 Datenleitungen in seinen 32- oder 64-Bit-Inkarnationen werden hohe schaltungstechnische Anforderungen an die Störanfälligkeit und -unterdrückung, mit anderen Worten: Hochfrequenztauglichkeit, der Leitungen und Komponenten des PCI-Busses gestellt. Probleme durch Übersprechen auf benachbarte Leitungen machen sich gerade dann bemerkbar, wenn hohe Schaltströme kurzzeitig in die Leitungen fließen oder davon abgezogen werden. Die im Standard festgeschriebenen Werte zu garantieren liegt einerseits in der Verantwortung des Systemherstellers, andererseits auch bei Herstellern von *Add-In Card*s. Der PCI-Bus bietet nur minimale Möglichkeiten zur Detektion von Übertragungsfehlern mittels Paritätsberechnung über die Datenleitungen und schreibt keinerlei Einrichtung zur Fehlerbehebung vor. Tatsächlich wird diese Aufgabe der Software überlassen. Die Vermeidung von Störungen durch die gegenseitige Beeinflussung parallel verlaufender Leitungen oder durch unerwünschte Signalverzerrungen kann durch entsprechend sorgfältige schaltungstechnische Auslegung der passiven und mechanischen Bauteile nach hochfrequenten Gesichtspunkten vermieden werden. In jedem Fall offenbaren Störungen dieser Art schaltungstechnische Mängel unabhängig von deren Ursache.

Schutzmaßnahmen

Als Schlußfolgerung aus oben geführter Diskussion würde ein besorgter Nutzer folgende Gegenmaßnahmen ergreifen:

1. Aus „ Identitätsprüfung “ geht hervor, daß die eindeutige Überprüfung der Aktivitäten eines *Master*s am PCI-Bus nur durch den Einsatz einer Instanz erfolgen kann, die den kompletten Verkehr zwischen dem *Master* und den schutzwürdigen Teilen des PCI-Busses filtert und überwacht, um gegebenenfalls regulierend einzugreifen.

2. Nach „Vertraulichkeit“ und „Probleme der EMV“ sollte ein nicht vertrauenswürdiges Gerät idealerweise vom Systembus isoliert an einem eigens dafür geschaffenen Bus betrieben werden, um das Abhören fremder Transaktionen auszuschließen und die schaltungstechnische Beeinträchtigung auf den isolierten Bus zu verlagern.

3. Schließlich dürfen entsprechend „Speicherschutz“ einem *Master*-Automaten eines *DMA*-fähigen Gerätes nur Adressen innerhalb von Bereichen übergeben werden, die dafür allokiert wurden. Wenn dies nicht garantiert werden kann, muß eine Möglichkeit geschaffen werden, den Zugriff auf Adressbereiche direkt für jeden Datentransfer über den Bus zu überwachen.

Die Umsetzung dieser Punkte sollte sinnvollerweise ohne Beeinflussung der Funktion des PCI-Busses als Transportmedium, d.h., transparent bezüglich aller übrigen dort angeschalteten Geräte, erfolgen, um auch im Falle eines Angriffes jederzeit den fehlerfreien Betrieb des Busses zu gewährleisten und die Zerstörung der Geräte zu vermeiden.

Kapitel 2. Der PCI-Bus: Überblick und Einordnung

PCI steht für *Peripheral Component Interconnect* und beschreibt mittels einer Ansammlung von Standards eine Möglichkeit, mit verschiedenartigen Peripherie-Komponenten, also solchen, die nicht an der Steuerung eines Systems beteiligt sind, innerhalb jenes Systems Daten auszutauschen. PCI ist für sogenannte lokale Komponenten konzipiert worden, was deren Nähe zur CPU verdeutlichen soll und sich in seinen elektrischen Eigenschaften und Betriebsgrenzen niederschlägt. PCI wurde 1992 von Intel als Systembus in Intels x86-Architektur vorgestellt und lag zum Zeitpunkt seiner Einführung im Vergleich zu weiteren Technologien wie NuBus, SBUS oder GIO technisch auf der Höhe der Zeit. Gleiches kann in der heutigen Zeit allerdings nicht mehr behauptet werden, weshalb im Folgenden anhand einer Auflistung charakteristischer Eigenschaften eine Gegenüberstellung mit Entwicklungen aktueller Vernetzungssyteme wie *HyperTransport* [HYPERT], *InfiniBand* [INFINIB] und *PCI Express* [PCIEX] erfolgen soll:

- Die Signalisierung erfolgt durch *Reflected Wave Signaling*. Ein nicht unerheblicher Anteil am Taktzyklus (etwa 30 bis 40%) muß dadurch für die Stabilisierung des Signalpegels reserviert werden.

  Sowohl HyperTransport als auch InfiniBand und PCI Express gehen mittels *LVDS* einen radikal anderen Weg, der ein um Größenordnungen verbessertes Störverhalten (größere Störfestigkeit durch differentielle Eingangspuffer und geringere Störstahlung durch Umschalten von Stromrichtungen anstelle von Spannungspegeln) bei weitaus höheren Frequenzen bietet.

- Jedem Teilnehmer muß der uneingeschränkte galvanische Kontakt mit den gemeinsam genutzten Signalleitungen ermöglicht werden.

  Die Architekturen von HyperTransport, InfiniBand und PCI Express basieren auf einem Schichtenmodell, auf dessen physikalischer Ebene serielle, bidirektionale Punkt-zu-Punkt-Verbindungen die kommunizierenden Teilnehmer über entsprechende Schaltnetzwerke anbinden und erst oberhalb der physikalischen Ebene logisch „verbinden“.

- Alle Teilnehmer werden zentral mit einem Taktsignal versorgt, wofür eine separate, gemeinsam genutzte Leitung bereitsteht, was der maximalen Leitungslänge durch begrenzte Signallaufzeiten entsprechende Beschränkungen auferlegt.

  Im Vergleich dazu benutzen InfiniBand und PCI Express einen in den Datenstrom kodierten Takt, was Synchronisationsprobleme vermeidet, da der Takt hier Teil des Signals ist und damit keine Verschiebungen der gemeinsamen Zeitbasis auf Grund von unterschiedlicher Laufzeit von Takt und Daten auftreten kann.

- Adressierungs- und Nutzdaten werden im Zeitmultiplex über die selben Leitungen übertragen, wobei für Steuersignale zusätzliche Leitungen existieren.

  Ahnlich der Adressierung in den bekannten Netzwerkprotokollen werden bei HyperTransport, InfiniBand und PCI Express die Bestimmungsdaten auf Verbindungsebene in dafür vorgesehene Felder innerhalb von Paketen verpackt, bevor sie „auf die Leitung gegeben“ werden.

- Der PCI-Bus ist sowohl in der 32-Bit- als auch der 64-Bit-Version für jeweils 64-Bit- oder 32-Bit-Geräte des selben Signalisierungsstandards benutzbar.

  HyperTransport, InfiniBand als auch PCI Express erlauben die dynamische Anpassung von Taktfrequenz und Busbreite (HyperTransport) oder „Spurbreite“ (PCI Express).

- Die Bus-Teilnehmer können zur Zeit der Buskonfiguration (üblicherweise im Anschluß an den Bus-Reset) oder darüber hinaus entsprechend ihrer Eigenschaften und Fähigkeiten ohne Nutzereingriff auf die spezielle PCI-Bus-Umgebung eingestellt sowie deren Adressierung festgelegt werden („Plug & Play“).

Sowohl HyperTransport als auch PCI Express folgen dem Softwaremodell von PCI, womit sich zuminderst auf dieser Ebene keine nennenswerten Umstellungen notwendig machen sollten.

Topologie

PCI in seiner topologischen Struktur besteht aus mehreren parallel (hier: gleichzeitig) betriebenen Sammelleitungen, was in der Elektrotechnik als *Bus* bezeichnet wird. Üblicherweise spricht man deshalb einfach vom *PCI-Bus*. [1] Diese strukturell flache, lineare Anordnung kann mittels *PCI-to-PCI-Bridge*s zu baumähnlichen, hierarchisch geordneten Strukturen erweitert werden, womit gleichzeitig die durch die endlich kurze Signallaufzeit beschränkte maximale Teilnehmerzahl an einem PCI-Bus überwunden wird.

Buszugriffssteuerung

Das Zugriffsmodell am PCI-Bus leitet sich von einem statistischen Verkehrsmodell ab, nach dem der Zugriff auf den Bus, d.h., die Erlaubnis, Datentransfers zu starten, ausgehend von seinem initialen Zustand der völligen Ruhe erst von den Teilnehmern explizit „nach Bedarf“ angefordert werden muß. Da der Algorithmus für die Zuweisung der Zugriffsrechte nicht vorgeschrieben ist, können optimale Lösungen für die unterschiedlichsten Szenarien entsprechend der an einem Bus tatsächlich angeschalteten Teilnehmer von der Konfigurationssoftware programmiert werden. Die Einrichtung, welche schließlich die Zugriffsrechte zuweist, wird *Arbiter* genannt. Obgleich der PCI-Standard strenge Auflagen für die Latenz von *Agent*s am Bus vorgibt, ist der PCI-Bus prinzipiell nicht echtzeitfähig, da ihm dazu entscheidende Merkmale zur Resourcenverwaltung fehlen, welche beispielsweise erlauben würden, einen festen Teil eines zyklischen Zeitabschnitts fester Länge einem *Agent* zu reservieren, um dessen Echtzeitanforderungen zu genügen.

Adressierung von Teilnehmern

Am PCI-Bus werden drei Adressräume unterschieden: *Configuration Space*, *Memory Space* und *IO Space*. Ersterer dient dem Zugriff auf die Konfigurationsregister von physisch präsenten Geräten, wobei die Adressierung explizit über eine jedem Gerät separat zugeordnete Signalleitung erfolgt. Die Adressierung über Letztere erfolgt hingegen implizit, indem sich aus der Menge aller Busteilnehmer nach Bekanntgabe der Startadresse zu Beginn einer Transaktion bei korrektem Betrieb nur einer zum angesprochenen Ziel der Transaktion erklärt. Ein adressierbarer Teilnehmer am PCI-Bus ist gekennzeichnet durch mindestens einen nur ihm zugewiesenen, nichtüberlappenden Bereich innerhalb des gesamten Adressraumes. Über diese Adressbereiche ist es einem Teilnehmer möglich, während der Detektion einer Transaktion sich eindeutig als das Ziel zu identifizieren, indem er selbstständig testet, ob die angegebene Startadresse innerhalb eines der ihm zugewiesenen Bereiche liegt. Diese Adressbereiche festzulegen und die Eindeutigkeit der Abbildung zu gewährleisten ist Aufgabe des Systems, innerhalb dessen der PCI-Bus betrieben wird.

Datentransfers

Datentransfers am PCI-Bus sind transaktionsbasiert, d.h., Initiator und Ziel des Transfers sprechen Protokolle, die zu jedem Taktzeitpunkt die genaue Feststellung des Buszustandes und Diagnosen über die Bereitschaft der Transaktionspartner erlauben. Ein Teilnehmer, der beabsichtigt, Daten zu übertragen, beantragt zunächst den Zugriff auf den Bus und, falls genehmigt, startet eine Transaktion als eine definierte Folge von Bus-Zuständen, nach deren Ende der Bus wieder in einen definierten (Ruhe-)Zustand überführt wird. Jede Transaktion muß mit einer, kann aber auch mit zwei Adressphasen beginnen, in deren Verlauf der initiierende Teilnehmer (im Folgenden *Master*

[1] Die physische Anordnung von parallelen Leitungen, wie sie üblicherweise auf Mainboards zu finden ist, resultiert in aller Regel aus der Anpassung an physikalische Gesetze (nur parallel verlegte Streifenleiter gleicher Breite bieten, ungeachtet der Einflüsse aus der Umgebung, die gleiche Signallaufzeit) und dem technologischen Aufwand (gerade Leitungen sind am einfachsten und genauesten zu plotten).

genannt) eine gültige Adresse aus dem Definitionsbereich des entsprechenden Adressraumes und einen Befehl auf den Bus schickt, den alle anderen Teilnehmer (im Folgenden *Target*s genannt), die bereit für Datentransfers sind, nach derartigen Befehlen abhören. Stellt ein *Target* fest, daß die Adresse innerhalb seines Adressbereichs liegt und der Befehl ausführbar ist, teilt es dies durch ein Signal mit, welches es auf eine Steuerleitung schickt (die wiederum der *Master* abhört).

Auf dem PCI-Bus werden Datentransfers prinzipiell nach deren Richtung ausgehend vom *Master* unterschieden. Spricht man von einem Lesezugriff, treibt das *Target* die Datenleitungen, ist hingegen von einem Schreibzugriff die Rede, übernimmt der *Master* diese Aufgabe. Alle Transaktionen auf dem PCI-Bus lassen sich entweder auf Lese- oder Schreibzugriffe zurückführen. Während der Adressphase(n) treibt der *Master* unabhängig von der beabsichtigten Zugriffsart die Datenleitungen. Dies hat wiederum zur Folge, daß bei Lesezugriffen ein sogenannter *Turnaround Cycle* zwischen die letzte Adressphase und die erste Datenphase eingeschoben werden muß, um mit Sicherheit zu gewährleisten, daß nur ein Teilnehmer die Datenleitungen treibt. Andernfalls könnten die Treiber der Komponenten durch überhöhte Ströme zerstört werden.

Falls ein *Target* für eine Transaktion Akzeptanz signalisiert hat, beginnt der eigentliche Teil des Nutzdatentransfers. Dieser kann sich über eine oder mehrere Datenphasen erstrecken. Für jede dieser Datenphasen müssen *Master* und *Target* explizit bekunden, ob sie über entsprechende Resourcen für die Datenaufnahme (*Master*: Lesezugriff, *Target*: Schreibzugriff) oder Datenbereitstellung (*Master*: Schreibzugriff, *Target*: Lesezugriff) verfügen. Eine Datenphase ist beendet, wenn ein Datum übertragen und/oder mittels entsprechender Steuersignale die Transaktion von *Master* und/oder *Target* beendet wurde.

Kapitel 3. Entwurf

Sowohl die elektrische als auch die Spezifikationen der Protokolle für die Datenübertragung und Buszugriffssteuerung am PCI-Bus sind im Interesse der problemlosen Zusammenarbeit von Geräten unterschiedlicher Hersteller sehr streng gehalten und resultieren in mehreren Einschränkungen in Bezug auf die möglichen Lösungsansätze des zu entwickelnden Geräts.

Die Aufgabenstellung verlangt, daß ein zu überwachendes *Device* auf einer *Add-In Card* an jedem standard-konformen PCI-Bus zu betreiben sein soll. Letztere Forderung kann nur durch die Konstruktion einer *Add-In Card* erfüllt werden, da diese die einzige standardisierte Möglichkeit darstellt, Komponenten wahlfrei an den PCI-Bus zu schalten. Durch den Einsatz eines PCI-Sockels zur Aufnahme einer zu überwachenden *Add-In Card* wird schließlich auch der ersten Forderung entsprochen. Das zu entwerfende Gerät muß damit mechanisch und elektrisch den Spezifikationen für *Add-In Card*s als auch *System Board*s folgen.

Punkte 1 und 2 der unter Kapitel 1, *Motivation*, „Schutzmaßnahmen“ vorgeschlagenen Maßnahmen lassen sich in ihrer Realisierung zusammenfassen, weil sie sich bezüglich der schaltungstechnischen Anforderungen grundsätzlich nicht unterscheiden. Die elektrische Trennung von zwei PCI-Bussen und der damit verbundene schaltungstechnische Aufwand, den Datenverkehr in beiden Richtungen weiterzuleiten, erlaubt gleichzeitig die Filterung und Überwachung dieses Verkehrs. Für die Weiterleitung in beide Richtungen müssen die strengen Auflagen an die Latenz der beteiligten Teilnehmer am PCI-Bus eingehalten werden, die u.a. Maximalwerte für die von Geräten verursachbaren Verzögerungen während der Adress- und Datenphasen festlegen, um die Blockierung des PCI-Busses über einen längeren Zeitraum durch ein einzelnes Gerät zu unterbinden und den Bus zu jeder Zeit kurzfristig verfügbar zu halten. Die Existenz eines synchronen Protokolls für den Datentransfer und eines asynchronen Protokolls für die Buszugriffssteuerung schließen prinzipiell die Möglichkeit aus, an einem PCI-Bus auf Transaktionen zu reagieren und sie gleichzeitig über einen anderen weiterzuleiten. Als Konsequenz der in „Schutzmaßnahmen“ gestellten Forderungen disqualifizieren sich folglich alle Ansätze, die auf den vom PCI-Bus isolierten Betrieb des zu überwachenden Gerätes verzichten oder auf einer direkten Weiterleitung der Daten zum selben Takt ohne deren Zwischenspeicherung aufbauen. Ein Gerät, das im Namen eines anderen auf Transaktionen reagiert, um diese an das Gerät, in dessen Namen reagiert wurde, weiterzuleiten, muß also über geeignete Resourcen verfügen, um o.g. Protokolle voneinander entkoppeln zu können. Geräte am PCI-Bus, die diese Forderungen erfüllen, werden allgemein *Bridge*s genannt, wobei im vorliegenden Fall von einer *PCI-to-PCI Bridge* als einem Spezialfall davon gesprochen wird. Der PCI-Bus, in dem die *PCI-to-PCI Bridge* selbst installiert werden kann, stellt den *Primary Bus* dar, während die Rolle des *Secondary Bus*ses vom zu isolierenden PCI-Bus übernommen wird.

Die Überwachung des Zugriffs auf bestimmte Adressbereiche am *Secondary Bus*, wie sie Punkt 3 unter Kapitel 1, *Motivation*, „Schutzmaßnahmen“ fordert, wird über den gesamten Adressraum des 32-Bit-PCI-Busses von 4 GB mit einer Granularität von 4096 Byte vorgenommen, was der Größe einer Speicherseite der Intel-x86-Architektur entspricht. Die Zugriffsinformationen für den gesamten 32-Bit-Adressraum lassen sich damit komplett in einem Speicher von 1 MBit Größe ablegen. Da die Zugriffsüberwachung die Funktionalität einer normalen *PCI-to-PCI Bridge* erweitert, muß das Gerät am *Primary Bus* zusätzlich eine *Function* bereitstellen, über die es derart programmiert werden kann, daß es selektiv den Zugriff auf jeden Bereich innerhalb der 4 GB des 32-Bit-Adressraumes freischaltet oder sperrt. Die Programmierung kann sinnvollerweise nur von einem speziellen Treiber, der die Eigenschaften des Geräts kennt, übernommen werden. Dieser Treiber ist Teil des Betriebssystems und wird im *Mediator* untergebracht.

Das Programmiermodell der Schaltung entspricht dem einer standardkonformen *PCI-to-PCI Bridge* mit zusätzlich definierten Registern für die Steuerung der Überwachung des Zugriffs auf bestimmte Adressbereiche am *Secondary Bus*. Die Register und deren Einfluß auf den Betrieb des Geräts sind unter Anhang C, *Konfigurationsregister der PCI-to-PCI Bridge* und Anhang D, *Steuerregister der Monitorfunktion* in aller Ausführlichkeit beschrieben.

Schaltungsstruktur

Die eingangs geführten Betrachtungen machen deutlich, daß nur eine *PCI-to-PCI Bridge* mit der Fähigkeit, *Posted Write*s sowie möglichst mehrere, parallel aktive *Delayed Transaction*s bearbeiten zu können, die gestellte Aufgabe erfüllt. Im Einzelnen werden für die Einhaltung der PCI-Protokolle folgende Baugruppen benötigt:

- jeweils voneinander unabhängige Kontexte für *Master* und *Target* an beiden Bussen, zusätzlich ein Kontext für den *Arbiter*, der den Zugriff auf den *Secondary Bus* regelt
- Kontexte, die den Ablauf der *Delayed Transaction*s an den gegenüberliegenden Bussen getrennt voneinander steuern
- tiefe FIFOs, um den Datenverkehr von einem PCI-Bus zum anderen so weit als möglich zu entkoppeln

Das Blockschaltbild der Schaltung ist in Abbildung A.1, „ Blockschaltbild der *PCI-to-PCI Bridge* “ wiedergegeben.

Die unter Kapitel 1, *Motivation*, „Schutzmaßnahmen“ geforderte Transparenz, nach der für jeden Teilnehmer am Bus die gleiche standardisierte Schnittstelle bereitzustellen ist, resultiert in einer nahezu symmetrischen Struktur der gesamten Schaltung. An jedem der beiden PCI-Busse werden ohne Restriktionen alle im Standard definierten Datentransfer- und Buszugriffsprotokolle unterstützt. Um *Posted Write*s und *Delayed Transaction*s in jeder Richtung zu ermöglichen, werden auch die dafür notwendigen Baugruppen auf jeder Seite der *PCI-to-PCI Bridge* benötigt, so daß bis auf die unterschiedlichen *Target*-Kontexte alle Baugruppen doppelt instanziiert werden.

`PCI Bus Target` implementiert die Funktionen eines *Target*s am PCI-Bus. Es kapselt die Komplexität der Datentransferprotokolle vor den darauffolgenden Kontexten, an die es bei Schreibzugriffen die über den PCI-Bus übertragenen Daten gepuffert übergibt und von denen es bei Lesezugriffen Nutzdaten zum Transport über den PCI-Bus ungepuffert weiterleitet. Es nutzt ausschließlich die Einstellungen des *Command Register*s im PCI-Konfigurationsraum für seine Steuerung. Als *Target* am *Secondary Bus* werden zusätzlich noch die Bits im *Bridge Control Register* ausgewertet. Darüberhinaus übernimmt es in Verbindung mit `PCI Bus Data Frontend` die Paritätsfehlerkontrolle und den Transport von Daten, die vom PCI-Bus eintreffen oder dort erscheinen sollen.

`PCI Bus Master` implementiert die Funktionen eines *Master*s am PCI-Bus. Er kapselt die Komplexität der Buszugriffs- und Datentransferprotokolle vor dem ihm zugeteilten `Bridge Tail`, dem er bei Lesezugriffen die über den PCI-Bus übertragenen Nutzdaten gepuffert übergibt und von dem er bei Schreibzugriffen Daten zum Transport über den PCI-Bus ungepuffert weiterleitet. Er nutzt ausschließlich die Einstellungen des *Command Register*s im PCI-Konfigurationsraum für seine Steuerung. Als *Master* am *Secondary Bus* werden zusätzlich noch die Bits im *Bridge Control Register* ausgewertet. Darüberhinaus übernimmt er in Verbindung mit `PCI Bus Data Frontend` die Paritätsfehlerkontrolle und den Transport von Daten, die vom PCI-Bus eintreffen oder dort erscheinen sollen.

`PCI Bus Data Frontend` stellt die von `PCI Bus Master` und `PCI Bus Target` gemeinsam nutzbaren Resourcen wie Puffer für die vom PCI-Bus übernommenen Daten und Paritätsberechnung für hereinkommende sowie ausgehende Daten bereit.

`PCI Bus 2-Arbiter` implementiert einen einfachen Round-Robin-Disponenten für den *Secondary Bus*, bei dem keiner der beiden möglichen *Master* am Bus bevorzugt wird. Falls sich der Bus im Ruhezustand befindet, wird der Bus beim *Device* am *Secondary Bus* geparkt.

`Bridge Head` implementiert die Adressdekoderlogik, mittels derer entschieden wird, ob eine Transaktion am PCI-Bus für den gegenüberliegenden Bus bestimmt ist und stellt den Speicher für Status und Daten der gleichzeitig bearbeiteten *Delayed Transaction*s zur Verfügung. Er übernimmt die Weiterleitung der *Delayed Request*s und *Posted Write*s über den ihm zugeordneten `Request FIFO` und empfängt bei Lesezugriffen die Daten der *Delayed Completion*s aus den ihm

zugeordneten Response FIFO. Secondary Bus Bridge Head erweitert die Funktionalität des Bridge Heads für die Überwachung der Adresszugriffe für Transaktionen, die vom *Secondary Bus* zum *Primary Bus* fließen. Die Information für die Zugriffsrechte erhält er aus dem Speicher, auf dessen Inhalt er über das SRAM Interface zugreift. Diese Zugriffe sind reine Lesezugriffe.

Request FIFO bildet das eigentliche Herz der Schaltung, da er die wichtigste Resource stellt, die für die Verarbeitung von *Posted Write*s und *Delayed Transaction*s notwendig ist. Seine prinzipielle Fähigkeit zur Datenspeicherung und das gleichzeitige, aber nicht synchronisierte Schreiben am Eingang und Lesen am Ausgang erlauben die volle Entkopplung des Datenverkehrs von einem PCI-Bus auf den anderen. Response FIFO ist für die Speicherung der Daten von Lesezugriffen vorgesehen. Alle Lesezugriffe, welche die *PCI-to-PCI Bridge* überqueren, werden als *Delayed Transaction*s ausgeführt.

Bridge Tail ist der einzige selbstständig aktive Kontext der Schaltung und ist nur in Betrieb, solange entweder der Request FIFO oder seine Warteschlange mit unvollendeten *Delayed Request*s gefüllt sind und auf Abarbeitung warten. Er ist die Instanz, die einen *Delayed Request* nach erfolgreichem Datentransfer in eine *Delayed Completion* überführt.

Config Space implementiert die Logik für den PCI-Konfigationsraum. Alle Bausteine, die mit diesen Informationen arbeiten müssen, haben Lesezugriff auf die entsprechenden Register. PCI Bus Target und PCI Bus Master dürfen darüberhinaus noch Bits im *Status Register* schreiben.

Monitor Control übernimmt die Verwaltung der Zugriffsinformationen sowie die Steuerung des Zugriffsdetektors im Secondary Bus Bridge Head. Er hat Lese- und Schreibrechte auf den Speicher mit den Zugriffsrechten, welchen er über das SRAM Interface erreicht.

SRAM Interface kapselt und synchronisiert den gleichzeitigen Zugriff von Monitor Control und Secondary Bus Bridge Head auf den Speicher mit den Zugriffsrechten für die diskreten Adressbereiche.

Funktionsbeschreibung

Nach Beendigung des PCI-Resets befindet sich die Schaltung zunächst im rückgesetzten Zustand. Sie reagiert generell nur auf Transaktionen, die sich an den beiden PCI-Bussen ankündigen und den gegenüberliegenden Bus, im Falle des *Primary Bus*ses auch die eigenen Konfigurationsregister oder die Monitorfunktion, adressieren.

Transaktionen am *Primary Bus*

Secondary Bus

Prinzipiell kann es sich bei Transaktionen, für die der Adressdekoder vom Bridge Head festgestellt hat, daß sie für den *Secondary Bus* bestimmt sind, entweder um *Posted Write*s, also Schreibzugriffe im *Memory Space*, oder *Delayed Transaction*s, also Lesezugriffe im *Memory Space*, Schreib- und Lesezugriffe in *Configuration Space* und *IO Space*, handeln.

*Posted Write*s werden prinzipiell akzeptiert, sobald Kapazität für mindestens ein Datum im *downstream* Request FIFO vorhanden ist, andernfalls wird die Transaktion mit *Retry* beendet. In der Folge wird nun zu jedem Taktzyklus ein Datum vom PCI-Bus gelesen und in den *downstream* Request FIFO geschrieben, solange der *Master* Daten liefert und der Request FIFO seine Kapazität noch nicht erreicht hat. Am anderen Ende stellt der Bridge Tail fest, daß neue Daten vorliegen und bereitet den PCI Bus Master mit den Adress- und Steuerdaten auf den Start einer neuen Transaktion vor. Der PCI Bus Master meldet nun Bedarf am Zugriff auf den *Secondary Bus* an und, falls stattgegeben, beginnt den Schreibzugriff, in dessen Folge er vom Bridge Tail beständig mit Daten versorgt wird, solange der Request FIFO Daten enthält.

Der Aufwand für die Bearbeitung von *Delayed Transaction*s ist vergleichsweise höher. Ein *Delayed Request* wird nur erzeugt, falls das Maximum der gleichzeitig aktiven *Delayed Transaction*s noch nicht erreicht wurde. In dem Falle werden alle Merkmale der Transaktion gesichert und der *Delayed*

Request in den *downstream* `Request FIFO` geschrieben. Der `Bridge Tail` am anderen Ende erkennt, daß es sich um einen *Delayed Request* handelt und bereitet den `PCI Bus Master` mit den Adress- und Steuerdaten auf den Start einer neuen Transaktion vor. Der `PCI Bus Master` meldet nun seinen Bedarf am Zugriff auf den *Secondary Bus* an und, falls stattgegeben, beginnt die Transaktion. Falls es sich um einen Lesezugriff handelt, liefert der `PCI Bus Master` dem `Bridge Tail` die Daten, welche dieser in die für diese Transaktion reservierte Spur im `Response FIFO` schreibt. Nach beendeter Transaktion wird der *Delayed Request* in eine *Delayed Completion* umgewandelt und vom `Bridge Tail` in den *upstream* `Request FIFO` geschrieben. Der `Bridge Tail` am *Primary Bus* entnimmt diese und ändert ihren Status erneut, indem er markiert, daß sie nun abholbereit ist. Wenn der `Bridge Head` im Anschluß daran eine Transaktion mit den selben Merkmalen „sieht“ und feststellt, daß diese bereits existiert und noch dazu abholbereit ist, „spielt“ er diese Transaktion nach, indem er sie exakt mit dem Ergebnis der *Delayed Completion* endet, welches der `PCI Bus Master` am *Secondary Bus* lieferte.

Configuration Space

Transaktionen, die den *Configuration Space* adressieren, werden ausschließlich am *Primary Bus* akzeptiert. Anfragen dergleichen am *Secondary Bus* werden ignoriert. Da sämtliche Informationen in Registern abgelegt sind, kann auf alle Transaktionen ohne Verzögerung direkt reagiert werden.

Monitorfunktion

Die Monitorfunktion legt ihre Steuerdaten ebenfalls in Registern ab, muß allerdings bei Zugriffen, welche Daten aus dem externen Speicher benötigen, mit Verzögerungen rechnen, da auf diesen gleichzeitig vom `Secondary Bus Bridge Head` aus zugegriffen werden kann und konkurrierende Zugriffe auf den Speicher serialisiert werden müssen.

Transaktionen am *Secondary Bus*

Die Abläufe der Transaktionen am *Secondary Bus* unterscheiden sich prinzipiell nicht von denen am *Primary Bus*. Der einzige Unterschied besteht in der zusätzlichen Logik im `Secondary Bus Bridge Head`, die die Detektorfunktion für den Adressdekoder und einen Adressinkrementierer für Transaktionen mit mehreren Datenphasen implementiert. Damit ist es möglich, Zugriffe, die zunächst auf freigeschaltete Adressbereiche zugreifen, in deren weiteren Verlauf zu überprüfen, da sie durchaus über den freigeschalteten Bereich hinaus erfolgen können.

Kapitel 4. Implementierung

Die in Abbildung B.1, „Schaltbild der *Add-In Card*“ wiedergegebene Schaltung stellt eine streng nach *PCI Conventional Revision 2.3* [PCI23], implementierte *Universal Add-In Card* dar, die damit an PCI-Bussen mit 5-V- oder 3,3-V-Signalisierung betrieben werden kann. Der Signalisierungsstandard am *Secondary Bus* wurde auf 5V/33MHz festgelegt, da mittlerweile die meisten heutzutage verfügbaren *Add-In Card*s für die 5-V- und 3,3-V-Signalisierung ausgelegt sind. Die maximale Verlustleistung mußte allerdings auf etwa 17,5 W begrenzt werden, um für die Schaltung selbst eher theoretische 7,5 W zu reservieren. Die gesamte Logik für die *PCI-to-PCI Bridge*, inklusive der standardkonformen Puffer und Treiber, sind in einem *FPGA* untergebracht, womit bezüglich der Hardware genügend Flexibilität geboten wird, um den Anforderungen der zahlreichen PCI-Umgebungen gerecht werden zu können.

Schaltungskonzept

Durch den Prototyp-Charakter der Aufgabenstellung fiel die Wahl nach der schaltungstechnischen Realisierung auf einen *FPGA* der Firma Xilinx. Dieser Schaltkreis bietet derzeit noch die einzige Möglichkeit, ein *Component* nach dem mittlerweile eher betagten 5V/33MHz-Signalisierungsstandard in beliebig konfigurierbarer Logik zu implementieren. Zusätzlich werden die 3,3V/33MHz- und 3,3V/66MHz-Standards unterstützt, was, wie im Folgenden gezeigt wird, bei entsprechender Auslegung den Einsatz der Schaltung in allen drei Welten ermöglicht. Zudem verfügt der *FPGA* über eingebaute *Dual-Ported RAM*-Strukturen, die die Implementierung einer *PCI-to-PCI Bridge* mit der Forderung nach tiefen FIFOs erheblich vereinfachen und den Verzicht auf externe FIFO-Bausteine überhaupt erst ermöglichen. Die Logik zur Steuerung des Konfigurationsvorganges wird in einem *CPLD* des selben Herstellers untergebracht. Die Bitströme für die Konfiguration des *FPGA*s werden in einem EEPROM abgelegt, um in der Lage zu sein, die Logik der Schaltung auch in Zukunft modifizieren zu können. Da die Zugriffsinformationen für die PCI-Adressbereiche einen beträchtlichen Umfang erreichen können, bei jedem System-Reset jedoch ihre Gültigkeit verlieren, werden sie in einem externen 128kx8-SRAM abgelegt. Die in Anhang D, *Steuerregister der Monitorfunktion*, „`page_block`“ und „`page_access_status`“ beschriebene Verwaltung in Blöcken von acht aufeinanderfolgenden Adressbereichen hat ihre Ursache in der Wortbreite des eingesetzten SRAMs von 8 Bit.

Eine für *PCI-to-PCI Bridge*s wesentliche Forderung, die sich aus dem PCI-Standard ableitet, ist jene, daß bereits 5 Taktzyklen nach Beendigung der Resetphase (150 ns bei einer Taktfrequenz von 33 MHz) die ersten Transaktionen über den PCI-Bus transportierbar sein müssen. Die Konfiguration des *FPGA*s mit der schnellsten verfügbaren Methode hätte andererseits aber nicht annäherend innerhalb dieser Zeitspanne erfolgen können. Konsequenterweise war es also nicht möglich, den PCI-Reset für die Schaltungsinitialisierung zu nutzen. In diesem Fall mußte der Reset privat erzeugt werden, wobei nun wiederum auf eine andere Regel im PCI-Standard zurückgegriffen werden konnte, die vorschreibt, daß die Resetphase nach frühestens 100 ms beendet werden darf, nachdem alle Betriebsspannungen ihren vorgeschriebenen Pegel stabil erreicht haben, wobei die Reihenfolge in der Verfügbarkeit der einzelnen Betriebsspannungen nicht definiert ist. Entsprechend dieser Anforderungen wurde eine Schaltung entwickelt, die zunächst überwacht, daß alle benötigten Betriebsspannungen über eine bestimmte Zeit stabil und synchron anliegen, um danach die Konfigurationssequenz für den *FPGA* zu starten und zu überwachen, damit dieser noch vor Beendigung des PCI-Resets in den betriebsbereiten Zustand versetzt wird.

In hochfrequenten Digitalschaltungen mit Impulsflanken von bis zu 4 V/ns, wie sie vom PCI-Standard vorgeschrieben werden, sind die elektrodynamischen Eigenschaften der Bauteile und insbesondere deren geometrischer Aufbau sowie das Layout der Platine für einen nachvollziehbaren Betrieb der Gesamtschaltung nicht mehr zu vernachlässigen. Der PCI-Standard schreibt unter anderem explizit vor, welche Länge die Leiterzüge der Schaltungskomponenten haben müssen, die PCI-Signale transportieren, und gibt deutliche Hinweise für die hochfrequente Entkopplung der Betriebsspannungen. Gleichsam geben die Hersteller von digitalen Hochgeschwindigkeitskomponenten ähnliche Empfehlungen für den stabilen Betrieb ihrer Produkte an. All diesen Punkten wurde nach Möglichkeit entsprochen.

Erzeugung der Betriebsspannungen

Jeder *Add-In Card* stehen laut Standard folgende, auf ±5% Abweichung stabilisierte Betriebsspannungen zur Verfügung:

1. 5V/5A
2. 3,3V/7,6A
3. +12V/500mA
4. -12V/100mA

Damit können alle externen Logikbausteine als auch die PCI-Puffer respektive -Treiber im *FPGA* nach entsprechender Entkopplung direkt an der 3,3-V-Versorgung betrieben werden. Die Entkopplung der 3,3-V-Versorgung mit `C801` und `C802` liefert als Resultat V_{CC}.

Die Betriebsspannung V_{CCINT} für die interne Logik des *FPGAs* von 2,5 V wird hingegen von der 5-V-Versorgung abgeleitet, um entsprechende Regelreserve bei der zu erwartenden Strombelastung zu bieten. Die Bereitstellung der 2,5 V wird durch `U702` [DS101267] in Verbindung mit den Koppelkondensatoren `C701` und `C702` übernommen, deren Werte sich streng nach den Vorgaben des Datenblatts von `U702` richten und sorgfältig ausgewählt wurden [TAJ]. Die Entscheidung für einen kontinuierlichen Regler im Vergleich zu einem Schaltregler erklärt sich einerseits aus der einfacheren Beschaltung, andererseits aus der Forderung nach definierter und vor allem begrenzter Einschaltverzögerung, die ein Schaltregler durch die notwendige Einschwingphase, die wiederum von schwer zu beeinflussenden Bauteiletoleranzen abhängt, nicht bieten kann.

Erzeugung des privaten Resets

Der private Reset wird zentral durch `U901` [SLVS087L] erzeugt. Da die beiden Betriebsspannungen V_{CC} und V_{CCINT} aus getrennten und unabhängingen Quellen erzeugt werden, muß zunächst geprüft werden, daß beide Versorgungen zur gleichen Zeit stabil anliegen. Erst danach kann der eigentliche Resetvorgang gestartet werden.

Die 5-V-Versorgung wird durch `U701` [SUPERVSR] in Verbindung mit `R701` überwacht, welche den Steuereingang $\overline{SD}$ von `U702` auf Nullpotential halten, falls die 5-V-Versorgung einen Schwellwert von 4,5 V unterschreitet. Nullpotential an $\overline{SD}$ von `U702` bewirkt, daß der Strom an *Vout* von `U702` zum Erliegen kommt.

Ähnlich wird von `U901` bezüglich der 3,3-V-Versorgung verfahren: Solange die Eingangsspannung an *SENSE* 2,93 V unterschreitet oder $\overline{RESIN}$ auf Nullpotential liegt, wird das Nullpotential an $\overline{RESET}$ von `U901` unabhängig von einem vorher gestarteten Reset aufrecht erhalten. Der Pegel an $\overline{RESIN}$ wird durch die Kombination von `U902` und `U903` [SCES295N] mit `R901` und `R902` definiert. `U902` stellt dabei eine Besonderheit dar, da er an V_{CCINT} betrieben wird, sein Ausgang allerdings an V_{CC} liegt. Ausgenutzt wird dabei seine Eigenschaft, daß sein Ausgang bei ausgeschaltetem V_{CCINT} (was darauf schließen läßt, daß die 5-V-Versorgung nicht aktiv ist) sperrt und kein Strom in den ausgeschalteten Schaltkreis abfließt. In Verbindung mit `R901` liegt damit stets ein stabiler Pegel am Eingang von `U903`. Durch diese Maßnahme wird der definierte Übergang von einer Versorgungsumgebung in die andere gewährleistet.

Nachdem $\overline{RESIN}$ und *SENSE* von `U901` beide volles Potential führen, wird der eigentliche Resetvorgang gestartet, der durch `C902` und `C903` auf etwa 5 ms festgelegt wird. Dieser Puls ist lang genug, um die beteiligte Logik sicher in ihren Ausgangszustand zurückzuversetzen.

Konfiguration des *FPGA*s

Der Konfigurationsvorgang von `U101` [DS001] wird von der in `U1301` [DS054, DS057] implementierten Logik gesteuert und folgt dem Ansatz in [XAPP178]. Den Bitstrom, mit dem der *FPGA* konfiguriert wird, speichert `U301` [AMD21354]. Da der PCI-Takt für die Zeitdauer des PCI-Resets als nicht definiert angesehen werden muß, wird zusätzlich ein eigener Takt benötigt, der

durch G1401 [FOX] erzeugt wird. Daneben wird über U1501 [SLCS136L] in Verbindung mit R1502, R1503, R1504 und R1505 entschieden, ob die Signalisierung am *Primary Bus* mit 5 V oder 3,3 V erfolgt. Pin B49 des PCI-Connectors liefert zusätzlich die Information, ob der PCI-Bus mit einem Takt von 66 MHz betrieben wird.

Mit dem Ende des privaten Resets in Form einer LH-Flanke an *RESET* von U901 wird einerseits der Taktgenerator G1401 gestartet und andererseits U1301 aus seinem Reset erwachen. Zunächst wartet U1301 darauf, daß G1401 einen stabilen Takt liefert. Wenn dies der Fall ist, und U1301 festgestellt hat, daß weder J201 gesetzt ist noch *INIT* von U101 auf Nullpotential gezogen wurde, startet U1301 die Konfigurationssequenz von U101, nachdem es aus der Kombination von *pci5V_detect_pin* und *pci66MHz_detect_pin* die Startadresse für den entsprechenden Bitstrom generiert und seinen Adresszähler startet. Dabei zieht U1301 seine Ausgänge *fpga_write_disable_pin*, *fpga_data_invalid_pin* und *flash__output_enable_pin* auf Nullpotential. Ab diesem Zeitpunkt wird bei jeder LH-Flanke des Taktes ein Wort von acht Bit Breite von U101 aus *DQ0-DQ7* von U301 gelesen. Sobald der Konfigurationsautomat in U101 die Aufnahme des kompletten und fehlerfreien Bitstroms mittels *DONE* meldet, werden *fpga_write_disable_pin* und *fpga_data_invalid_pin* hochohmig sowie *flash__output_enable_pin* auf volles Potential gezogen und U1301 beginnt die abschließende Startup-Sequenz über 16 Zyklen für U101 zu takten. Gleichzeitig mit der LH-Flanke an *OE#* von U301 werden dessen Ausgänge hochohmig geschaltet, da die selben Leitungen von U401 [ALLIANCE] im eigentlichen Betrieb von U101 getrieben werden. Nach Ablauf der Startup-Sequenz zieht U1301 seinen *fpga_output_tristate_pin* auf Nullpotential, was alle Ausgangstreiber von U101 freischaltet und schließlich deren Pegel definiert. Damit ist der Konfigurationsvorgang abgeschlossen und U101 voll betriebsbereit.

Eine komplette Bootstrap-Sequenz beginnend mit dem stabilen Anliegen der 5-V- und 3,3-V-Versorgungsspannungen bis zum Freischalten der Ausgänge von U101 dauert etwa 30 ms, womit gewährleistet ist, daß U101 noch während des ersten PCI-Resets nach Systemstart in Betriebsbereitschaft versetzt wird. Alle weiteren PCI-Resets, deren Ursache nicht die Abschaltung der Versorgungsspannungen ist, erfordern keine Rekonfiguration von U101.

Für die Zeit des PCI-Resets, in der U101 noch nicht betriebsbereit ist, sind auch dessen Ausgänge hochohmig. Das hat zur Folge, daß ein am *Secondary Bus* befindliches Gerät den PCI-Reset erst „sehen“ würde, nachdem U101 konfiguriert wäre. Um dies zu vermeiden, wird mit R102 an *sec_bus__rst_pin* von U101 ein Widerstand nach Masse geschaltet, der den Pegel bei Hochohmigkeit auf Nullpotential zieht, was dem Reset-Pegel entspricht. Eine ähnliche Aufgabe übernimmt R101, der für die Zeit der global hochohmigen Ausgänge von U101 *CE#* von U301 auf Masse zieht, was letzteren aus dem Standby-Modus in den aktiven Modus befördert. Letzterer wird lediglich für die Zeit eingenommen, während der U101 konfiguriert wird, was wiederum der Bestimmung von U301 entspricht.

JTAG-Schnittstelle

In erster Linie für die Programmierung von U1301 und U301 als auch allgemeine Testzwecke wurde eine *JTAG*-Schnittstelle vorgesehen, die entsprechende Flexibilität in der Entwicklung auf Hardware-Ebene bietet und den in Anhang E, *JTAG-Adapter für den IBM-PC-Parallelport* beschriebenen Adapter für den Betrieb an der parallelen Schnittstelle eines IBM-PCs benutzt. Sie bietet den Zugriff auf das *Boundary Scan*-Register, welches durch die Verkettung von U1301 mit U101 gebildet wird.

Die Schnittstelle erlaubt es, bei vorhandenen 5-V- und 3,3-V-Versorgungen diese Spannungen über die Schnittstelle weiterzugeben, womit beispielsweise Testgeräte versorgt werden können. Im Gegenzug kann die Schaltung ihrerseits über die 5-V- und 3,3-V-Pins entsprechend von Testgeräten versorgt werden, was äußerst vorteilhaft ist, da sie nicht in einem beschalteten PCI-Sockel stecken muß, um getestet werden zu können. Die Pegel aller Signale über die Schnittstelle sind auf 5 V festgelegt, womit eine Pegelwandlung nötig wird, da die gesamte Logik auf der *Add-In Card* in 3,3-V-Technik ausgeführt ist. Zu diesem Thema ist reichlich Literatur verfaßt worden [SCEA035A], die eine Vielzahl an Lösungsvorschlägen beschreibt. Entsprechend der Verfügbarkeit wurden schließlich U1001 [DS012411] und U1002-U1005 [SCES296N] ausgewählt. Alle Eingangspegel werden durch Widerstände bei Nichtbenutzung der Schnittstelle auf definiertes Potential gelegt, um Oszillation der Eingangsstufen der Puffer zu vermeiden. Die maximale Taktfrequenz wird im Wesentlichen durch R1004, R1007, R1009 in Verbindung mit den Eingangskapazitäten der Folgestufen begrenzt und liegt bei etwa 2 MHz.

Die Zusatzsignale *AUX1* und *AUX2* werden direkt von U1301 verarbeitet und können damit beispielsweise für die Fernsteuerung des Konfigurationsvorgangs von U101 verwendet werden. Bei Setzen des Brückensteckers J901 erlaubt $\overline{RESET}$ die Fernsteuerung des eingangs beschriebenen privaten Resets.

Programmierung von *CPLD* und EEPROM

Sowohl U1301 als auch U301 erhalten ihre Programmierdaten über die in „*JTAG*-Schnittstelle“ beschriebene *JTAG*-Schnittstelle. U1301 wird direkt mit der *ISP*-Software von Xilinx programmiert, während die im Laufe der Entwicklung der Schaltung entstandene *JTAG*-Software das Schreiben von U301 mittels einer Folge von EXTEST- und SAMPLE/PRELOAD-Befehlen über die Pins von U1301 und U101 übernimmt.

Platine

Auf Grund der Anforderungen an die Signalstabilität im Hochfrequenzbereich und die Vielzahl der unterschiedlichen Betriebsspannungen wurde die Platine der Schaltung in vier Lagen ausgeführt.

Die Bestückungsseite trägt alle aktiven Bauteile und wird hauptsächlich für die Leitungen der PCI-Signale verwendet, um kürzeste und vor allem nachvollziehbare Leitungsstrukturen zu implementieren, da der PCI-Standard explizite Angaben zu Wellenwiderstand und Länge der Signalleitungen macht. Zudem wurde unter dem *FPGA* noch eine Massefläche gefüllt, die einen induktivitätsarmen Weg für die Ladungsträger vom Schaltkreis zur Spannungsversorgung darstellt, damit unerwünschte Effekte wie *ground bounce* von Vornherein vermieden werden.

Auf die erste Innenlage wurden die Betriebsspannungen in einer Weise aufgeteilt, nach der mittig unter den Leitungen für die PCI-Signale eine große Fläche für die 3,3-V-Versorgung zu liegen kommt, die ihrerseits auf der Höhe des *FPGA*s eine Insel für die 2,5-V-Versorgung bietet. An den Seitenrändern verlaufen die Flächen für die 5-V- und ±12-V-Versorgungen. Alle Flächen werden über zahlreiche Durchkontaktierungen von den Pins des PCI-Connectors auf der Bestückungs- und Rückseite gespeist. Gleiches gilt für die zweite Innenlage, die komplett mit Masse gefüllt wurde. Die Versorgung der Bauelemente auf den Außenlagen wird wiederum durch örtliche Durchkontaktierungen an den Bauelementen realisiert, um parasitäre Induktivitäten so gering wie möglich zu halten.

Die Rückseite wird hauptsächlich für Signale zwischen den Halbleitern verwendet und kann durch Beschränkungen in der Bauteilehöhe, die der PCI-Standard vorgibt, lediglich kleine Bauteile wie Widerstände, Kondensatoren und Single-Gate-Logik tragen.

Logik

Die Logik der implementierten *PCI-to-PCI Bridge* ist modular entworfen und vollständig in *Verilog* formuliert worden. Damit kann sie einerseits mit Simulationswerkzeugen getestet werden, die über ein Verilog-Frontend verfügen und andererseits problemlos mit den von Xilinx gelieferten Werkzeugen in einen Bitstrom für den *FPGA* kompiliert werden.

Die Module konnten direkt nach den Bausteinen in Abbildung A.1, „ Blockschaltbild der *PCI-to-PCI Bridge* “ implementiert werden. Dabei wurde auf sparsamen Einsatz von Registern bzw. Gattern und hohe Lokalität Wert gelegt, da dies dazu führt, daß die generierte Logik „beweglich“ bleibt, also innerhalb des *FPGA*s ihre Lage bei zukünftiger Modifizierung ändern kann, gleichzeitig aber mit der urspünglichen Pin-Belegung benutzbar bleibt. Desweiteren wurde auf einen streng synchronen Entwurf mit einer einzigen Zeitbasis geachtet, was Probleme durch nicht kalkulierbare Signallaufzeiten und Schaltspitzen durch verformte Impulse ausschließen soll. Der globale Takt wird direkt vom Taktsignal des *Primary Bus*ses abgeleitet, wobei er nach entsprechender Konfektionierung keiner weiteren Teilung unterliegt. Die Arbeitsgeschwindigkeit der Schaltung variiert demnach direkt mit der Änderung der Taktrate am *Primary Bus*. Die Hinweise in [XAPP027] flossen direkt in die grundlegende Konzeption der Schaltungsstruktur ein. Die Implementierung der FIFOs und Adressinkrementierer orientieren sich zudem an den Anregungen in [XAPP175, XAPP173, XAPP052, XAPP014].

Kapitel 5. Auswertung

Die vorgelegte Implementierung einer *PCI-to-PCI Bridge* ist in einem einzigen hochintegrierten, konfigurierbaren Logik-Schaltkreis untergebracht, wodurch sich bei entsprechend sorgfältiger Auslegung einer konsequent synchronen Schaltung und Befolgen der Herstellerangaben prinzipiell unerwünschte Erscheinungen wie asynchrones Schalten durch unerwartete Signallaufzeiten oder nicht nachvollziehbare Registerzustände durch Störimpulse (sog. *glitch*es) vermeiden lassen. Der wichtigste Anteil an der Entwicklung mit konfigurierbarer Logik ist die optimale Anpassung der zunächst formal aufgestellten Verhaltensbeschreibung an die Randbedingungen, wie sie beispielsweise durch die Architektur der verwendeten Bausteine selbst als auch das Layout der Platine vorgegeben werden. Sind diese Bedingungen eingehalten, dann kann davon ausgegangen werden, daß die während des Entwicklungsprozesses gewonnenen Simulationsergebnisse durch die implementierte Schaltung tatsächlich reproduziert werden können.

Der Automat des `PCI Bus Target`s ist in der Lage, bei beständiger Verfügbarkeit von Speicherplatz am Eingang des `Request FIFO`s (bei Schreibzugriffen) oder an seinem Datenausgang am PCI-Bus (bei Lesezugriffen) zu jedem Taktzyklus Daten zu übertragen. Gleichsam kann der Automat des `PCI Bus Master`s bei beständiger Verfügbarkeit von Daten am Ausgang des `Request FIFO`s (bei Schreibzugriffen) oder an seinem Dateneingang am PCI-Bus (bei Lesezugriffen) zu jedem Taktzyklus Daten übertragen.

Die Automaten für die Kontexte hinter den `PCI Bus Target`s reagieren mit einer Latenz von einem Taktzyklus auf die Adressphase, weil erst zu diesem Zeitpunkt die um einen Takt verzögerte Paritätsinformation für die Adresse vorliegt und damit eindeutig festgestellt werden kann, ob die detektierte Adresse gültig oder ein Bitfehler aufgetreten ist. In letzterem Fall ignorieren die Kontexte die Transaktion. Datentransfers können unter der Voraussetzung eines ungefüllten `Request FIFO`s ohne Latenz mit maximaler Datenrate, die durch den Bustakt vorgegeben ist, durchgeführt werden.

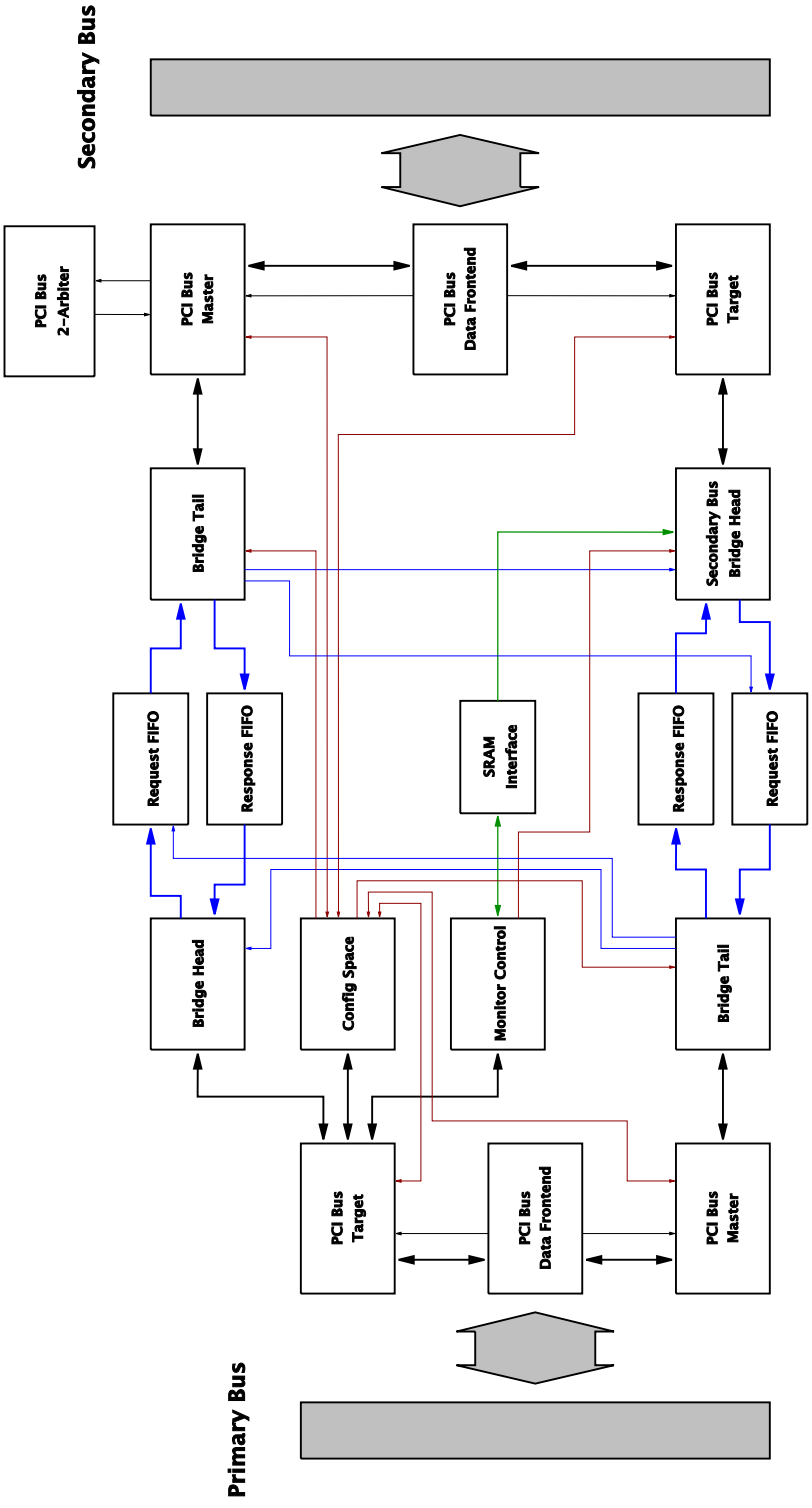
Die durch die Prüfung der Zugriffsrechte vom `Secondary Bus Bridge Head` verursachten Latenzen können während eines Datentransfers über 2048 Byte (was 512 Taktzyklen entspricht und den `Request FIFO` komplett füllen würde) maximal drei Taktzyklen betragen, wobei dieser Fall nur eintritt, wenn die Grenze zu einem Block von acht aufeinanderfolgenden 4096-Byte-Adressbereichen überschritten wird und die Zugriffsrechte für die in diesem Block organisierten Bereiche unbekannt oder aktualisiert worden sind. Bei geschickter Speicherbereichzuteilung, die sich am gewählten Verwaltungsmodell für die Zugriffsrechte orientieren sollte, können diese Latenzen durch Verhindern besagter Grenzüberschreitung vermieden werden.

Kapitel 6. Zusammenfassung

Die im Verlauf der Arbeit entstandene Schaltung bietet durch die Realisierung als *Add-In Card* die Möglichkeit, an jedem PCI-Bus, der *Add-In Card*s nach den 5V/33MHz-, 3,3V/33MHz- und 3,3V/66MHz-Signalisierungsstandards unterstützt, *Add-In Card*s nach dem 5V/33MHz-Standard zu betreiben, wobei deren Gesamtverlustleistung allerdings unter 17,5 W liegen muß. Die implementierte Monitorfunktion kann über den gesamten 4-GB-Adressraum des 32-Bit-PCI-Busses mit einer Granularität von 4096 Byte eingesetzt werden. Obgleich die mechanische Lösung mit einer „Karte in einer Karte“ sicher nicht die stabilste oder für den Massenanwender gedacht ist, bietet sie dem Entwickler von Hardware und Software durch ihre freie Programmierbarkeit reichlich Flexibilität und die Aussicht, unterschiedliche Algorithmen implementieren und unter praxisnahen Bedingungen testen zu können. Obgleich die Schaltung in ihrer schaltungstechnischen Realisierung durch Einschränkungen bezüglich der Verfügbarkeit von Bauteilen und anderer unvorhersehbarer Umstände eher als suboptimal zu bewerten ist, konnten alle wesentlichen Forderungen der Aufgabenstellung erfüllt werden. Wünschenswert wäre sicher noch die Möglichkeit, einen *Power Down* des *Secondary Bus*ses zu erlauben, was die (De-)Installation einer zu testenden *Add-In Card* bei laufendem Testrechner erlauben würde.

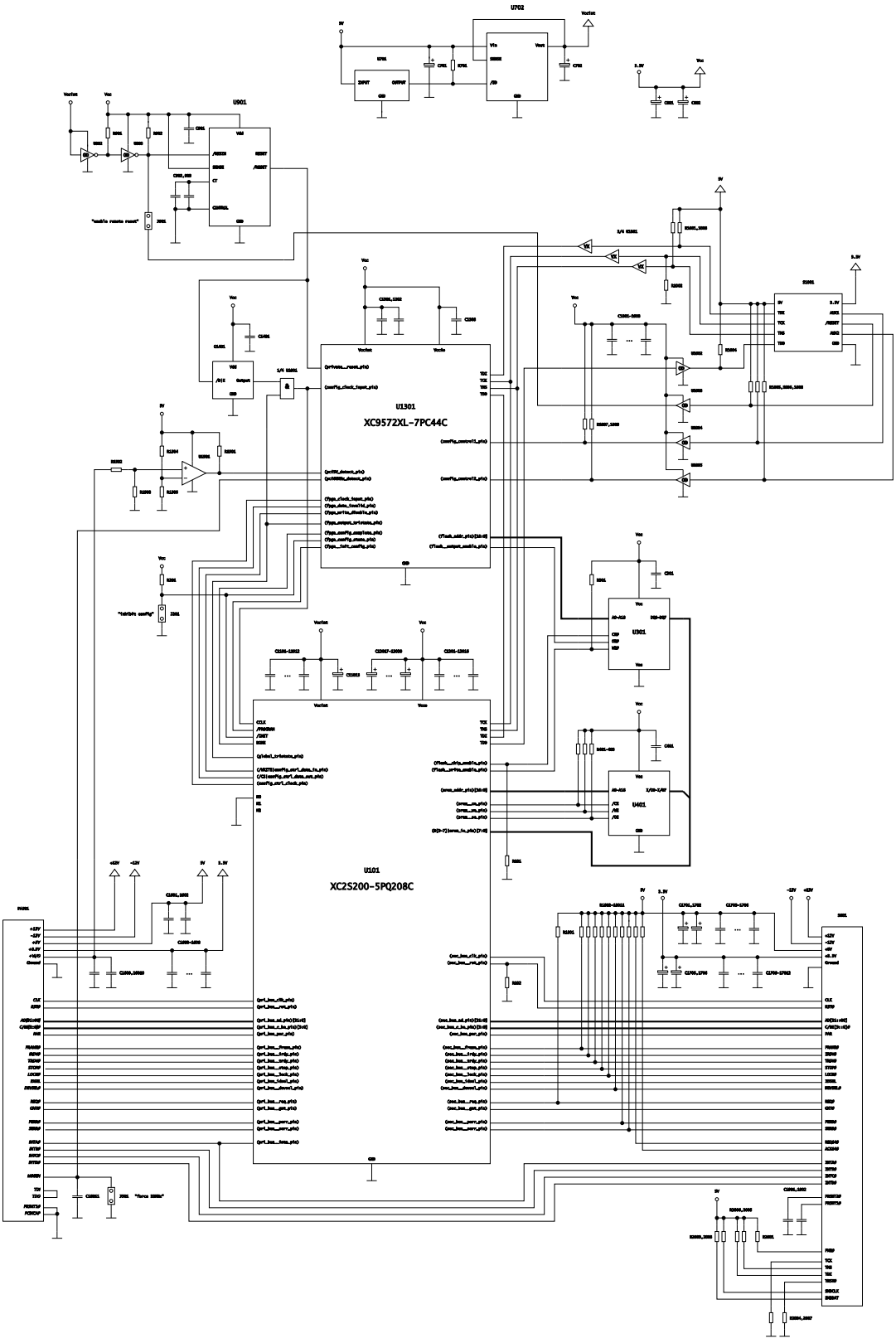
Anhang A. Blockschaltbild der *PCI-to-PCI Bridge*

Abbildung A.1. Blockschaltbild der *PCI-to-PCI Bridge*



Anhang B. Schaltbild der *Add-In Card*

Abbildung B.1. Schaltbild der *Add-In Card*



Anhang C. Konfigurationsregister der *PCI-to-PCI Bridge*

Die Implementierung der vom PCI-Standard definierten Konfigurationsregister erfolgt exakt nach den darin aufgestellten Vorgaben. Konsequenterweise werden Schreibzugriffe auf nicht definierte Register ignoriert und Lesezugriffe auf nicht definierte oder implementierte Register liefern `0x0`.

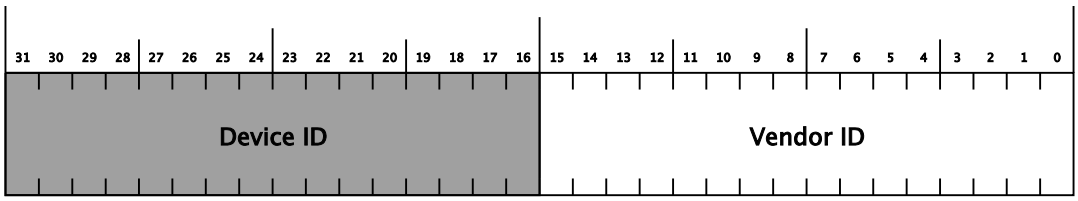
Abbildung C.1. Adressbereich der PCI-Konfigurationsregister

3	2	1	0	
Device ID		Vendor ID		0x0
Status		Command		0x1
Class Code			Revision ID	0x2
BIST	Header Type	Latency Timer	Cache Line Size	0x3
Monitor Control Registers Base Address				0x4
Base Address 1				0x5
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	0x6
Secondary Status		I/O Limit	I/O Base	0x7
Memory Limit		Memory Base		0x8
Prefetchable Memory Limit		Prefetchable Memory Base		0x9
Prefetchable Base Upper 32 Bits				0xA
Prefetchable Limit Upper 32 Bits				0xB
I/O Limit Upper 16 Bits		I/OBase Upper 16 Bits		0xC
Reserved			Capability Pointer	0xD
Expansion ROM Address				0xE
Bridge Control		Interrupt Pin	Interrupt Line	0xF

Vendor ID

Das Vendor ID Register ist ein Leseregister, welches den Hersteller eines PCI-Geräts eindeutig identifiziert. Ein Lesezugriff liefert `0x404`.

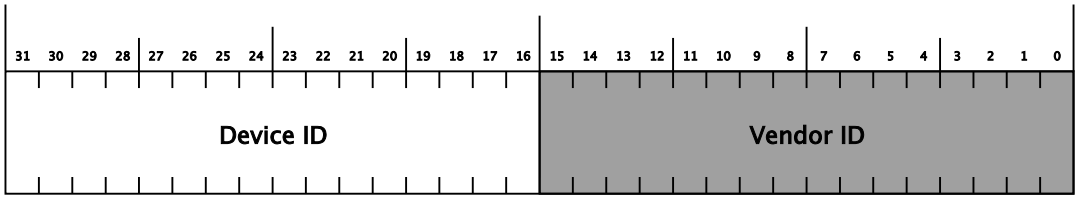
Abbildung C.2. Vendor ID Register



Device ID

Das Device ID Register ist ein Leseregister, welches zur eindeutigen Identifizierung eines PCI-Geräts innerhalb des Namensraumes des Herstellers dient. Ein Lesezugriff liefert `0x1`.

Abbildung C.3. Device ID



Command

Das Command Register bietet der Systemsoftware die Möglichkeit, das Verhalten von Geräten am PCI-Bus in deren grundlegender Funktion festzulegen. Nicht implementierte Funktionalität wird durch Lesezugriffe angezeigt, die eine `0` liefern, nachdem vorher ein Schreibzugriff mit einer `1` unternommen wurde.

Abbildung C.4. Command Register

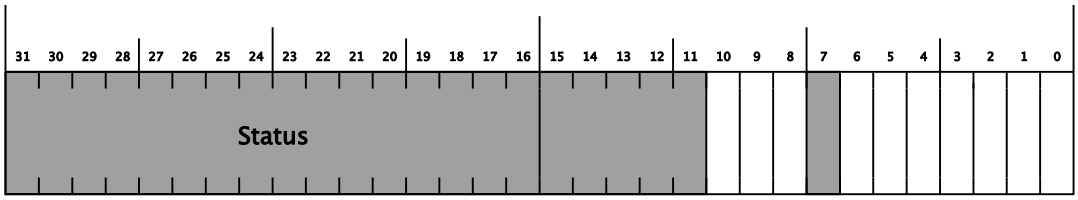


Tabelle C.1. Command Register

Bit	Name	Typ	Beschreibung
0	IO Space	L	Nicht implementiert. Das Gerät reagiert auf IO-Space-Transaktionen weder für die lokale Funktion noch für Transaktionen, die für den *Secondary Bus* bestimmt sind.
1	Memory Space	L, S	Implementiert. Das Gerät verfügt über Adressdekoder und reagiert auf Adressen innerhalb seines zugewiesenen Adressbereichs.
2	Bus Master	L, S	Implementiert. Das Gerät initiiert Transaktionen auf dem Bus.
3	Special Cycles	L	Nicht implementiert. Das Gerät ignoriert Special-Cycle-Transaktionen.
4	Memory Write and Invalidate Enable	L	Nicht implementiert. Das Gerät initiiert Memory-Write-and-Invalidate-Transaktionen am Bestimmungsbus nur, wenn es eine solche am Ursprungsbus empfangen hat.
5	VGA Palette Snoop	L	Nicht implementiert.
6	Parity Error Response	L, S	Implementiert. Das Gerät reagiert auf Paritätsfehler, indem es dies seinem Gegenüber durch *PERR#* signalisiert, so es dafür konfiguriert wurde.
8	SERR#	L, S	Implementiert. Das Gerät signalisiert Fehler durch *SERR#*, so dies freigeschaltet

Bit	Name	Typ	Beschreibung
	Enable		wurde.
9	Fast Back-to-Back Enable	L	Nicht implementiert. Das Gerät initiiert keine Fast-Back-to-Back-Transaktionen.
10	Interrupt Disable	L, S	Implementiert. Das Gerät generiert Interrupts, unbeschadet der Konfiguration an anderer Stelle, so es ihm erlaubt ist.

Status

Das Status Register bietet einem Gerät die Möglichkeit, neben der Kommunikation mit seiner Treibersoftware, Zustandsinformationen in einer standardisierten Form zu liefern. Dabei liefern Bits für nicht auftretende Ereignisse stets eine `0` bei einem Lesezugriff und werden bei Schreibzugriffen ignoriert.

Abbildung C.5. Status Register

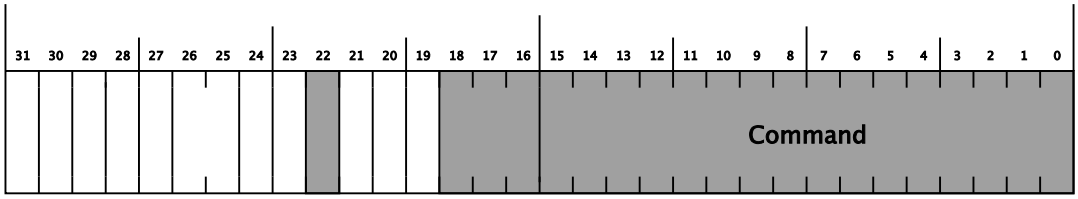


Tabelle C.2. Status Register

Bit	Name	Typ	Beschreibung
19	Interrupt Status	L	Implementiert. Ein Lesezugriff liefert eine `1`.
20	Capabilities List	L	Nicht implementiert.
21	66 MHz Capable	L	Ein Lesezugriff liefert eine `1`, falls das Gerät erkannt hat, daß es an einen 66MHz-Bus geschaltet wurde.
23	Fast Back-to-Back Capable	L	Implementiert. Ein Lesezugriff liefert eine `1`.
24	Master Data Parity Error	L, S	Implementiert. Das Bit wird gesetzt, wenn bei einer vom Gerät initiierten Transaktion ein Paritätsfehler entdeckt wurde und es dem Gerät erlaubt ist, das Auftreten eines Paritätsfehlers zu signalisieren.
25-26	DEVSEL# timing	L	Das Gerät implementiert „medium“ als die größte Verzögerung, die das *Target* benötigt, um unmißverständlich festzustellen, daß es adressiert wurde. Ein Lesezugriff liefert `0x1`.
27	Signaled Target Abort	L, S	Implementiert. Das Bit wird gesetzt, wenn das *Target* eine Transaktion mittels *Target Abort* beendet.
28	Received Target Abort	L, S	Implementiert. Das Bit wird gesetzt, wenn bei einer vom Gerät initiierten Transaktion das adressierte *Target* eine Transaktion mittels *Target Abort* beendet.
29	Received Master Abort	L, S	Implementiert. Das Bit wird gesetzt, nachdem sich kein Empfänger für eine vom Gerät initiierte Transaktion gefunden hat.

Bit	Name	Typ	Beschreibung
30	Signaled System Error	L, S	Implementiert. Das Bit wird gesetzt, nachdem das Gerät einen Systemfehler gemeldet hat.
31	Detected Parity Error	L, S	Implementiert. Das Bit wird gesetzt, wenn das Gerät bei einer Transaktion, an der es teilgenommen hat, einen Paritätsfehler detektiert.

Revision ID und Class Code

Das Class Code Register identifiziert eindeutig die Art von Gerät, welche die Systemsoftware sieht. Diese Information kann zum Auffinden von installierter Treibersoftware oder für anderweitige Initialisierungsaufgaben nützlich sein.

Abbildung C.6. Revision ID und Class Code Register

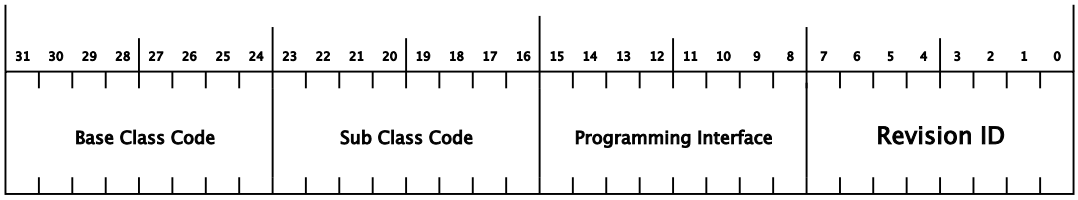


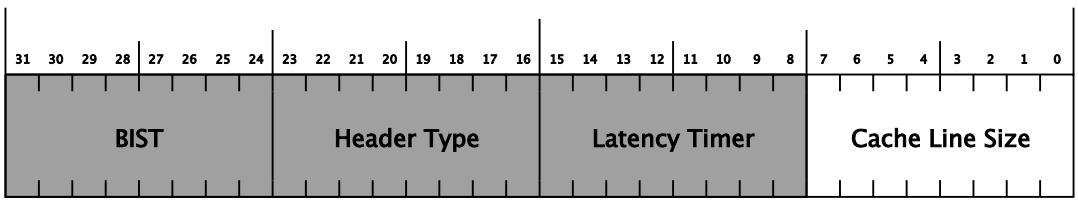
Tabelle C.3. Revision ID und Class Code Register

Bit	Name	Typ	Beschreibung
0-7	Revision ID	L	Ein Lesezugriff liefert `0x1`.
8-15	Programming Interface	L	Ein Lesezugriff liefert `0x0`.
16-23	Sub Class Code	L	Ein Lesezugriff liefert `0x04`.
24-31	Base Class Code	L	Ein Lesezugriff liefert `0x06`.

Cache Line Size

Das Cache Line Size Register gibt die in einem System optimale Granularität von Datentransfers an. Das Gerät nutzt den Wert in diesem Register, um als *Target* bei Adressierung mittels *Cache Line Wrap* seinen Adresszähler zu programmieren und als *Master* die Abbruchbedingung für Transaktionen über mehrere Datenphasen zu gewinnen. Es unterstützt alle Größen, die durch 2^x (mit x<8, x natürlich) beschrieben werden.

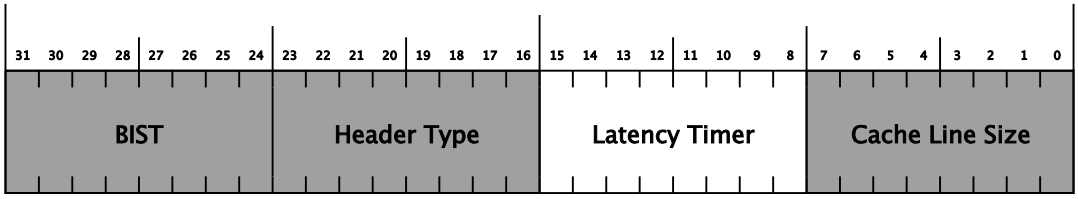
Abbildung C.7. Cache Line Size Register



Latency Timer

Das Latency Timer Register bietet in Verbindung mit der Signalisierung auf der *GNT#*-Leitung des *Master*s am *Primary Bus* eine weitere Abbruchbedingung für Transaktionen über mehrere Datenphasen.

Abbildung C.8. Latency Timer Register



Header Type

Beschreibt den Satz von Registern im zweiten, geräteabhängigen Teil des Standard-Konfigurationsraumes. Da das Gerät eine *PCI-to-PCI Bridge* implementiert, liefert ein Lesezugriff einen dieser Geräteklasse entsprechenden Wert, womit die Systemsoftware in die Lage versetzt wird, ein korrektes und vollständiges Bild der PCI-Bus-Topologie zu zeichnen, indem sie die Konfiguration auf Busse jenseits des *Secondary Bus*es ausweiten kann.

Abbildung C.9. Header Type Register

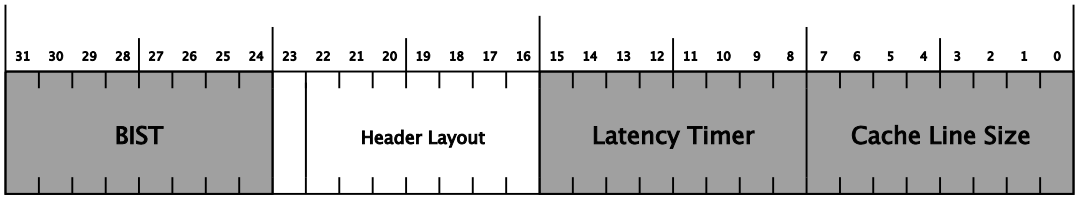


Tabelle C.4. Header Type Register

Bit	Name	Typ	Beschreibung
16-22	Header Layout	L	Ein Lesezugriff liefert `0x1`.
23	Multifunction Device	L	Ein Lesezugriff liefert eine `0`.

Monitor Control Registers Base Address

Dieses Register wird mit der Startadresse des Adressbereichs der Monitorfunktion durch die Systemsoftware programmiert. Nachdem das Register mit `0xffffffff` beschrieben wurde, liefert ein anschließender Lesezugriff einen Wert von `0xfffff000`, womit ein Bereich von 4096 Byte im *Memory Space* deklariert wird, welcher der Empfehlung im PCI-Standard für Geräte mit geringeren Anforderungen an die Größe des Adressbereichs entspricht. Die Definition der Register im Adressbereich der Monitorfunktion erfolgt detailliert in Anhang D, *Steuerregister der Monitorfunktion*.

Abbildung C.10. Monitor Control Registers Base Address

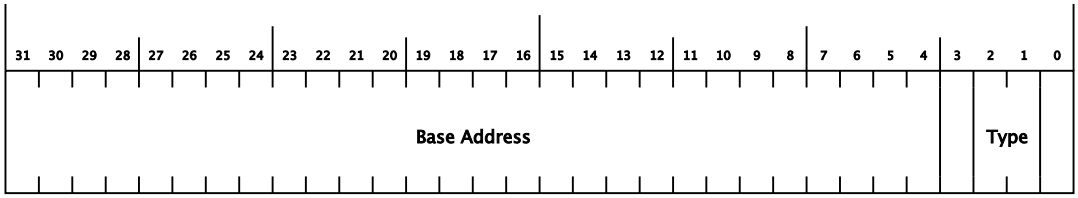


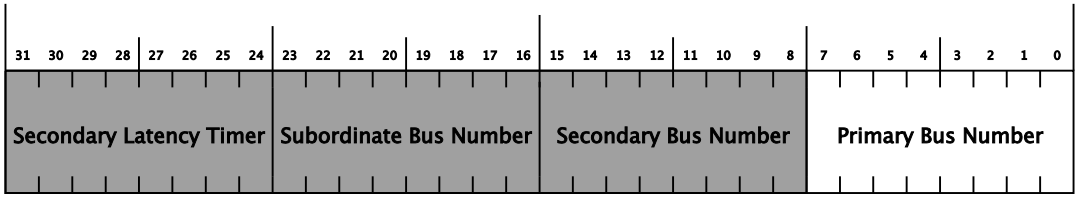
Tabelle C.5. Monitor Control Registers Base Address

Bit	Name	Typ	Beschreibung
0	Memory Space Indicator	L	Ein Lesezugriff liefert eine `0`.
1-2	Type	L	Es wird der 32-Bit-Adressraum unterstützt. Ein Lesezugriff liefert `0x0`.
3	Prefetchable	L	Vorausschauendes Lesen nicht implementiert. Ein Lesezugriff liefert eine `0`.
4-31	Base Address	L	Ein Lesezugriff liefert `0xfffff00`, was einem Adressbereich von 4096 Byte entspricht.

Primary Bus Number

Das Primary Bus Number Register wird von der Systemsoftware mit der Nummer für den *Primary Bus* programmiert.

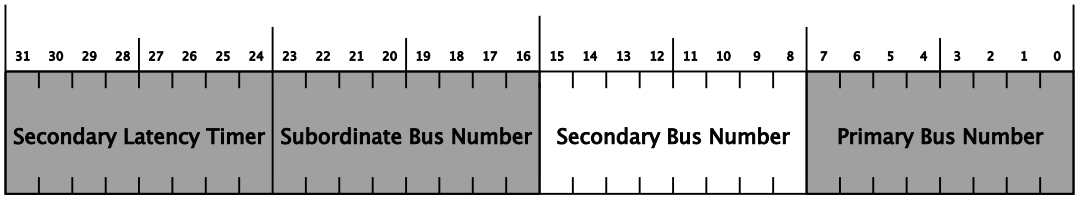
Abbildung C.11. Primary Bus Number Register



Secondary Bus Number

Das Secondary Bus Number Register wird von der Systemsoftware mit der Nummer für den *Secondary Bus* programmiert.

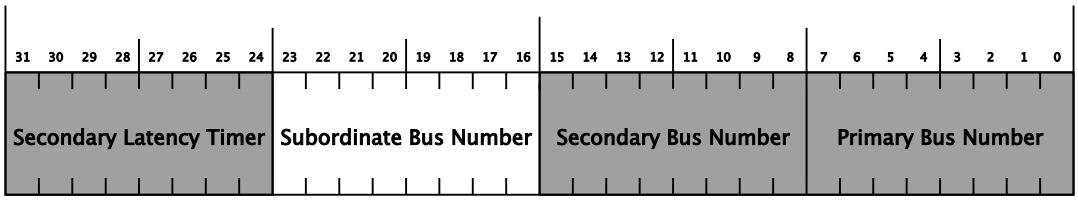
Abbildung C.12. Secondary Bus Number Register



Subordinate Bus Number

Das Subordinate Bus Number Register wird von der Systemsoftware mit der höchsten Bus-Nummer jenseits dieser *PCI-to-PCI Bridge* programmiert. In Verbindung mit dem Wert im Secondary Bus Number Register kann damit entschieden werden, ob eine Konfigurationstransaktion vom Typ 1 für ein Gerät jenseits dieser *PCI-to-PCI Bridge* bestimmt ist und die Transaktion entsprechend weitergeleitet werden soll.

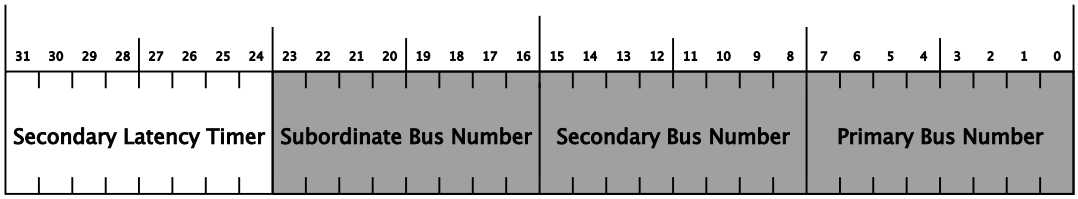
Abbildung C.13. Subordinate Bus Number Register



Secondary Latency Timer

Das Secondary Latency Timer Register bietet in Verbindung mit der Signalisierung auf der *GNT#*-Leitung des *Master*s am *Secondary Bus* eine weitere Abbruchbedingung für Transaktionen über mehrere Datenphasen.

Abbildung C.14. Secondary Latency Timer Register



Secondary Status

Das Secondary Status Register bietet für den *Secondary Bus* die gleichen Möglichkeiten der Benachrichtigung wie das Status Register am *Primary Bus*, abzüglich der Bits für *Interrupt Status* und *Signaled System Error*.

Abbildung C.15. Secondary Status Register

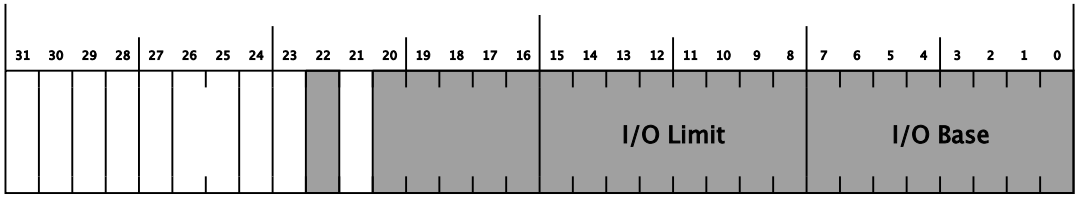


Tabelle C.6. Secondary Status Register

Bit	Name	Typ	Beschreibung
21	66 MHz	L	Nicht implementiert. Ein Lesezugriff liefert eine `0`.

Bit	Name	Typ	Beschreibung
	Capable		
23	Fast Back-to-Back Capable	L	Implementiert. Ein Lesezugriff liefert eine `1`.
24	Master Data Parity Error	L, S	Implementiert. Das Bit wird gesetzt, wenn bei einer vom Gerät initiierten Transaktion am *Secondary Bus* ein Paritätsfehler entdeckt wurde und es dem Gerät erlaubt ist, das Auftreten eines Paritätsfehlers zu signalisieren.
25-26	DEVSEL# timing	L	Das Gerät implementiert „medium“ als die größte Verzögerung, die das *Target* am *Secondary Bus* benötigt, um unmißverständlich festzustellen, daß es adressiert wurde. Ein Lesezugriff liefert `0x1`.
27	Signaled Target Abort	L, S	Implementiert. Das Bit wird gesetzt, wenn das *Target* am *Secondary Bus* eine Transaktion mittels *Target Abort* beendet.
28	Received Target Abort	L, S	Implementiert. Das Bit wird gesetzt, wenn bei einer vom Gerät initiierten Transaktion am *Secondary Bus* das adressierte *Target* eine Transaktion mittels *Target Abort* beendet.
29	Received Master Abort	L, S	Implementiert. Das Bit wird gesetzt, nachdem sich kein Empfänger für eine vom Gerät initiierte Transaktion am *Secondary Bus* gefunden hat.
30	Received System Error	L, S	Implementiert. Das Bit wird gesetzt, wenn ein Gerät am *Secondary Bus* *SERR#* signalisiert hat.
31	Detected Parity Error	L, S	Implementiert. Das Bit wird gesetzt, wenn das Gerät bei einer Transaktion am *Secondary Bus*, an der es teilgenommen hat, einen Paritätsfehler detektiert.

Memory Base und Memory Limit

Die Memory Base und Memory Limit Register werden von der Systemsoftware mit den Start- und Endadressen des Adressbereichs programmiert, der alle Geräte jenseits der *PCI-to-PCI Bridge* überdeckt. Mit diesen Werten werden die Adressdekoder programmiert, deren Ergebnis schließlich in die Entscheidung aufgenommen wird, ob die *PCI-to-PCI Bridge* die aktuelle Transaktion an den jeweils gegenüberliegenden Bus weiterleiten muß.

Abbildung C.16. Memory Base und Memory Limit Register

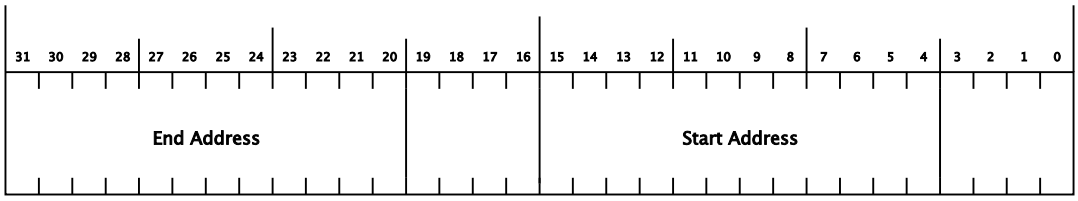


Tabelle C.7. Memory Base und Memory Limit Register

Bit	Name	Typ	Beschreibung
0-3		L	Ein Lesezugriff liefert `0x0`.
4-15	Start Address	L, S	Definiert die oberen 12 Bit im 32-Bit-Adressraum des PCI-Busses als die Startadresse des zusammenhängenden Bereichs, der dem *Secondary Bus* „hinter“ der *PCI-to-PCI Bridge* zugeordnet wurde.
16-19		L	Ein Lesezugriff liefert `0x0`.
20-31	End Address	L, S	Definiert die oberen 12 Bit im 32-Bit-Adressraum des PCI-Busses als die größte

Bit	Name	Typ	Beschreibung
			Adresse innerhalb des zusammenhängenden Bereichs, der dem *Secondary Bus* „hinter“ der *PCI-to-PCI Bridge* zugeordnet wurde.

Interrupt Line und Interrupt Pin

Mit dem Interrupt Pin Register wird der Systemsoftware die Information gegeben, an welcher der vier vom PCI-Bus definierten Interrupt-Leitungen das Gerät Interrupts generiert.

Abbildung C.17. Interrupt Line und Interrupt Pin Register

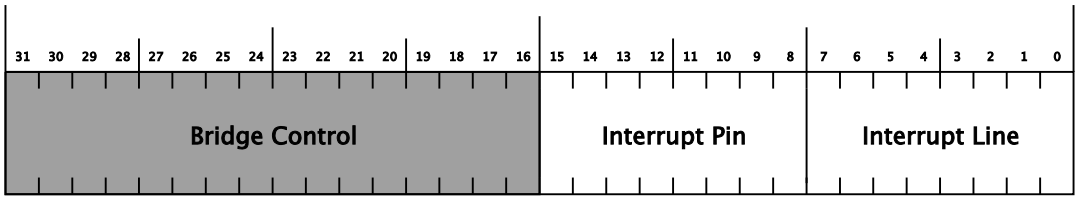


Tabelle C.8. Interrupt Line und Interrupt Pin Register

Bit	Name	Typ	Beschreibung
0-7	Interrupt Line	L, S	Implementiert. Wird von der Systemsoftware und/oder Treibersoftware benutzt.
8-15	Interrupt Pin	L	Ein Lesezugriff liefert `0x1`, womit angezeigt wird, daß das Gerät seine Interrupts an Pin A generiert.

Bridge Control

Das Bridge Control Register bietet als Erweiterung zum Command Register die Möglichkeit, speziell das Verhalten von *PCI-to-PCI Bridge*s zu kontrollieren. Die Kontrollfunktionen beeinflussen das Verhalten des Geräts an beiden Bussen.

Abbildung C.18. Bridge Control Register

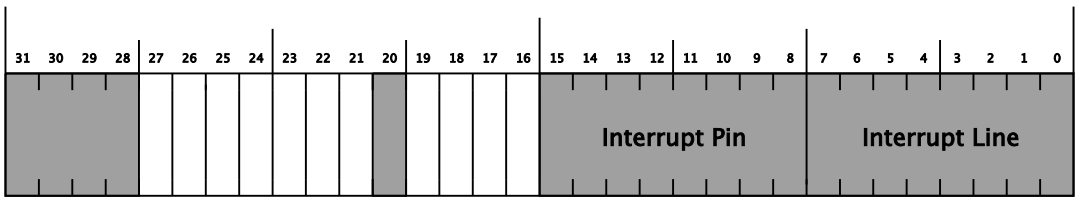


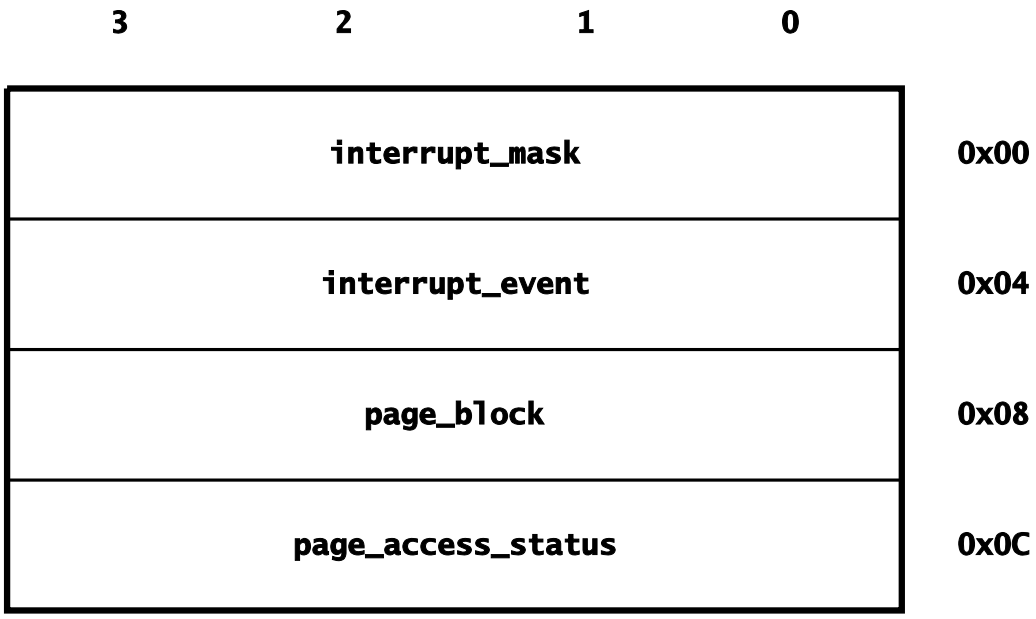
Tabelle C.9. Bridge Control Register

Bit	Name	Typ	Beschreibung
16	Parity Error Response	L, S	Implementiert. Das Gerät reagiert auf Paritätsfehler am *Secondary Bus*, indem es dies seinem Gegenüber durch *PERR#* signalisiert, so es dafür konfiguriert wurde.
17	SERR# Enable	L, S	Implementiert. Das Gerät leitet durch *SERR#* am *Secondary Bus* signalisierte Fehler an den *Primary Bus* weiter, so dies freigeschaltet wurde.
18	ISA Enable	L	Nicht implementiert.

Bit	Name	Typ	Beschreibung
19	VGA Enable	L	Nicht implementiert.
21	Master Abort Mode	L, S	Implementiert. Schreiben einer `0` bewirkt, daß sich das Gerät bei einem *Master Abort* derart verhält, als ob bei einem Lesezugriff nur Einsen geliefert werden und ein Schreibbefehl als erfolgreich angesehen wird. Durch Schreiben einer `1` wird das Verhalten bei einem *Master Abort* am Bestimmungsbus dahingehend modifiziert, als daß während einer *Delayed Transaction* mit einem *Target Abort* am Ursprungsbus und während eines *Posted Write*s durch Signalisierung über *SERR#* am *Primary Bus* reagiert wird.
22	Secondary Bus Reset	L, S	Implementiert. Schreiben einer `1` startet einen Reset am *Secondary Bus*. Der Reset hat zur Folge, daß alle FIFOs gelöscht und die Logik, die am *Secondary Bus* beteiligt ist, in ihren Ausgangszustand zurückversetzt werden.
23	Fast Back-to-Back Enable	L	Nicht implementiert. Das Gerät initiiert keine Fast-Back-to-Back-Transaktionen am *Secondary Bus*.
24	Primary Discard Timeout	L, S	Implementiert. Durch Schreiben einer `0` wird der Timeout am *Primary Bus* auf 2^{15}, bei einer `1` auf 2^{10} Taktzyklen festgelegt.
25	Secondary Discard Timeout	L, S	Implementiert. Durch Schreiben einer `0` wird der Timeout am *Secondary Bus* auf 2_{15}, bei einer `1` auf 2^{10} Taktzyklen festgelegt.
26	Discard Timer Status	L, S	Implementiert. Lesen einer `1` gibt an, daß eine *Delayed Completion* an einem der beiden Busse verworfen wurde.
27	Discard Timer SERR# Enable	L, S	Implementiert. Durch Schreiben einer `1` wird das Erreichen des Timeouts an *Primary Bus* oder *Secondary Bus* durch die Signalisierung mittels *SERR#* am *Primary Bus* mitgeteilt.

Anhang D. Steuerregister der Monitorfunktion

Abbildung D.1. Adressbereich der Monitorfunktionsregister



interrupt_mask

Das `interrupt_mask` Register erlaubt die explizite Freischaltung der asynchronen Benachrichtigung der vom Gerät registrierten Ereignisse über Interrupts.

Nach einem Reset werden keine Interrupts generiert, d.h., ein Lesezugriff liefert `0x0`.

Abbildung D.2. `interrupt_mask` Register

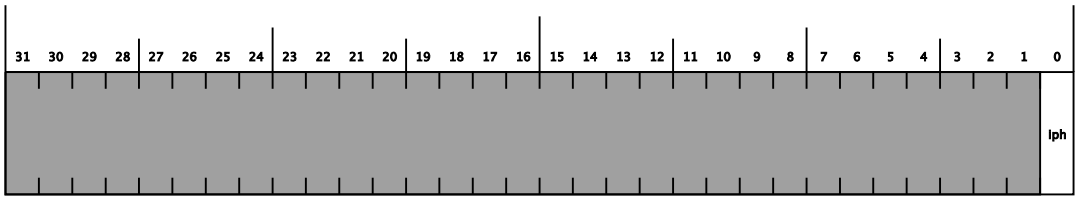


Tabelle D.1. `interrupt_mask` Register

Bit	Name	Typ	Beschreibung
0	ipa	L, S	Schreiben einer `1` schaltet die Interrupt-Erzeugung ein für den Fall des Zugriffs auf eine Adresse innerhalb eines nicht freigegebenen Adressbereichs. Nach Schreiben einer `0` wird kein Interrupt mehr erzeugt, eine Transaktion mit einem Zugriff auf eine nicht freigegebene Adresse wird dennoch abgebrochen.
1-31		L	Schreibzugriffe werden ignoriert, während Lesezugriffe `0x0` zurückliefern.

interrupt_event

Nach jedem Interrupt sollte zunächst das `interrupt_event` Register nach möglichen Ursachen konsultiert und durch Rücksetzen der entsprechenen Bits deren Kenntnisnahme quittiert werden. Solange dies nicht geschieht, werden neue Ereignisse der gemeldeten Art ignoriert und kein diesbezüglicher Interrupt ausgelöst. Mit einem Interrupt kann das Auftreten mehrerer verschiedener Ereignisse simultan gemeldet werden, es sollten also alle definierten Bits konsultiert werden.

Dieses Register ist nach einem Reset gelöscht, d.h., ein Lesezugriff liefert `0x0`.

Abbildung D.3. `interrupt_event` Register

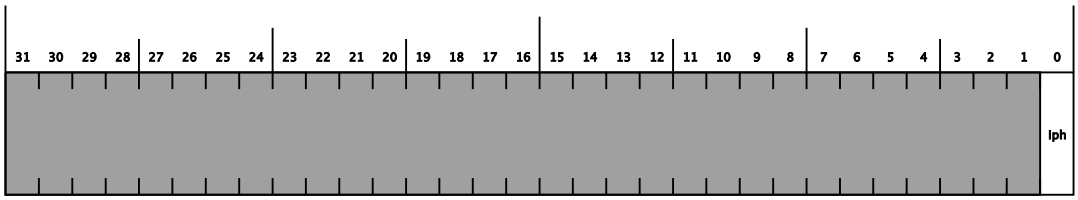


Tabelle D.2. `interrupt_event` Register

Bit	Name	Typ	Beschreibung
0	ipa	L, S	Eine `1` zeigt nach einem Interrupt an, daß ein Zugriff auf einen nicht freigeschalteten PCI-Adressbereich vom Gerät am *Secondary Bus* unternommen wurde. Schreiben einer `0` quittiert die Kenntnisnahme der Benachrichtigung.
1-31		L	Schreibzugriffe werden ignoriert, während Lesezugriffe `0x0` zurückliefern.

page_block

Der Zugriff auf PCI-Adressbereiche kann in Blöcken zu acht aufeinanderfolgenden, ausgerichteten Bereichen von je 4096 Byte Größe verwaltet werden. Dazu ist es notwendig, die Startadresse des ersten Bereiches innerhalb eines solchen Blockes zu übergeben, wozu dieses Register dient. Diese Adresse ist ausgerichtet auf das Achtfache der Größe eines dieser Bereiche. Überlappende Blöcke werden damit vermieden.

Nach einem Reset ist dieses Register auf `0x0` zurückgesetzt. Zugriffe können also für die untersten acht PCI-Adressbereiche von 4096 Byte Größe, beginnend mit `0x0`, kontrolliert werden.

Abbildung D.4. `page_block` Register

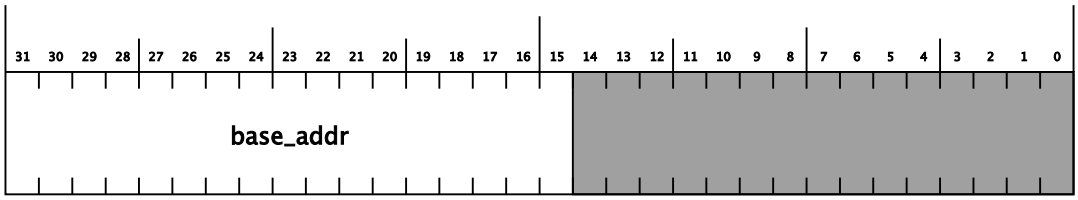


Tabelle D.3. `page_block` Register

Bit	Name	Typ	Beschreibung
15-32	base_addr	L, S	Enthält die Startadresse für den Block der acht aufeinanderfolgenden Adressbereiche, deren Zugriffsrechte in `page_access_status` spezifiziert

Bit	Name	Typ	Beschreibung
			werden können.
1-14		L	Schreibzugriffe werden ignoriert, während Lesezugriffe `0x0` zurückliefern.

page_access_status

Auf Adressbereiche, die mittels des `page_block` Registers spezifiziert wurden, kann der Zugriff entweder genemigt oder versagt werden. Die Möglichkeit, dies festzulegen, wird vom `page_access_status` Register geboten. Es erlaubt die Verwaltung der acht aufeinanderfolgenden Adressbereiche, die durch die Bits 0 bis 7 eindeutig bezeichnet werden. Bit 0 gibt den Bereich an, der durch die Startadresse im `page_block` Register beschrieben wird. Bit 1 beschreibt demnach den Bereich, der mit einem positiven Versatz von 4096 Byte bezüglich der Startadresse beginnt und seinerseits wiederum 4096 Byte groß ist uswusf.

Dieses Register ist außerhalb eines Resets für die im `page_block` Register genannte Startadresse gültig.

Abbildung D.5. `page_access_status` Register

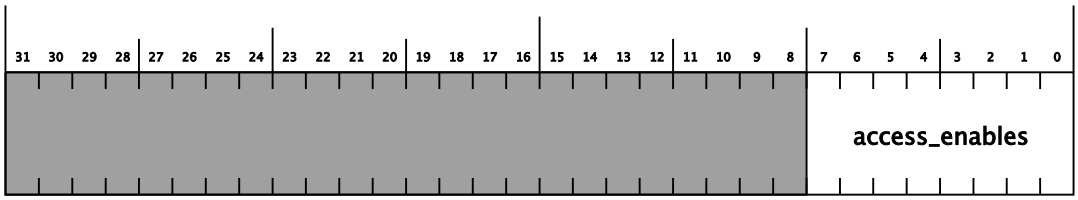


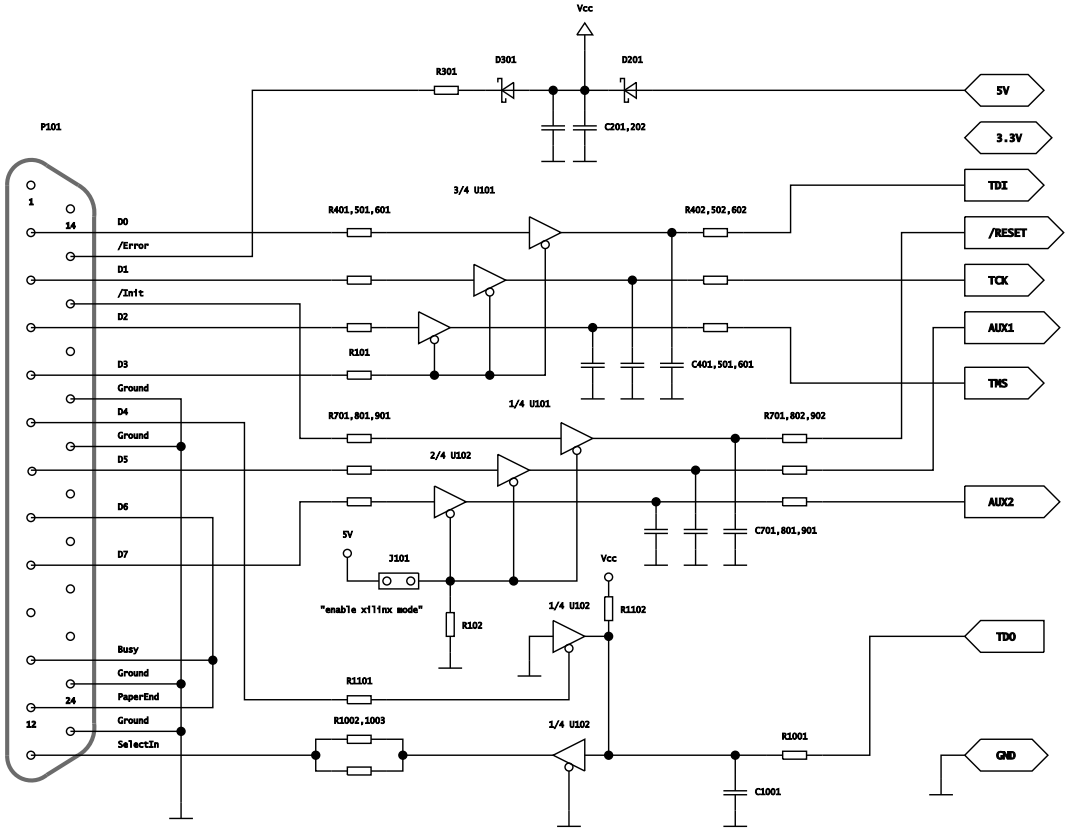
Tabelle D.4. `page_access_status` Register

Bit	Name	Typ	Beschreibung
0-7	access_enables	L, S	Enthält die Zugriffsrechte für acht aufeinanderfolgende PCI-Adressbereiche, deren Startadresse in `page_block.base_addr` spezifiziert wurde. Das Setzen eines Bits erlaubt den Zugriff auf den entsprechenden Bereich innerhalb des Blocks, während ein Rücksetzen ihn verwehrt.
8-31		L	Schreibzugriffe werden ignoriert, während Lesezugriffe `0x0` zurückliefern.

Anhang E. *JTAG*-Adapter für den IBM-PC-Parallelport

Zur Anpassung einer einfachen *JTAG*-Schnittstelle an die parallele Schnittstelle eines IBM-PCs wurde ein Adapter entwickelt, dessen Schaltbild Abbildung E.1, „Schaltbild des *JTAG*-Adapters für den IBM-PC-Parallelport“ wiedergibt.

Abbildung E.1. Schaltbild des *JTAG*-Adapters für den IBM-PC-Parallelport



Die Schaltung ist im Wesentlichen an die Empfehlung von Xilinx angelehnt worden [XJTAG]. Damit wird der Adapter mit der *ISP*-Software von Xilinx einsetzbar, was für die Programmierung des CPLDs Voraussetzung ist, da hierfür keine eigene Software verfügbar ist. Neben den vier vorgeschriebenen *JTAG*-Signalen *TCK*, *TMS*, *TDI* und *TDO* werden zusätzlich die Signale *RESET*, *AUX1* und *AUX2* angeboten. Die Bedeutung von *AUX1* und *AUX2* ist bewußt nicht definiert und kann von der steuernden Software und/oder gesteuerten Schaltung vergegeben werden. Der Brückenstecker `J101` erlaubt das Hochohmigschalten der Treiber für die zusätzlichen Signale, da nicht bekannt ist, zu welchem Zeitpunkt welche Pegel an die von den Zusatzsignalen benutzten Leitungen am Parallelport von der Xilinx-Software gelegt werden.

Literaturquellen

[PCI23] *PCI Local Bus Specification Revision 2.3*. PCI-SIG. May 29, 2002.

[PCIARCH] Tom Shanley und Don Anderson. *PCI System Architecture*. Mindshare. 1999.

[TTLCMOS] E. Kühn. *Handbuch TTL- und CMOS-Schaltkreise*. Verlag Technik. 1986.

[HYPERT] http://www.hypertransport.org/tech/tech_specs.cfm . HyperTransport Consortium.

[INFINIB] http://www.infinibandta.org/specs . InfiniBand Trade Association.

[PCIEX] *PCI Express Base Specification Revision 1.0*. PCI-SIG. April 29, 2002.

[DS001] *Spartan-II 2.5V FPGA Family: Complete Data Sheet*. Xilinx. August 2, 2004.

[ALLIANCE] *AS7C31025A Datasheet v.0.9.4*. Alliance Semiconductor. 5/17/01.

[DS054] *XC9500XL High-Performance CPLD Family Data Sheet (v1.8)*. Xilinx. August 2, 2004.

[DS057] *XC9572XL High Performance CPLD*. Xilinx. September 15, 2004.

[AMD21354] *Am29LV040B Data Sheet Publication Number 21354 Rev. E Amendment +1*. AMD. June 11, 2004.

[FOX] *F4100 SERIES Data Sheet* . FOXElectronics. 6/1/04.

[DS101267] *LP3963/LP3966 3A Fast Ultra Low Dropout Linear Regulators*. National Semiconductor Corporation. May 2003.

[SLVS087L] *TLC7733 Micropower Supply Voltage Supervisors*. Texas Instruments Incorporated. FEBRUARY 2003.

[SUPERVSR] *TS831 MICROPOWER VOLTAGE SUPERVISOR RESET ACTIVE LOW*. STMicroelectronics. November 2001.

[SLCS136L] *LMV331 SINGLE, LMV393 DUAL, LMV339 QUAD, GENERAL-PURPOSE LOW-VOLTAGE COMPARATORS* . Texas Instruments Incorporated. 2004.

[SCES295N] *SINGLE INVERTER BUFFER/DRIVER WITH OPEN-DRAIN OUTPUT*. Texas Instruments Incorporated. 2003.

[SCES296N] *SINGLE BUFFER/DRIVER WITH OPEN-DRAIN OUTPUT*. Texas Instruments Incorporated. 2003.

[DS012411] *74LCX08 Low Voltage Quad 2-Input AND Gate with 5V Tolerant Inputs* . Fairchild Semiconductor Corporation. 2004.

[HC125] *M74HC125*. STMicroelectronics. August 2001.

[MBRS130] *MBRS130T3*. Semiconductor Components Industries. October, 2000.

[TAJ] *TAJ Series Standard Tantalum*. AVX Corporation.

[MERITEC] *PCI Connectors and Card Edge Connectors PCI R498 10M*. Meritec.

[XAPP176] *Configuration and Readback of the Spartan-II and Spartan-IIE Families (v1.0)*. Xilinx. March 12, 2002.

[XAPP188] *Configuration and Readback of Spartan-II FPGAs Using Boundary Scan (v2.1)*. Xilinx. March 27, 2002.

[XAPP178] *Configuring Spartan-II FPGAs from Parallel EPROMs (v0.9)*. Xilinx. December 3, 1999.

[XAPP179] *Using SelectI/O Interfaces in Spartan-II and Spartan-IIE FPGAs (v2.0)*. Xilinx. July 23, 2002.

[XAPP175] *High Speed FIFOs In Spartan-II FPGAs (v1.0)*. November 23, 1999. Xilinx.

[XAPP173] *Using Block SelectRAM+ Memory in Spartan-II FPGAs (v1.1)*. Xilinx. December 11, 2000.

[XAPP189] *Powering Xilinx Spartan-II FPGAs (v1.1)*. Xilinx. July 20, 2001.

[XAPP450] *Power-On Requirements for the Spartan-II and Spartan-IIE Families (v1.0)*. Xilinx. November 15, 2001.

[XAPP623] *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors (v2.0)*. Xilinx. April 5, 2004.

[XAPP115] *Planning for High Speed XC9500XL Designs (v1.0)*. Xilinx. September 28, 1998.

[XAPP114] *Understanding XC9500XL CPLD Power (v1.1)*. Xilinx. January 22, 1999.

[XAPP111] *Using the XC9500XL Timing Model (v1.3)*. August 20, 2001. Xilinx.

[XAPP215] *Design Tips for HDL Implementation of Arithmetic Functions (v1.0)* . Xilinx. June 28, 2000.

[XAPP052] *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators (v1.1)* . Xilinx. July 7,1996.

[XAPP014] *Ultra-Fast Synchronous Counters (001)*. Xilinx.

[XAPP027] *Implementing State Machines in LCA Devices (001)*. Xilinx.

[SCEA035A] *Selecting the Right Level-Translation Solution*. Texas Instruments Incorporated. June 2004.

[XJTAG] http://toolbox.xilinx.com/docsan/3_1i/data/common/jtg/dppb/appb.htm . Xilinx.

Glossar

Begriffe des PCI-Busses

Add-In Card	Erweiterung des Begriffs *Device*, der die mechanische und elektrische Integration eines *Agent*s an den PCI-Bus beschreibt. *Add-In Card*s sind Baugruppen, die vom Nutzer ohne Spezialwerkzeuge in dafür vorgesehene Steckplätze installiert werden können.
Address Phase	Zustand des PCI-Busses während eines Taktzyklusses, bei dem Adressierungs- sowie Steuerdaten vom *Master* zum *Target* übertragen werden.
Address Space	Gibt den Definitionsbereich an, innerhalb dessen auf dem PCI-Bus zu übertragende Daten Gültigkeit besitzen. Bietet überdies eine Möglichkeit, nach der *Target*s an einem PCI-Bus entsprechend ihrer Adressierbarkeit eingeteilt werden können.
Agent	Bezeichnet allgemein einen Teilnehmer am PCI-Bus, der die PCI-Protokolle implementiert. *Agent*s können entweder nur *Target*- oder sowohl *Target*- als auch *Master*-Funktionalität besitzen.
Arbiter	Vom Standard nicht weiter definierte übergeordnete Instanz, die Zugriffsanfragen auf den PCI-Bus entgegennimmt und Zugriffsrechte verteilt.
Arbitration Parking	Beschreibt den Zustand des PCI-Busses, während dessen keine Transaktion aktiv ist, einem *Master* jedoch der Zugriff auf den Bus ohne dessen explizite Anfrage gewährt wurde.
Bridge	Gerät, welches den Übergang von einem PCI-Bus zu einem anderen Transportmedium erlaubt. Verfügt über geeignete Resourcen, den (in den überwiegenden Fällen) bi-direktionalen Datentransport zwischen PCI-Bus und dem anderen Transportmedium zu entkoppeln. Bedient sich auf PCI-Seite der *Delayed Transaction*s.
Bus Parking	Synonym zu *Arbitration Parking*.
Component	Beschreibt die Gesamtheit der elektrischen Eigenschaften eines Teilnehmers am PCI-Bus.
Configuration Space	Wohl definierte Möglichkeit, die Funktionalitäten, die von *Agent*s an einem PCI-Bus geboten werden, zu klassifizieren und entsprechend ihrer Eigenschaften Systemresourcen zuzuweisen oder anderweitig in das System zu integrieren und dadurch für den Datenaustausch über den PCI-Bus nutzbar zu machen.
Data Phase	Zustand des PCI-Busses während eines oder mehrerer Taktzyklen einer Transaktion, bei dem Nutzdaten im Maximum der Busbreite übertragen werden können.
Delayed Completion	Das Ergebnis eines *Delayed Request*s nach Beendigung der Transaktion am Bestimmungsbus. Besteht aus den Merkmalen einer Transaktion am Ursprungsbus und dem gelieferten Ergebnis am Bestimmungsbus.
Delayed Request	Beschreibt die Merkmale einer Transaktion, welche durch *Retry* am Ursprungsbus, aber noch nicht am Bestimmungsbus, beendet wurde.
Delayed Transaction	Beschreibt die Methode eines *Target*s, eine Transaktion mit *Retry* zu beenden, alle Merkmale der Transaktion jedoch zu sichern, um unabhängig

	eines erneuten Versuchs des *Master*s die Daten für dessen wiederholte Anfrage selbstständig bereitzustellen und später auf eine erneute diesbezügliche Anfrage ohne Verzögerung antworten zu können. Wird von *Bridge*s auf Seiten des PCI-Busses benutzt.
Device	Ist die Summe der Eigenschaften von *Agent* und *Component*. In der Regel bezeichnet der Begriff eine schaltungstechnische Realisierung, z.B. in Form eines integrierten Schaltkreises.
downstream	Gibt die Richtung einer Transaktion an, die über eine *PCI-to-PCI Bridge* von einem PCI-Bus oberer Hierarchieebene zu einem PCI-Bus einer unteren Ebene gerichtet ist (wortwörtlich: *stromabwärts*).
Function	Separater und unabhängiger Kontext innerhalb eines *Target*s, der durch den nur ihm zugewiesenen Adressbereich eindeutig über den PCI-Bus adressierbar ist. Jede *Function* muß durch genau einen *Configuration Space* eindeutig identifizierbar und konfigurierbar sein, um am Datenaustausch über den PCI-Bus teilnehmen zu können.
Latency	Gibt ein Maß für die Reaktionsfähigkeit eines *Agent*s während einer Transaktion an.
Master	Beschreibt die Gesamtheit aller Eigenschaften und Funktionen eines *Agent*s, welche mit dem agierenden (aktiven) Teilnehmer einer Transaktion assoziiert sind.
PCI-to-PCI Bridge	Als Spezialfall einer *Bridge* verbindet dieses Gerät zwei PCI-Busse miteinander und leitet Transaktionen von einem Bus zu einem anderen weiter. Benötigt dazu pro Bus jeweils einen *Master* als auch ein *Target*. Erlaubt die Bildung einer baumähnlichen Struktur von hierarchisch geordneten PCI-Bussen.
Primary Bus	Der einer oberen Hierarchieebene zugewandte Bus einer *PCI-to-PCI Bridge*.
Posted Write	Beschreibt einen Schreibzugriff, bei dem ein *Target* im Sinne des beschleunigten Datentransfers die Daten unabhängig von deren schlußendlicher Bestimmung oder Verarbeitung zunächst „blind“ aufnimmt, damit jedoch die Verantwortung für deren Weiterleitung übernimmt. Da ein *Master* nicht unterscheiden kann, ob es sich bei einem Schreibzugriff um einen *Posted Write* handelt, darf er wie bei jeder Transaktion davon ausgehen, daß nach deren erfolgreichem Abschluß die Daten für eine andernfalls nötige Wiederholung der Transaktion nicht mehr gesichert werden müssen.
Retry	Beschreibt eine Beendigung einer Transaktion durch das *Target*, bei der keine Daten übertragen werden. Gibt in der Regel nur eine kurzzeitige Verhinderung des *Target*s auf Grund von gefüllten FIFOs und dergleichen an.
Secondary Bus	Der einer oberen Hierarchieebene abgewandte Bus einer *PCI-to-PCI Bridge*.
Target	Beschreibt die Gesamtheit aller Eigenschaften und Funktionen eines *Agent*s, welche mit dem reagierenden (passiven) Teilnehmer einer Transaktion assoziiert sind. *Target*s müssen mindestens eine und können darüberhinaus mehrere *Function*s implementieren.
Transaction	Eine Transaktion auf dem PCI-Bus ist die wohl definierte Abfolge von einer (oder, bei 64-Bit-Adressierung auf 32-Bit-Datenbreite, zwei) Address- und einer oder mehreren Datenphasen, während derer Daten übertragen werden können. Es kann zur gleichen Zeit stets nur eine einzige Transaktion auf dem Bus aktiv sein.

Turnaround Cycle — Ein Taktzyklus, während dessen keiner der Teilnehmer die gemeinsamen Daten- oder Steuerleitungen treibt, um das gleichzeitige Treiben der Leitungen und die damit verbundene Zerstörung der Treiber zu vermeiden. Wird beispielsweise notwendig, wenn bei einem Lesezugriff der *Master* in der Adressphase die Datenleitungen treibt, im Anschluß daran aber vom *Target* Daten erwartet.

upstream — Gibt die Richtung einer Transaktion an, die über eine *PCI-to-PCI Bridge* von einem PCI-Bus unterer Hierarchieebene zu einem PCI-Bus einer oberen Ebene gerichtet ist (wortwörtlich: *stromaufwärts*).

Begriffe der Computertechnik

DMA — Beschreibt einen Datentransfer in den oder aus dem physikalischen Speicher, der nicht von der CPU sondern einem peripheren Gerät initiiert und ausgeführt wird. Entzieht sich der Kontrolle durch die CPU und damit der virtuellen Speicherverwaltung mit ihren Schutzfunktionen.

Mediator — Teil der Speicherverwaltung eines Betriebssystems, der Anfragen nach physikalischem Speicher für *DMA*-Operationen bewertet und nur den tatsächlich zur Verfügung gestellten Adressbereich explizit für das Gerät freischaltet, welches die *DMA*-Operation ausführen soll.

Treiber — Beschreibt (in den überwiegenden Fällen) einen Teil der Betriebssystem-Software, der die Anpassung eines Gerätes an die vom Betriebssystem definierten Software-Schnittstellen zur Steuerung des Geräts und Datenübertragung über das Gerät vornimmt.

Begriffe der digitalen Schaltungstechnik

Boundary Scan — Synonym zu *JTAG*.

Bus — Beschreibt eine schaltungstechnische Struktur von mehreren, gleichzeitig betriebenen Sammelleitungen, bei der Sender und Empfänger nicht nur an deren Enden, sondern auch an beliebigen Stellen entlang der Leitungen galvanisch angeschaltet werden können [TTLCMOS]. Man bezeichnet diese Art der Anordnung der Teilnehmer auch als *multi-drop*. Elektrisch entspricht dies einer Parallelschaltung aller Ein- und Ausgänge mit entsprechenden Konsequenzen für die schaltungstechnische Auslegung der niederohmigen Ausgangsstufen, die in den überwiegenden Fällen mit einem dritten, hochohmigen Zustand ausgestattet werden, damit keine zwei Ausgänge gleichzeitig die selben Leitungen treiben. In letzterem Falle könnte ein auf Nullpotential gezogener Ausgang in die Gefahr der Zerstörung durch Aufnahme des vollen Kurzschlußstromes aller auf H-Pegel liegenden Ausgänge geraten.

CPLD — Steht für *Complex Programmable Logic Device* und beschreibt ein überwiegend hochintegriertes, digitales Halbleiterbauelement, dessen innere Schaltungsstruktur durch die Belegung nichtflüchtiger Speicherzellen (praktisch) beliebig oft und gegebenenfalls direkt in der eingebauten Schaltung konfiguriert werden kann. Ist, im Gegensatz zu *FPGA*s, nach Anlegen der Betriebsspannung und üblichwerweise im Mikrosekundenbereich liegender Initialisierung sofort betriebsbereit.

Dual-Ported RAM — Speicherbaustein, der den freien und gleichzeitigen Zugriff von zwei unabhängigen Kontexten auf den gesamten Adressraum des Speichers zuläßt, indem die Zugriffe intern durch zusätzliche Logik synchronisiert werden. Wird üblicherweise zur Implementierung von FIFOs eingesetzt.

Entkoppelkapazität	Wird für die Speicherung von Ladungen möglichst nahe an den Versorgungsklemmen eines aktiven Bauelementes platziert, um für die Dauer des Schaltvorgangs als niederohmige Spannungsquelle einen durch den Schaltstrom verursachten Abzug von bezüglich des Bauelements lokalen Ladungsträgern zu verhindern. Ein derart hervorgerufener Ladungsträgermangel würde zu einem Abfall des Betriebsspannungspegels führen, was wiederum das Auftreten falscher Pegel innerhalb des Bauelementes zur Folge hätte. Zusätzlich werden Stromspitzen und damit Spannungseinbrüche auf den Versorgungsleitungen und Spannungsspitzen auf den Masseleitungen auf dem Weg zum und vom Bauelement reduziert. Wird gelegentlich auch als Stützkondensator bezeichnet.
FPGA	Steht für *Field Programmable Gate Array* und beschreibt ein überwiegend hochintegriertes, digitales Halbleiterbauelement, dessen innere Schaltungsstruktur durch die Belegung statischer Speicherzellen beliebig oft und direkt in der eingebauten Schaltung konfiguriert werden kann. Verliert nach Abschalten der Betriebsspannung den Inhalt seiner Konfigurationsregister, muß also nach erneutem Anlegen der Betriebsspannung erst konfiguriert werden, um betriebsbereit zu sein.
ground bounce	Störender Effekt in digitalen Hochgeschwindigkeitsschaltungen, der die Verschiebung des Nullpotentials während eines Schaltvorgangs durch parasitäre Induktivitäten der Leiterzüge und/oder der *Pin*s eines Schaltkreises beschreibt. Die Folge davon ist, daß im Schaltkreisinneren Spannungspegel detektiert werden, die tatsächlich gar nicht geschaltet wurden: Das Nullpotential „springt umher“. Tritt umso deutlicher in Erscheinung, je niedriger die Betriebsspannung für die innere Logik und je größer die von den Puffern/Treibern geschalteten Ströme bei immer kürzeren Schaltzeiten sind. Stark beeinflußt von der Schaltkreiskonfektionierung.
ISP	Steht für *In System Programming* und bezeichnet die Methode, programmierbare Logik direkt in der Schaltung zu programmieren, in die sie verlötet wurde. Unterscheidet sich von der althergebrachten Methode, nach der die Bauteile mit speziell dafür vorgesehenen, externen Geräten über entsprechende Sockel und Adapter programmiert werden. Damit entfallen zusätzlich die nötigen Sockel in der Zielschaltung, deren Aufgabe es wäre, die Bauteile nach deren externer Programmierung in der Schaltung zu fixieren.
JTAG	Steht für *Joint Test Action Group* und bezeichnet einen Standard für eine Testmethode, mit der hochintegrierte Schaltkreise in hochminiaturisierten Schaltungen nach deren Einbau kontaktlos getestet werden können, um die früher benutzte mechanisch problematische Technologie der *Bed Of Nails* zu vermeiden. Ist als *IEEE 1149.1* festgeschrieben.
LVDS	Steht für *Low Voltage Differential Signaling* und bezeichnet einen Standard, der die elektischen Eigenschaften von Sende- und Empfangsstrukturen innerhalb von digitalen Hochgeschwindigkeitsschaltungen definiert. Zu den besonderen Merkmalen zählen als Konstantstromquellen betriebene Ausgangsstufen, als differentielle Spannungsverstärker ausgeführte Eingangsstufen und dadurch insgesamt reduzierte Schaltleistung, womit wiederum die elektromagnetische Beeinflussung der Umgebung reduziert wird. Ist als *ANSI/TIA/EIA-644-A-2001* niedergelegt.
Reflected Wave Signaling	Betrieb einer elektrisch langen, nicht terminierten Leitung mit dem Ziel, die durch die Reflexion am offenen Leitungsende hervorgerufene rücklaufende Spannungswelle der hinaufenden Welle zu überlagern, um den im Gegensatz zu einer angepaßt abgeschlossenen Leitung notwendigen, höheren Schaltstrom für die Erreichung des selben Umschaltpegels zu vermeiden.