RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Stefan Philipp

Entwicklung einer PCI-Karte zur Steuerung und Überwachung nicht-lokaler Rechner für den Einsatz in Computerclustern

Diplomarbeit

HD-KIP-01-20

KIRCHHOFF-INSTITUT FÜR PHYSIK

Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik

vorgelegt von

Stefan Philipp

aus Hannover an der Leine

2001

Entwicklung einer PCI-Karte zur Steuerung und Überwachung nicht-lokaler Rechner für den Einsatz in Computerclustern

Die Diplomarbeit wurde von *Stefan Philipp* **ausgeführt am**

Kirchhoff-Institut für Physik

unter der Betreuung von

Herrn Prof. Dr.Volker Lindenstruth

Entwicklung einer PCI-Karte zur Steuerung und Überwachung nicht-lokaler Rechner für den Einsatz in Computerclustern.

Diese Arbeit beschreibt die Entwicklung einer Steckkarte auf PCI-Basis. Die Aufgabe der Karte ist die vollständige Steuerung und Überwachung des bestückten Rechners über eine Netzwerkverbindung. In der Arbeit wird ein funktioneller Prototyp erstellt und erste Funktionen aus einer Liste von Anforderungen implementiert. Die Realisierung erfolgt durch den Einsatz einer FPGA/PCI-Entwicklungskarte in Verbindung mit dem CPU-Modul µCSimm . Die Steuerung des Rechners erfolgt durch die Emulation der Eingabegeräte Maus, Tastatur und Floppy, sowie der Gehäusetaster für Stromversorgung und Reset. Eine Diagnosemöglichkeit wird durch das Mitlesen von POST-Werten beim Systemstart angeboten. Die Arbeitsweise der Karte kann mit Hilfe der erstellten Programme für das CPU-Modul demonstriert werden. Die Verbindung an das Steuerungsnetzwerk wird über einen Internet-Daemon hergestellt. Die Arbeit beschreibt die Implementierung der Hardware und Software und gibt einen Ausblick auf die weitere Entwicklung.

Development of a PCI-card to control and monitor remote systems for use in computer-clusters.

This diploma thesis describes the development of a plug-in card on the basis of PCI. The task of the card is the full control and monitoring of the host system via a network connection. A functional prototype is created and first functions are implemented based on a list of requirements. The project is realized using a FPGA/PCI development board in connection with the CPU-Module µCSimm. The control is made by the emulation of the input devices mouse, keyboard and floppy as well as the power and reset push buttons. A possibility for diagnosis is offered by catching the POST values at system startup. The functions of the card can be demonstrated by use of the developed programs for the CPU Module. The connection to the control network is realized using an internet deamon. The thesis describes the implementation of the hardware and software and gives an outlook to further development possibilities.

Inhaltsverzeichnis

Kapitel 1

Einleitung, Motivation, Problemstellung

Mit der Weiterentwicklung der Leistungsfähigkeit von Computern können zunehmend mehr Problemstellungen mit informationstechnischen Methoden gelöst werden. Doch trotz ihrer immer weiter gesteigerten Rechenkraft, sind Computer gleichzeitig immer günstiger geworden. Aus diesem Grund können heutzutage leistungsfähige Rechner zu vergleichsweise erschwinglichen Preisen erworben werden. Dies führt auch zur Entwicklung neuer Anwendungsgebiete. Bestimmte Aufgabenstellungen können erst durch den massiven Einsatz von heute verfügbarer Rechenkraft gelöst werden.

Als Beispiel sei in diesem Zusammenhang die Forschung der Hochenergiephysik am *CERN* [Cer] bei Genf, Schweiz genannt. Um zu immer höheren Energiebereichen vordringen zu können, müssen dort immer größere Detektoren gebaut werden. Die erzeugten Datenmengen und Datenraten der einzelnen Experimente am Beschleunigerring *LHC* [Lhc] beispielsweise stellen völlig neue Anforderungen an die Datenverarbeitung. Die dritte Triggerstufe (*High Level Trigger*) des Experiments *ALICE* [Ali] an diesem Ring muß beispielsweise die maximale Datenrate von 13,2 GByte/Sek verarbeiten können.

Diese Anforderungen an die Datenverarbeitung haben ursprünglich die Entwicklung *des EU-Data-Grid* [Grd][Hof01][Grid99] motiviert. Dabei handelt es sich im Wesentlichen um eine Spezifikation und Entwicklung von Software zur Zusammenschaltung und zum Betrieb von einem Netz aus weltweit verteilten Computerfarmen (*Tier Center*). Die Rechenleistung der Farmen wurde optimiert für großen Datenmengen und großen Datendurchsatz. Im Laufe der Zeit kamen weitere Anwendungsgebiete hinzu, insbesondere die Auswertung von Satellitenbildern in der Erdbeobachtung und die Gensequenzanalyse in der Biologie. Die verschiedenen Rechenfarmen des Grids sind hierarchisch geordnet. Auf unterster Ebene bestehen sie aus einer lokalen Ansammlung von untereinander vernetzten Rechnern. Eine solche größere Ansammlung vernetzter Rechner bezeichnet man üblicherweise als *Cluster* [Clus98]. Die einzelnen Rechner eines Clusters bezeichnet man als *Knoten*.

Teilweise werden solche Rechencluster aus speziellen und teuren Supercomputern aufgebaut, die selbst über mehrere tausend Prozessoren verfügen können. Die speziellen Aufgabenstellungen an das Grid kennzeichnet jedoch neben dem hohem Datendurchsatz und der hohen Datenrate meist weiterhin eine starke Teilbarkeit der Operationen. Dadurch ist der Einsatz von Supercomputern zum Aufbau

eines Rechenclusters aber nicht mehr zwingend erforderlich. Bei stark verteilbaren Aufgaben kann dieser Vorteil nicht angemessen zur Geltung kommen.

Statt dessen kann der Cluster wesentlich ökonomischer aus einer entsprechenden größeren Anzahl von leistungsschwächeren Rechnern aufgebaut werden. Durch die oben angedeutete Entwicklung der Computer sind bereits die heute üblichen PCs leistungsstark genug, um in einem solchen Rechencluster als Knoten eingesetzt werden zu können. Aus diesem Grund verwendet man zum Aufbau der Cluster leicht und in großer Zahl zu erwerbende Standardkomponenten und spricht in diesem Zusammenhang von *COTS*-Komponenten (*commercial off-the-shelf*).

Durch den Einsatz von Standardkomponenten ergeben sich eine Reihe von Vorteilen:

- Obwohl eine größere Anzahl von Rechnern erworben werden muß, ist die Gesamtrechenleistung im Vergleich zu Supercomputern wesentlich günstiger.
- Durch die Verwendung von Standardkomponenten besteht eine gewisse Grundkompatibilität zwischen den Rechnern, beispielsweise in der verwendeten Prozessorarchitektur. Dadurch kann der Cluster zu einem späteren Zeitpunkt auf einfache Weise ausgebaut werden. Oftmals sind die neuen Rechner sogar vom Typ her vergleichbar, aber viel schneller, wodurch ein Teil der älteren Hardware entbehrlich geworden ist. Auf diese Weise hält sich trotz eines Ausbaus der Gesamtstromverbrauch und der Kühlaufwand in Grenzen.
- Beim Ausfall eines Knotens ist dieser leicht und schnell zu ersetzen.
- Die Planungszeit ist in Bezug auf den Hardwarekauf verkürzt. Die Rechner können erst kurz vor der eigentlichen Verwendung gekauft werden. Dies ist deswegen nützlich, da die Rechenkraft ständig steigt und man kurz vor dem Einsatzbeginn die maximale Rechenkraft für das aufgewendete Geldvolumen erhält.

Es gibt jedoch auch spezielle Problemstellungen, bei denen ein großer Datenaustausch zwischen den einzelnen Rechenkomponenten unvermeidbar ist, weil die Aufgaben schlecht verteilbar sind. In diesem Fall kann auf klassische Großrechner nicht verzichtet werden und für solche Anwendungen sind Cluster aus Standardrechnern nicht geeignet.

Bei dem Aufbau und Betrieb eines solchen Clusters ergeben sich neue logistische und administrative Schwierigkeiten.

Neben dem eigentlichen Aufbau stellt sich zunächst die Frage, wie das Betriebssystem und die notwendige Software auf die Rechner verteilt werden soll. Hier möchte man möglichst eine automatische Lösung, da es viel Zeit in Anspruch nimmt, jeden Rechner von Hand zu konfigurieren und zu installieren. Idealerweise sollte eine Referenzinstallation einmal gemacht werden und sich dann problemlos auf die restlichen Knoten übertragen lassen. Die weitere Administration sollte von einem zentralen Ort aus möglich sein. Auch eine spätere mögliche Änderung der Konfiguration oder des Betriebssystemes sollte problemlos auf allen Rechnern möglich sein, ebenfalls von einem zentralen Arbeitsplatz aus.

Ein weiteres Problem ist die Zuverlässigkeit des Clusters. Beim Betrieb von Supercomputern garantieren in der Regel die Hersteller eine totale Verfügbarkeit. Man spricht hier von *24/7* und meint die ganze Woche rund um die Uhr. Dies wird meist durch spezielle, im Betriebssystem eingebaute, redundante Systeme erreicht, die für ausgefallene Komponenten einspringen können.

Ein Cluster aus Standardkomponenten besteht aus sehr viel mehr eigenständigen Systemen, die diese Verfügbarkeit von Haus aus nicht mitbringen. Sie wurden auf ein gutes Preis/Leistungsverhältnis optimiert und der Mehraufwand für Ausfallsicherheit rechnet sich bei diesen Komponenten in der Regel nicht für die Hersteller. Kommt es zu einem Ausfall eines Knotens, dann sollte idealerweise sofort ein anderer Rechner dessen Aufgaben übernehmen können. Zu diesem Zweck sind Überwachungsmöglichkeiten erforderlich, die Ausfälle von wichtigen Komponenten wie beispielsweise eines Lüfters erkennen können und entsprechend reagieren. Diese Funktionen müssen dem Cluster gesondert hinzugefügt werden. Die Fehlertoleranz eines solchen Grids, oder vielmehr jedes einzelnen Rechners, ist zu einem wichtigen Aspekt geworden.

Ein weiterer Punkt, an dem Kosten gespart werden können, ist die überflüssige Hardware, wie Floppy oder Grafikkarte, die in der Regel nicht oder nur einmal benötigt werden. Sie sind jedoch bei einigen Systemen für diese erste Installation erforderlich, die Möglichkeiten, dies durch eine Unterstützung des BIOSses zu umgehen, sind bisher begrenzt.

Aus diesen Gründen folgt, daß eine vollständige Kontrolle über einen Rechner aus der Ferne wünschenswert wäre. Eine entsprechende Lösung könnte einmalig in Form einer Steckkarte in den Rechner eingebracht werden und weitere Bauteile wie Grafikkarte und Floppy überflüssig machen. Idealerweise sollte ein bestückter Rechner von diesem Zeitpunkt an vollständig über ein Netzwerk auf korrekte Funktion überwachbar und steuerbar sein, wenn möglich von einem zentralen Ort aus durch einen einzigen Administrator. Die Aufstellung der Rechner des Clusters könnte dann durch einfache Angestellte abgewickelt werden. Ein wichtiger Punkt bei der Entwicklung wäre eine möglichst kostengünstige Lösung, bei welcher der Preis der Karte nur einen kleinen Teil des Preises des bestückten Rechners ausmacht.

An dieser Stelle soll noch auf ein weiteres Einsatzgebiet solch einer Lösung eingegangen werden, das besonders bei großen Unternehmen eine Rolle spielt. In verschiedenen Bereichen werden verteilte Rechner eingesetzt, die regelmäßig gewartet werden müssen. Diese Rechner sind teilweise schwer oder nur umständlich zugänglich, wenn sie beispielsweise über eine große Entfernung verteilt sind. Jede nötige Administration vor Ort kann durch die notwendige Anfahrt teuer und langwierig sein. In der Regel handelt es sich aber auch bei diesen Rechnern um Standardkomponenten. Auch in diesem Falle wäre eine Möglichkeit wünschenswert, über eine Netzwerkverbindung die korrekte Funktion des Rechners zu kontrollieren und den Betrieb zu steuern.

Die geschilderten Situationen lassen die Notwenigkeit einer Lösung zur Steuerung und zur Kontrolle eines Rechners aus der Ferne über ein Netzwerk aufkommen. Ein Administrator sollte einen einmal bestückten Rechner nicht mehr vor Ort bedienen müssen. Schon jetzt wird klar, daß diese Lösung vom eigentlichen Betriebszustand unabhängig sein muß und damit eine Implementierung in Hardware nach sich zieht, am besten durch die Ausführung als steckbare Erweiterungskarte in einen Rechner.

Die Konstruktion des Prototypen einer solchen Steuerkarte ist der Inhalt dieser Arbeit.

Die Diplomarbeit wurde in Kooperation mit dem Diplomanden *Martin Kirsch* ausgeführt. Einige Teile wurden aus diesem Grund innerhalb seiner Diplomarbeit behandelt. Im Text wird an entsprechender Stelle darauf hingewiesen.

Kapitel 2

Konzept und Lösungsansatz

In der Einleitung wurden administrative Probleme erläutert, die sich beim Betrieb von Clustern oder schwer zugänglichen Rechnern stellen. Im ersten Teil dieses Kapitels werden die konkreten Anforderungen an eine Lösung zusammengestellt. Im Zweiten Teil des Kapitels wird der in der Diplomarbeit verwendete Lösungsansatz erläutert und das Konzept der Arbeit vorgestellt. Im folgenden Kapitel wird auf die konkrete Auswahl der Komponenten eingegangen.

2.1 Zusammenstellung der Anforderungen

Die in der Einleitung erläuterten Probleme bei der Administration von Computer-Clustern oder bei der Verwendung von weit entfernten Rechnern können auf Probleme mit der Administration einzelner Rechner zurückgeführt werden.

Dabei ergibt sich als Hauptanforderung, die vollständige Kontrolle über einen Rechner nur über ein Netzwerk zu ermöglichen. Der Begriff *vollständig* ist so zu verstehen, daß ein einmal entsprechend ausgestatteter Rechner im weiteren Verlauf nicht mehr vor Ort administriert werden muß. Weitere Schritte, wie die Installation eines Betriebssystemes, sollen über die Netzverbindung erledigt werden. Die Kontrolle gliedert sich in die folgende Bereiche:

- Die Kontrolle über die Stromversorgung und den Betriebszustand des Rechners. Dies bezeichnet die Möglichkeit, den Rechner im Fehlerfalle aus- und wieder einzuschalten sowie bei Bedarf zurückzusetzen und setzt eine separate Stromversorgung für den Steuerteil voraus.
- Die Möglichkeit, den Rechner über ein Netzwerk zu bedienen und zu administrieren. Moderne Betriebssysteme verfügen bereits über Möglichkeiten, mit einem Rechner über ein Netzwerk zu arbeiten. Dies setzt jedoch die bereits erfolgte Installation eines solchen Betriebssystemes voraus. Die gewünschte Kontrolle bezieht sich daher besonders auf Bereiche *vor* der Installation eines solchen Betriebssystemes oder im Fehlerfalle. Beispielsweise soll ein BIOS-Setup über das Netzwerk möglich sein oder Einfluß auf den Startvorgang des Rechners genommen werden können.
- Die Möglichkeiten zur Fehlersuche und Diagnose des Rechners, ohne das Vorhandensein eines Betriebssystemes vorauszusetzen. Eine Diagnosemöglichkeit auf Hardwareebene wäre

wünschenswert, so daß im Idealfall auch kritische Werte wie Temperatur des Prozessors, Funktion von Lüftern oder die Stromversorgung überwacht werden können.

Für die Verwendung in Clustern ergeben sich weitere Anforderungen, da eine große Anzahl von Rechnern zum Einsatz kommen, welche aus Kostengründen erst kurz vor der Aufstellung gekauft werden und deren Hardwarekonfiguration nicht notwendigerweise bekannt ist:

- Angestrebt wird eine möglichst universelle Lösung, welche zum Einsatz mit vielen verschiedenen Rechnerkonfigurationen geeignet sein soll.
- Möglichst kostengünstige Lösung.

2.2 Vergleich mit verfügbaren Lösungen

In der Regel wird für den Aufbau eines Clusters oder für den entfernten Einsatz eines Rechners ein Betriebssystem gewählt, welches sich weitgehend über das Netz bedienen und administrieren läßt. Als kostenloses und trotzdem leistungsfähiges Beispiel sei hier das System *Linux* genannt. Doch wie oben bereits erläutert stellt bei einer großen Anzahl von Rechnern bereits die Installation des Systems auf den einzelnen Rechnern ein erstes administratives Problem dar.

Es gibt heute bereits Hauptplatinen, die über einige Fähigkeiten verfügen, dies zu automatisieren. Beispielsweise kann der Rechner mit Hilfe der Technik *Wake-On-LAN* über einen Netzwerkzugriff eingeschaltet werden. Weiterhin verfügen einige Hauptplatinen über eine Implementierung der Protokolle *BOOTP* [Bo85] zusammen mit *DHCP* [Dh93], wodurch sich der Startprozess weiter steuern läßt.

Alles in allem bieten sich damit aber nur eingeschränkte Möglichkeiten. Sie beschränken sich im Wesentlichen auf die Vergabe einer IP-Adresse und das Laden einer auszuführenden Datei. Die Möglichkeit ein BIOS-Setup durchzuführen oder den Rechner im Fehlerfalle kontrolliert aus- und wieder einschalten zu können läßt eine solche Lösung leider vermissen. Überwachungsmöglichkeiten von Hardwarekomponenten wie Stromversorgung oder Lüfterzustand, welche unabhängig vom Betriebssystem über das Netz angeboten werden, fehlen meist völlig. Hinzu kommt, daß man sich mit diesen Hauptplatinen auf eine spezielle Hardware festlegt, die bereits in die gehobene Preisklasse fällt.

Weiterhin gibt es bereits eine vergleichbare Lösung der Firma *AMI* namens *MegaRac* [Ami]. Diese Karte verfügt jedoch über viel weitreichendere Möglichkeiten und ist sehr aufwendig hergestellt, wodurch sich ein Preis von über DM 1000 ergibt (Stand: Sept. 2001). Die Karte ist demnach für ein anderes Einsatzgebiet vorgesehen und kann aufgrund des hohen Preises nicht verwendet werden. In den Anforderungen der Arbeit wurde Wert darauf gelegt, daß der Preis der Steuerkarte nicht wesentlich zum Preis des bestückten Rechners beiträgt.

Insgesamt läßt sich sagen, daß keine kommerziellen Produkte zu Verfügung standen, welche obige Anforderungen erfüllen konnten. In der vorliegenden Diplomarbeit wurde mit der eigenen Konzeption einer Lösung begonnen und ein erster Prototyp erstellt.

Im folgenden werden die einzelnen Grundgedanken an das Konzept ausgeführt. Am Ende des Kapitels wird eine zusammenfassende Darstellung gegeben:

2.3 PCI Karte als Träger

Ein Großteil der Anforderungen konnte nicht alleine mit spezieller Software erfüllt werden. Schon um zu jedem Zeitpunkt die Kontrolle über einen Rechner zu haben und um weitere Funktionen wie Überwachung der Hardware zu verfügen, kam nur eine Hardwarelösung in Frage.

In der Diplomarbeit wurde sich zur Entwicklung eines Prototypen in Form einer steckbaren Erweiterungskarte entschieden. Auf dieser Karte sollten alle weiteren Komponenten untergebracht werden. Als geeigneter Steckplatz kam dabei nur PCI in Frage. Zunächst soll kurz auf die Vorteile vom PCI Bus eingegangen werden.

Der PCI Bus

Der PCI Bus (engl: *periphal component interconnect*) ist ein Steckbus für Erweiterungskarten und wurde konstruiert, um jede Art von Hardwareerweiterungen wie Grafikkarten, Soundkarten etc. aufzunehmen, welche intern im Rechner plaziert werden sollen. Er hat aufgrund seiner Stabilität und hohen Geschwindigkeit bereits kurz nach der Einführung seine Vorgänger ISA, EISA, MCA, VESA etc. vom Markt verdrängt. Die heute üblichen Hauptplatinen verfügen meist über mehrere PCI Steckplätze, welche mit verschiedenen Karten bestückt werden. Alle Karten sind über den Bus und den Chipsatz des Rechners mit dem Hauptprozessor und dem Hauptspeicher verbunden.

Momentan ist der PCI Bus der universellste Erweiterungsbus für Hardwarekomponenten und wird in allen Rechner verwendet, welche durch ihr Preis-Leistungsverhältnis für den Einsatz in Clustern oder als entfernte Steuerungsrechner in Frage kommen. Aufgrund seiner Vielseitigkeit, der weiten Verbreitung und dabei hohen maximalen Datentransferrate von bis zu 528 Mbyte/Sek (64 Bit, 66 MHz) ist davon auszugehen, daß er auch in zukünftigen Rechnern noch verwendet wird. Durch die Wahl von PCI läßt sich die erstellte Karte damit großflächig ohne Probleme einsetzen und stellt eine universelle und zukunftssichere Lösung dar.

Über den PCI Stecker konnten die einzelnen Komponenten des Prototypen zunächst mit Strom versorgt werden. Zusätzlich besteht die eigentliche Aufgabe des PCI Busses jedoch darin, weitere Geräte mit dem Prozessor und Hauptspeicher zu verbinden. Damit ergab sich die Möglichkeit, auf Teile der Hardware des Rechners zuzugreifen und über das Netzwerk einem entfernten Administrator Auskünfte über die vorhandene Hardware zu liefern. Weiterhin besteht über den PCI-Bus der Zugriff auf den Hauptspeicher und den I/O-Speicherbereich des Rechners. Der Hauptzweck sollte aber die Emulation einer VGA-Karte werden, um eine Möglichkeit zu haben, über den Inhalt des Textbildschirmes des Rechners zu verfügen und diesen ebenfalls über das Netzwerk übermitteln zu können. In der Arbeit wird an verschiedenen Stellen weiter auf den Bus eingegangen, insbesondere in Abschnitt 3.2 und 4.4.

2.4 Separate CPU

Um die gesamten anfallenden Aufgaben ohne eine CPU zu koordinieren, wäre eine erhebliche Menge an Elektronik erforderlich gewesen. Aus diesem Grund wurde eine eigene CPU auf der Karte integriert. Sie sollte sowohl die Verbindung nach Außen über ein Netzwerk, als auch die Steuerung

des Rechners übernehmen und die zu überwachenden Betriebszustände sammeln. Bei Bedarf werden die Daten über das Netz übertragen.

Ein weiterer wichtiger Vorteil war, daß durch die Verwendung einer eigenen CPU eine gewisse Flexibilität in der Gestaltung der Karte bestand, da das auf der CPU laufende Programm nach Belieben den aktuellen Anforderungen angepaßt werden konnte.

Die Verwendung einer CPU erforderte auch das für deren Betrieb nötige RAM und ein programmierbares ROM, um die auszuführenden Programme dort speichern zu können.

Stromversorgung der CPU

Insbesondere sollte ein mit dieser Karte bestückter Rechner auch bei ausgeschaltetem Zustand über das Netzwerk reaktiviert werden können. Deshalb mußte die CPU und das Netzwerk zusätzlich über eine eigene Stromversorgung verfügen.

Um die Stromversorgung zu gewährleisten, sollte eine entsprechende Notversorgung per Akku genügen. Der Akku wird im Betrieb ständig geladen und stellt die notwendige Spannung zur Verfügung. Möglicherweise könnte auch eine alternative Versorgung per externem Netzteil hinzugefügt werden, um den Rechner auch längere Zeit ausgeschaltet zu lassen, beispielsweise für den Einsatz als einzel entfernter Steuerungsrechner.

2.5 Netzwerk und Steuerung

Ethernet

Um von einem zentralen Punkt aus die Steuerungsfunktionen der Karte nutzen zu können, mußte die CPU über eine Netzwerkverbindung verfügen. Hier bot sich besonders Ethernet an, da es sehr weit verbreitet, relativ günstig und universell einsetzbar ist. Ethernet hat im Vergleich zu diversen Industriestandards wie CAN zudem den Vorteil, daß entsprechende Steuerkarten für jeden PC oder Laptop billig zu erwerben sind und gleichzeitig relativ hohe Geschwindigkeiten von 10, 100 oder 1000 Mbit/Sek mit gewöhnlichem STP (*shielded twistet pair*) Kabel möglich ist.

Um Ethernet einsetzen zu können, würde zusätzlich ein spezieller Netzwerkchip benötigt werden. Diese Chips sind ebenfalls preiswert und leicht zu erwerben.

An dieser Stelle wird darauf hingewiesen, daß die Netzwerkverbindung der Steuerkarte nichts mit der Netzwerkverbindung des bestückten Rechners gemein hat. Beide Verbindungen können jedoch an dasselbe Netzwerk angeschlossen sein, falls es sich um den selben Typ handelt.

Weitere Steuerzugänge

Um die Karte nicht nur über die Ethernet-Netzwerkverbindung steuern zu können, sollten weitere lokale Steuerzugänge vorhanden sein. Zu diesem Zweck sollten übliche Verbindungen herangezogen werden, wie sie häufig in Labors eingesetzt oder von tragbaren Rechnern verwendet werden.

Möglich wäre die Verwendung von serieller Verbindung, CAN-Bus oder die speziellen Anschlüsse an verbreiteten Handhelds wie *Palm* etc. gewesen. Hier könnte z.B. die Infrarotschnittstelle *IrDA*

verwendet werden. Dieser Punkt sollte am Schluß der Arbeit als mögliche Erweiterung in Angriff genommen werden.

2.6 Kontrolle des Betriebszustandes

Um ihren Zweck als Fernwartungskarte zu erfüllen, sollte die im Projekt realisierte Karte auch die Möglichkeit haben, den bestückten Rechner falls nötig neu zu starten oder ihn aus- und wieder einzuschalten. Normalerweise greift man zu diesem Zweck auf die Funktionen des installierten Betriebssystems zurück, welches einen kontrollierten Neustart veranlaßt, bei dem Dinge wie die Integrität des Dateisystemes beachtet werden. Aus dem täglichen Umgang mit Rechnern hat sich jedoch gezeigt, daß es von Zeit zu Zeit nötig ist, dies am Betriebssystem vorbei zu erzwingen.

Um alles möglichst einfach zu halten, sollte die Karte gedrückte Gehäusetaster des Rechners simulieren. Zusätzlich sollte eine Möglichkeit geschaffen werden, den momentanen Betriebszustand abzufragen.

2.7 Steuerung durch Emulation der Peripherie

Um einen Rechner steuern zu können, sollten die grundlegenden Peripheriegeräte emuliert werden. Wenn es mit diesen Mitteln möglich wäre ein Betriebssystem zu laden, so kann zum weiteren Betrieb auf die höheren Funktionen des Betriebssystems zurückgegriffen werden. Da die Karte auch für den Einsatz auf weiter entfernten Rechnern konzipiert war, werden die verwendeten Betriebssysteme solche Funktionen auch über das Netzwerk bereitstellen können.

Emulation von Eingabegeräten

Ein wichtiger Teil des Projektes war die Forderung, den mit der Karte bestückten Rechner über das Netzwerk fernsteuern zu können. Zu diesem Zweck mußte sie in der Lage sein, Eingaben von Maus oder Tastatur entweder auf Software- oder auf Hardwareebene zu emulieren und zumindest die Ausgaben des Textbildschirmes an den entfernten Rechner eines Administrators zu übertragen. Da die Steuerung auch den Zeitpunkt vor der Installation eines Betriebssystemes abdecken sollte, blieb nur eine Hardwarelösung übrig.

Um möglichst universell einsetzbar zu sein wurde sich dazu entschlossen, die Eingabegeräte Maus und Tastatur direkt als extern vorhandene Geräte zu emulieren. Auf der Karte befinden sich dann spezielle Ausgänge für die emulierten Geräte, welche über die üblichen Standardstecker an die üblichen Eingänge des Rechners angeschlossen werden. Möglicherweise läßt sich zu einem späteren Zeitpunkt die Emulation vereinfachen, indem direkt über den PCI Bus eine Erweiterungskarte mit angeschlossener Tastatur oder Maus emuliert wird. Aus diese Weise könnte dann die Verkabelung weggelassen werden.

Für den Betrieb von Eingabegeräten werden verschiedene Anschlüsse verwendet. In Frage kamen hier die seriellen Anschlüsse des Rechners (*COM*), sowie die üblicherweise verwendeten PS/2 Schnittstellen als auch der relativ neue Anschluß USB, welche leistungsfähiger und vielseitiger ist, aber einiges mehr an Aufwand erfordern würde.

Emulation einer Floppy

Viele der heutigen Rechner lassen sich so konfigurieren, daß sie nach dem Systemtest spezielle Startdateien von einer Diskette laden und ausführen. Auf diese Weise bietet sich eine Möglichkeit, bereits frühzeitig lenkend in den Startprozess des Rechners eingreifen zu können, um beispielsweise ein neues Betriebssystem zu installieren oder unterschiedliche Softwarekonfigurationen zu starten. Die begrenzte Größe einer Diskette reicht dabei aus, um auf ihr die notwendigen Programme unterzubringen, welche das Starten weiterer Software von Festplatte, CD oder über das Netzwerk veranlassen.

Aus diesem Grund wurde zusätzlich auf die gleiche Weise wie bei der Emulation der Eingabegeräte eine vorhandene Floppy emuliert, über welche eine im Laufwerk einliegende Diskette mit beliebigem Inhalt vorgetäuscht werden kann. Die Dateien auf der Diskette können speziell an die Hardwarekonfiguration oder bestimmte Aufgaben des Rechners angepaßt und jederzeit über das Netzwerk geändert werden. Mit dieser Maßnahme wurde eine einfache Installation oder Aktualisierung des auf dem Rechner vorhandenen Betriebssystems nur über das Netzwerk erst möglich.

Emulation einer VGA Grafikkarte

Da die Karte einen Rechner über das Netzwerk steuern können sollte, mußte zusätzlich zur Eingabe auch die Ausgabe von Programmen an einen entfernten Administrator übermittelt werden können. Wie schon bei der Eingabe stellte sich auch hier das Problem, daß keine Funktionen von Betriebssystemen genutzt werden konnten, da die Karte insbesondere zur Steuerung des Rechners *vor* der Installation eines solchen verwendet werden sollte, um beispielsweise auf dem Bildschirm ausgegebene Fehlermeldungen während des Startvorganges zu übermitteln. Daher mußte auch in diesem Falle die entsprechende Hardwarekomponente selbst emuliert werden.

Ausgegebene Informationen werden im Text- oder Grafikspeicher der installierten Grafikkarte gespeichert. Ein weiteres Ziel der Arbeit bestand also in der erfolgreichen Emulation einer im PCI Bus eingesteckten Grafikkarte. Dadurch könnte die Karte selbst die anfallenden Grafikausgaben entgegennehmen und speichern. Bei Bedarf könnte der Bildschirminhalt abgerufen und über das Netzwerk an einen Administrators versendet werden, wo er sich diese anzeigen lassen könnte.

Der am weitesten verbreitete Standard in diesem Bereich ist VGA, der von allen heutigen Rechnern unterstützt wird. Auch der PCI Bus kann mit VGA-Karten bestückt werden, und es ist abzusehen, daß auch künftige Rechner, welche in Clustern oder als entfernte Steuerungsrechner eingesetzt werden, mit Standard-VGA Karten betrieben werden können. Selbst heutige bessere Grafikkarten sind immer noch zu dem etablierten Standard kompatibel und starten ihre Anzeige im VGA-Modus. Sie werden erst durch spezielle Funktionen innerhalb ihrer Treiber in leistungsfähigere Modi umgeschaltet. Aus diesem Grund steuern viele Betriebssysteme eine ihnen unbekannte Grafikkarte zunächst im VGA-Modus an, und auch das BIOS der Rechner startet diese gewöhnlich im VGA-Modus. Eine VGA Karte stellt damit eine universelle, auf unterschiedlichen Plattformen einsetzbare Grafikkarte dar.

Aufgrund dieser Eigenschaften war die erfolgreiche Emulation der Minimalversion einer VGA-Karte ein wichtiger Anforderungspunkt der Arbeit, wobei zunächst die korrekte Arbeit im Textmodus gelöst werden sollte. Im Falle des Erfolgs würde damit sogar die Verwendung einer zusätzlichen Grafikkarte überflüssig werden und die Gesamtkosten reduzieren.

Auf die in diesem Zusammenhang auftretenden speziellen Schwierigkeiten wird an entsprechender Stelle in der Arbeit näher eingegangen.

2.8 FPGA zur Aufnahme der Logik

Die vielfältigen Aufgaben wie Steuerung durch Emulation von Peripherie und Überwachung kritischer Komponenten hätten sehr viel Logik auf der Karte erfordert, welche mit konventioneller Elektronik nicht zu realisieren gewesen wäre. Da viele verschiedene Aufgaben parallel zu erledigen waren, schied die Verwendung der CPU für diese Aufgaben von vornherein aus und es zeigte sich bereits früh, daß aus diesem Grund ein Mikrochip auf der Karte erforderlich sein würde.

Eine Möglichkeit wäre gewesen, einen eigenen Mikrochip zu entwickeln, diese bezeichnet man auch als *ASIC (application specific integrated circuit*). Die Herstellung eines solchen Chips ist jedoch ein teurer, aufwendiger und vor allem irreversibler Prozeß und daher für die Entwicklungsphase nicht geeignet. In der vorliegenden Arbeit wurde daher eine vergleichbare aber programmierbare Alternative gewählt, ein sogenannter *FPGA* (*field programmable gate array*).

Im Folgenden wird kurz auf den Aufbau und die Eigenschaften eines FPGAs eingegangen.

FPGAs

Bei FPGAs handelt es sich prinzipiell um elektrisch frei programmierbare Logikbausteine hoher Komplexität [Si00] . Sie besitzen folgenden Aufbau:

- Eine große Anzahl von Logikzellen, welche zumeist in einer quadratischen Matrix angeordnet sind. In jede einzelne Zelle kann bis zu einer gewissen Komplexität eine frei programmierbare logische Funktion abgebildet werden, beispielsweise Register, FIFOs oder Produktterme. Bei einigen FPGAs gibt es weiterhin Bereiche mit speziellen Zellen, welche besonders platzsparend auf bestimmte Funktionen wie RAM oder Look-Up-Funktionen abgebildet werden können.
- Spezielle I/O-Zellen für die Ein- und Ausgabe, welche mit den Anschlußbeinchen des Logikbausteines verbunden sind. Sie verfügen über spezielle Treiber und Schutzfunktionen und lassen sich explizit als Eingang, Ausgang oder Tristate (hochohmig) programmieren. Der aktuelle logische Zustand wird in einem Register gespeichert.
- Ein bestehendes elektrisches Verbindungsnetzwerk, an welchem alle Logikzellen und I/O-Zellen angeschlossen sind. Durch die Programmierung können einzelne Verbindungen von Logikzellen geschaltet oder unterbrochen werden. Auf diese Weise kann eine nahezu beliebige Verbindung der Zellen untereinander erreicht werden, die jedoch durch die Anzahl der verfügbaren Leitungen begrenzt ist. Innerhalb des Verbindungsnetzwerkes gibt es meist noch speziellere Leitungen, die zur Verwendung eines globalen Taktsignales verwendet werden. Durch ihren Aufbau kommt das Signal an jeder Logikzelle nahezu zur gleichen Zeit an.
- Zusätzliche existiert weitere Logik welche die Aufgabe der Programmierung des FPGAs übernimmt und nicht Teil der programmierten logischen Gesamtfunktion ist. Die Programmierung der Logikzellen und des Verbindungsnetzwerks wird in SRAM-Zellen

gespeichert, so daß sie sich jederzeit ändern läßt. Leider ist die Programmierung in solchen SRAM-basierten FPGAs jedoch flüchtig, wenn er nicht mehr mit Strom versorgt wird und so gibt es auch einmalig permanent beschreibbare Ausführungen.

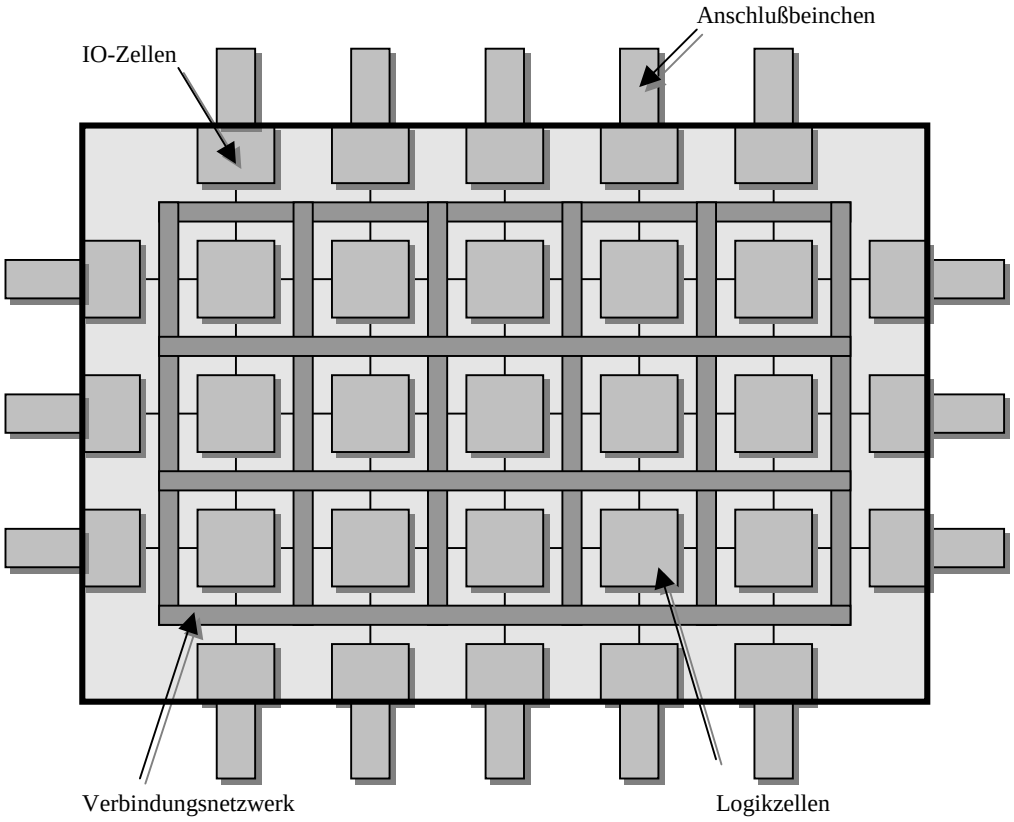


Abbildung 1: Schematische Darstellung des Aufbaus eines FPGAs.

Bei FPGAs achtet man auf eine großen Anzahl von Logikzellen und frei nutzbaren I/O-Zellen. Die logische Größe und Leistungsfähigkeit wird dabei meist in sogenannten Gatter-Äquivalenten angegeben, welche die zur Verfügung stehende Komplexität mit einem Aufbau aus reinen NAND-Gattern vergleichen soll. FPGAs gibt es in unterschiedlichen Bauformen und von verschiedenen Herstellern wie Altera, Xilinx, Actel, Lucent etc.

Der große Vorteil von FPGAs liegt in der Möglichkeit, sie immer wieder mit nahezu beliebigen Schaltungen konfigurieren zu können. Trotz dieser Vielseitigkeit ist ein FPGA letztlich ein Mikrochip und erreicht Schaltgeschwindigkeiten von bis zu 200 MHz und mehr. Durch die freie Programmierbarkeit bieten FPGAs aber weniger Platz für Logik und sind bei großen Stückzahlen deutlich teurer als ASICs. In der Regel werden deshalb FPGAs in der Entwicklungsphase verwendet, um bei größeren Stückzahlen zur Produktion von ASICs überzugehen.

Normalerweise plaziert man neben einem FPGA noch ein wiederbeschreibbares EEPROM, in welchem der Bitstrom einer Grundkonfiguration des FPGAs gespeichert wird. Diese geht dann auch nach Wegnahme der Stromversorgung nicht verloren. Viele FPGAs sind in der Lage, beim Einsetzen der Stromzufuhr diese Grundkonfiguration aus dem EEPROM zu laden und sich mit ihr zu initialisieren.

2.9 Sensoren

Ein wichtiger Punkt bei der Konzeption der Karte sollte die Möglichkeit sein, kritische Hardwarekomponenten des bestückten Rechners zu überwachen um bei einem drohenden Ausfall rechtzeitig eine Warnung abzugeben. Bei vielen Rechnern verfügt bereits das BIOS über solche Überwachungsfunktionen. Die Meldungen im Fehlerfalle beschränken sich aber meist auf akustische Signale oder die Möglichkeit, den Rechner auszuschalten. Meldungen über ein Netzwerk sind meist nicht möglich, ohne das BIOS des Rechners auszutauschen.

Mit einer zusätzlichen Überwachung durch die Karte wäre im Fehlerfalle eine Meldung über das an die Karte angebundene Netzwerk ohne weiteres möglich. Als mögliche zu überwachende Komponenten kamen in Frage:

- Temperatur kritischer Bauteile um eine Überhitzung zu erkennen.
- Vorhandene Lüfter, insbesondere der CPU. Diese verfügen meist schon an ihren Steckern über entsprechende Kontrollsignale, welche normalerweise mit Steckplätzen auf der Hauptplatine verbunden und vom BIOS des Rechners kontrolliert werden. Auf der Karte könnten ähnliche Steckplätze angebracht werden.
- Korrekter Zustand der Spannungsversorgung von kritischen Komponenten um beispielsweise eine Veränderung bei Überlastung rechtzeitig zu erkennen. Diese müßte an entsprechenden Stellen auf der Hauptplatine abgegriffen und mit den Normalwerten verglichen werden.
- Möglich wäre auch der Einbau eines Mikrophons in Kombination mit einem ADC, da sich im täglichen Umgang mit Festplatten gezeigt hat, daß diese teilweise ihr Laufgeräusch über eine längere Zeit ändern, bevor sie ausfallen.
- Mögliche weitere Sensoren, je nach Einsatzgebiet des bestückten Rechners.

Die Sensoren könnten zusätzlich in den Rechner eingebracht werden. Falls es gelänge, die intern bereits vorhandenen Sensoren beispielsweise über den PCI Bus anzusteuern, sollten diese verwendet werden.

2.10 Diagnose über PCI Bus

Da die zu konstruierende Karte als PCI-Erweiterungskarte ausgelegt wurde, sollte, um zusätzliche Hardware einzusparen, über den PCI Bus auf die internen Sensoren zugegriffen werden. Dies sollte zumindest teilweise möglich sein, weil das BIOS der Rechner ebenfalls auf diese Informationen zurückgreifen und dabei die vorhandene Hardware nutzen muß.

Üblicherweise werden diese Werte über die sogenannten *Southbridge*-Chipsätze der Hauptplatine über den an den PCI Bus angeschlossenen ISA- bzw. SM-Bus (*system management bus*) und/oder den I^2C-Bus (*I-Quadrat-C* Bus) ausgelesen. Für das Betriebssystem Linux gibt es beispielsweise schon die Softwarepakete *i^2c* und *lm_sensors*, welche diese Werte auslesen können [LinB], [LmS].

BIOS Statusnummern

Die Fähigkeit, auf den PCI Bus zuzugreifen bot weitere Möglichkeiten zur Fehlerdiagnose, nämlich die Überprüfung von sogenannten *POST*-Statusnummern.

Schon seit den ersten PCs existiert ein Mechanismus zur Fehlersuche auf Hardwarebene, mit dem sich Probleme zu einzelnen Hardwarebereichen in einem Rechner ermitteln lassen. Aufgrund seiner einfachen Implementierung und seiner leichten Durchführbarkeit wird er auch heute noch verwendet.

Während des Systemstartes eines Rechners führt das BIOS einen *POST* genannten Selbsttest aus (engl: *power on self test*). In dieser Zeit wird das Vorhandensein und die korrekte Funktion einzelner Hardwarekomponenten, angefangen von CPU, über Speicher, Chipsatz, Erweiterungskarten, Festplatten etc. geprüft. Das Problem ist, daß bei Fehlern in wichtigen Komponenten das BIOS oft nicht mehr in der Lage ist, eine korrekte Meldung auf dem Bildschirm auszugeben, sondern allenfalls akustische Signale über den eingebauten Lautsprecher abgeben kann. Es gibt jedoch eine weitere Möglichkeit, Fehler zu melden, die lediglich eine CPU mit Basisfunktionen voraussetzt.

Die Überprüfung der Hardware erfolgt in mehreren Abschnitten, welche den vorhandenen Komponenten zugeordnet sind. Jeder Abschnitt trägt dabei eine eigene interne Nummer. Jedesmal, wenn das BIOS zu einem neuen Testbereich übergeht, wird dessen Nummer mit einem I/O-Befehl der CPU auf eine bestimmte Adresse, meist 0x80 geschrieben. Der Schreibbefehl erscheint dadurch auf dem Datenbus der Hauptplatine des Rechners und weiter auch auf den dort angeschlossenen Bussen für Steckkarten wie ISA oder PCI. Eine Karte in einem solchen Bus kann nun diesen auf Schreibbefehle an die entsprechende I/O-Adresse überprüfen und die an diese Adresse geschriebenen Nummern mitlesen und speichern. Bei einem Fehler während des Selbsttestes wird die CPU vom BIOS angehalten und die fehlerhafte Hardwarekomponente ist anhand der letzten gelesenen Nummer zu identifizieren.

Das Besondere ist die Tatsache, daß das BIOS des Rechners, welches speziell auf ihn abgestimmt ist, bereits die einzelnen Komponenten testet. Dadurch ist es nur durch das rein passive Mitlesen der Fehlernummern möglich, mit relativ geringem Aufwand Probleme mit der eingebauten Hardware zu erkennen. Die Nummern müssen nach dem Auslesen in einer Liste des BIOS-Herstellers auf ihre Bedeutung hin nachgeschlagen werden.

PCI Scan

Schließlich wurde noch eine weitere Möglichkeit bedacht, um den PCI Bus zu Diagnosezwecken zu nutzen: Fast sämtliche Geräte in einem Rechner, welche durch Software angesprochen werden müssen, werden zu diesem Zweck durch ein PCI-Gerät auf dem Bus repräsentiert. Bei den Geräten kann es sich sowohl um eine tatsächlich eingesteckte Karte, als auch um eine Komponente auf der Hauptplatine des Rechners handeln.

Durch die systematische Absuche des Busses nach Geräten kann somit leicht eine Übersicht über die im Rechner eingebaute Hardware gewonnen werden. Da die zu entwickelnde Karte auch zur Installation eines Betriebssystemes genutzt werden sollte, wäre es mit diesem Mechanismus beispielsweise möglich, automatisch das richtig konfigurierte System auszuwählen

Aus diesem Grund sollte die Möglichkeit, die vorhandenen Geräte des PCI Busses zu ermitteln, mit implementiert werden. (Fähigkeit, als PCI-*Master* zu arbeiten, siehe Abschnitt 4.4)

2.11 Zusammenfassung des Konzeptes

Ziel der Diplomarbeit war die Konstruktion einer Hardwarekarte, mit der sich entfernte Rechner nach einmaliger Bestückung über ein Netzwerk steuern und kontrollieren lassen. Das Einsatzgebiet sind Computer-Cluster aus vielen Rechnern oder einzelne aber schwer zugängliche Rechner. Die Karte sollte möglichst universell einsetzbar und preislich nicht zu teuer sein.

Im Folgenden wird nun das Konzept für die Entwicklung der Karte noch einmal zusammengefaßt. Der bestückte Rechner wird im folgenden als *Wirtsrechner* bezeichnet:

- Als Träger wird eine PCI Steckkarte verwendet, um in möglichst vielen verschiedenen Rechnern einsatzfähig zu sein.
- Zur Koordination wird ein Prozessor aus dem Bereich der embedded CPUs verwendet, der Anweisungen über ein Netzwerk von einem Steuerungsrechner erhält. Ein lokaler Zugang zur CPU macht auch eine Arbeit vor Ort möglich. Ein Akku lädt sich bei Betrieb auf und hält den Prozessor auch bei ausgeschaltetem Wirtsrechner einige Zeit am Laufen. Über die CPU kann der Rechner ausgeschaltet, eingeschaltet oder zurückgesetzt werden.
- Zur Steuerung des Wirtsrechners werden die Peripheriegeräte Maus, Tastatur und Floppy auf Hardwareebene emuliert und an die entsprechenden Eingänge des Rechners angeschlossen. Diese werden zunächst als gängige PS/2 Geräte ausgelegt, später ist eine Übertragung in das modernere USB vorgesehen.
- Um auf die Bildschirmausgaben des Rechners zugreifen zu können wird die PCI Karte selbst als Minimalversion einer VGA Karte konstruiert und der Bildschirminhalt wird auf der Karte gespeichert. Durch diese Maßnahme wird eine weitere Grafikkarte überflüssig, und die Kosten werden gesenkt.
- Zur Überwachung von Lüfterdrehzahl, Temperatur und Spannungsversorgung werden zusätzliche Sensoren verwendet oder auf die im Rechner bereits vorhandenen Sensoren zurückgegriffen. Später kann auch eine Überwachung der Festplatten über ein Mikrophon konstruiert werden. Falls der Rechner nicht mehr startet, sollen die POST-Statusnummern des BIOS-Selbsttestes auf dem PCI Bus mitgelesen werden können. Durch eine Analyse der Geräte auf dem PCI Bus kann die eingebaute Hardware identifiziert und an den Steuerrechner übermittelt werden.
- Möglichst viel der benötigten Logik wird in einem FPGA gespeichert, insbesondere die Logik zur Emulation der Peripheriegeräte und die Ansteuerung des PCI Busses.

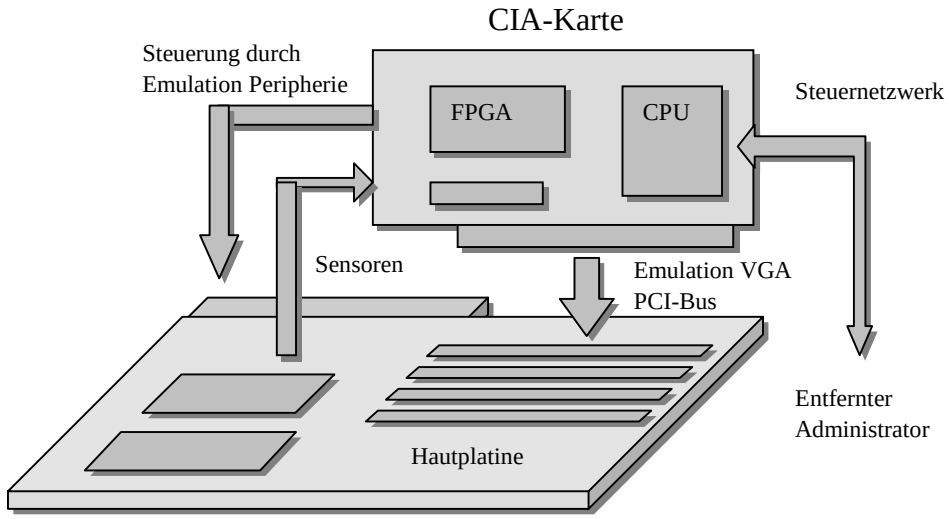


Abbildung 2: Prinzipieller Aufbau der CIA-Karte

Aufgrund der Eigenschaften wurde sich für den Namen *CIA-Karte* für den Prototypen entschieden. Die Abkürzung steht für *Cluster Interface Agent* und soll die Steuerungs- und Kontrollmöglichkeit der einzelnen Rechner des Clusters andeuten.

Die Schwerpunkte in der Entwicklung lagen also auf der Erstellung eines Hardwaredesigns für den ausgewählten FPGA, auf der Programmierung der CPU und auf der elektrischen Verbindung aller Komponenten. Im nächsten Kapitel wird die Zusammenstellung der Komponenten für den ersten Prototypen beschrieben. Auf weitere Möglichkeiten zum Ausbau wird am Abschluß dieser Arbeit eingegangen.

Kapitel 3

Auswahl der Komponenten, Aufbau des Prototypen

Im vorangegangenen Kapitel wurde das Konzept der Arbeit vorgestellt. Dieses Kapitel begründet die Auswahl der verwendeten FPGA/PCI-Entwicklungskarte und des verwendeten CPU-Moduls µCSimm. Weiterhin wird der Laboraufbau des Prototypen der CIA-Karte beschrieben. Die konkrete Realisierung des Designs im FPGA, die Beschaltung der Adapterplatine und die erstellte Software wird in den folgenden Kapiteln separat behandelt.

3.1 Auswahl der CPU

Teil des Konzepts war eine CPU auf der zu entwickelnden Karte, welche das Zusammenspiel der restlichen Komponenten steuern und die Netzwerkverbindung herstellen sollte. Zunächst mußte eine geeignete Auswahl getroffen werden. In diesem Zusammenhang waren folgende Randbedingungen zu berücksichtigen:

- Platzsparende Bauweise mit geringem Stromverbrauch.
- Leistungsfähigkeit, da schon wegen der Netzanbindung größere Datenmengen verarbeitet werden mußten.
- Einfache Verwendung im Projekt durch möglichst einfache Beschaltung, so daß nur wenig zusätzliche Chips erforderlich sind.
- Einfache Entwicklung der Programme, idealerweise mit eigener Entwicklungsumgebung auf einem separaten Rechner und Möglichkeiten, diese zu testen.

Das Einsatzgebiet war demnach typisch für sogenannte *embedded CPUs*, dabei handelt es sich im allgemeinen um kleine, autarke CPUs, die vielfach spezielle Steuerfunktionen wie Timer oder serielle Schnittstellen bereits integriert haben. Sie werden zu Steuerzwecken in größeren Komponenten

eingebaut. Aus diesen Bedingungen war zunächst klar, daß beispielsweise die in Arbeitsplatzrechnern eingesetzten CPUs wie die aus der x86-Reihe für den Einsatz nicht in Frage kamen.

Als mögliche Kandidaten ergaben sich verschiedene CPUs aus einer großen Auswahl wie Mikrochip PICs, Intel 8051, weiterhin CPUs der Firmen Atmel, Motorola, ARM oder embedded MIPS und anderen. Alle diese CPUs haben jeweils ihre speziellen Vor- und Nachteile, die hier aber nicht alle einzeln aufgeführt werden sollen.

Das CPU-Modul µCSimm

Letztendlich wurde sich für ein CPU-Modul der Firma *RT-Control* namens *µCSimm* [uCS] entschieden. Bei der CPU handelte es sich um die bekannte *Dragonball*-CPU von Motorola, welche eine Variante der in den bekannten *Palm*-Handhelds verwendeten CPUs ist. Seinen Namen erhielt das Modul aufgrund seines Steckers im sogenannten *SIMM*-Format (engl: *single inline memory module*), welcher bei Speicherbausteinen verwendet wird und eine platzsparende Verbindung vieler Signale erlaubt.

Es wurde allen obigen Anforderungen gerecht und hatte gegenüber dem großen sonstigen Angebot zwei wichtige Eigenschaften:

- Das Modul integrierte einen kompletten, sofort einsatzbereiten Rechner inklusive CPU, RAM, ROM und sogar ein Ethernet-Netzwerk auf einer nur 2,5 mal 9 cm großen, steckbaren Platine.
- Für das Modul stand ein vollwertiges Betriebssystem zur Verfügung, welches eigenständig auf dem Modul laufen konnte und darüber hinaus eine Abwandlung des bekannten Systemes *Linux* war.

Dadurch, daß es sich bei dem verwendeten Netzwerk bereits um das im Projekt erwünschten Ethernet handelte, konnte sehr viel Zeit beim Aufbau des Prototypen gespart werden. Die Rechenleistung des Modules war mit 15MHz nicht sehr groß aber für das Projekt ausreichend. Leider war die Netzwerkverbindung recht träge, da die korrekte Implementierung eines TCP/IP Netzwerkes einiges an Rechenleistung verlangt. Die CPU wurde daher meist über die serielle Verbindung angesprochen.

Das Modul verfügt über umfangreiche Möglichkeiten, um Strom zu sparen. Es kann bis in einen Modus versetzt werden, in dem nur einige µ-Ampère verbraucht werden. Durch die Integration des Netzwerkes in das CPU-Modul konnte die Akku-Versorgung von CPU und Netzwerkteil auf eine Notstromversorgung des CPU-Modules beschränkt werden. Da das Modul lediglich eine konstante Gleichspannung erwartet, sollte sich dieses leicht realisieren lassen.

Das Modul verfügte weiterhin über mehrere frei verwendbare Portleitungen. Über diese sollte die Verbindung zu dem restlichen Teil des Projektes hergestellt werden. Auch die Emulation der Gehäusetaster des Wirtsrechners sollte über einige dieser Leitungen geschehen, da im ausgeschalteten Zustand des Rechners nur das CPU-Modul über seinen Notstrom verfügen würde.

Bezeichnung	uC68EZ328 Single Inline Microcontroller Module
Formfaktor	2 cm * 9 cm, 30 Pin SIMM Stecker 3,3 V
CPU	16 MHz, 2,7 Mips Motorola Dragonball MC68EZ328
Speicher	8 MB DRAM, 2 MB Flash ROM
Netzwerk	Crystal LAN 10-Base T
Stromverbrauch	max 100 mA bis zu wenige mA (low power sleep mode)
IO	RS-232 9600 baud Seriell, 4 Mbit SPI, parallele Ports
weitere Funktionen	Watchdog, Timer in CPU, Ansteuerung für QVGA LCD
Betriebssystem	uCLinux, Anpassungen für Linux Kernel 2.0 und 2.4 verfügbar
Größe BS	ca. 1 MB
Entwicklung	Cross Compiler

Tabelle 1: Technische Daten µCSimm

Das Modul wurde zusammen mit einer kleinen Platine namens *gardener-kit* geliefert. Auf dieser befanden sich bereits Buchsen und Sockel für die Anschlüsse von Stromversorgung, Netzwerk und Seriellem Port. Der Zusammenbau gestaltete sich als einfach und unkompliziert. Über den seriellen Port konnte die CPU auf dem Modul auch direkt ohne den Umweg über das Netzwerk betrieben werden. Diese Möglichkeit sollte später für die lokale Verwendung der Karte genutzt werden.

Der relativ hohe Preis lag eine Größenordnung über der Summe der Einzelpreise der verwendeten Bauteile. Er wurde aber in Kauf genommen, da es sich bei der in der Arbeit zu entwickelnden Karte zunächst um einen Prototypen handeln sollte, um das Prinzip zu testen. Es war nicht geplant, das fertige Modul auch in späteren Versionen des Projektes einzusetzen.

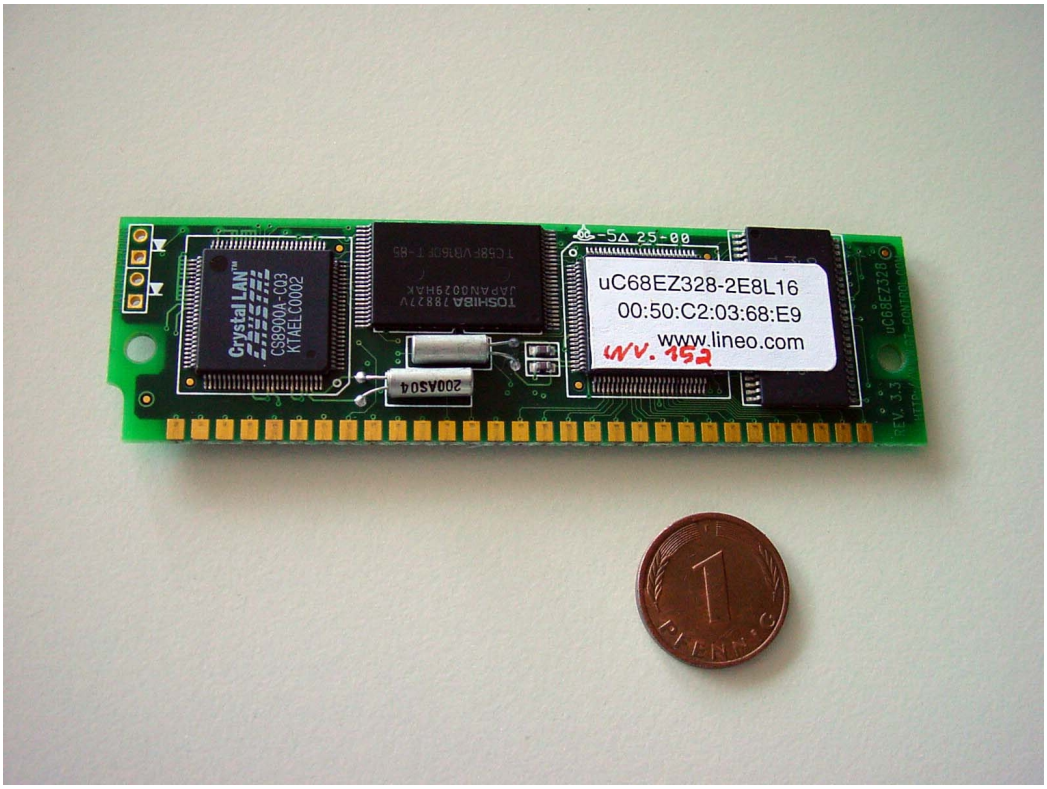


Abbildung 3: Das Modul µCSimm

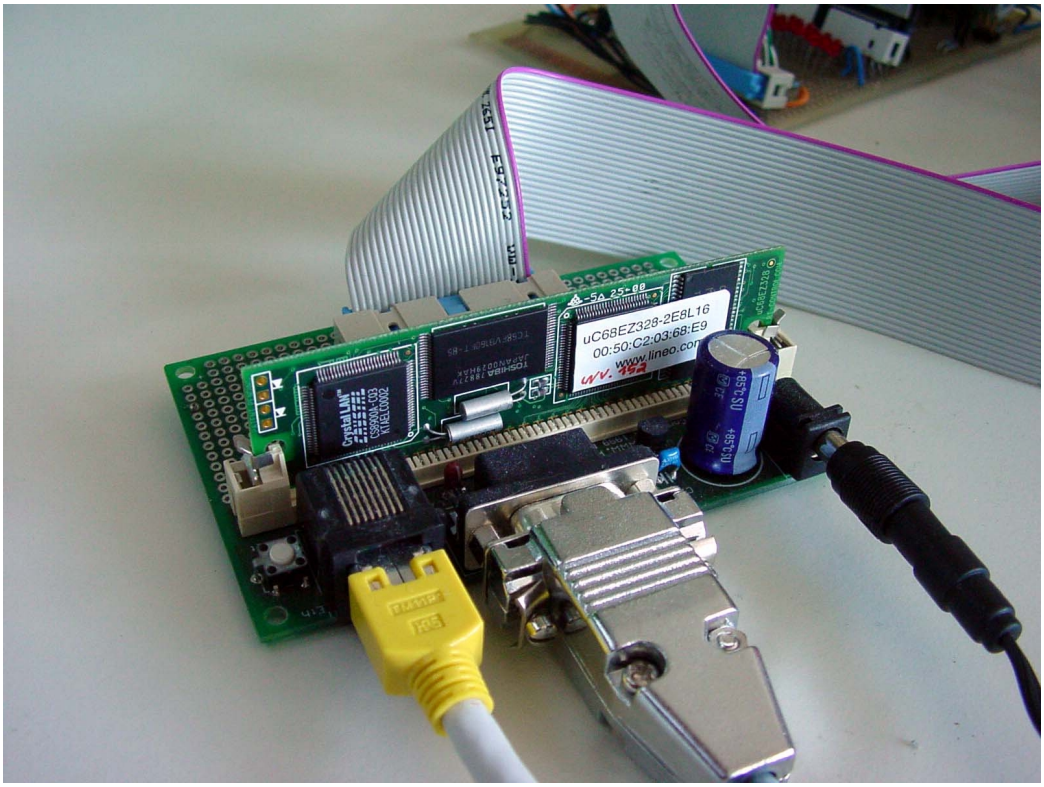


Abbildung 4: µCSimm beschaltet im Gardener Kit

Das Betriebssystem µCLinux

Der große Vorteil bei der Verwendung des Moduls µCSimm bestand in der Tatsache, daß es für dieses Modul bereits eine Anpassung des Betriebssystems µ*CLinux* gab, welches auf dem Modul betrieben werden konnte.

Das Betriebssystem µCLinux [uCL][En01] wird von einer unabhängigen Gruppe entwickelt und ist im Internet frei erhältlich. Es handelt sich um eine Umsetzung des bekannten Betriebssystemes *Linux*, welche aber speziell für kleine embedded CPUs konzipiert wurde. Insbesondere kann µCLinux auch auf Systemen ohne MMU verwendet werden, da es eine andere Speicherverwaltung besitzt. Ansonsten entspricht es bis auf einige unwesentliche Einschränkungen dem großen Bruder und verfügt über alle wichtigen Fähigkeiten wie Dateisystem, Benutzerverwaltung, Speicherverwaltung und Netzwerkunterstützung bis hin zu einem vollwertigem TCP/IP Stack.

Zusammen mit dem auf dem Modul befindlichen Netzwerkchip ergab sich damit ein vollwertiges Netzwerkbetriebssystem als Grundlage für die zu erstellenden Steuersoftware. Insbesondere kann sich ein Benutzer von außen auf dem System einwählen und beispielsweise Kommandos absetzen oder Programme starten.

Durch diesen Umstand wurde die Entwicklung von Steuerprogrammteilen sehr stark vereinfacht. Es ist in der Regel ohne größeren Aufwand möglich, bereits für Linux existierende Programme auch für µCLinux neu zu übersetzen und auf dem Modul zu starten. Zur Entwicklung wird ein gewöhnliches Linux-System benötigt, unter dem mit einem sogenannten *Cross-Compiler* ausführbare Binärdateien für µCLinux erzeugt werden.

3.2 Auswahl einer FPGA / PCI Entwicklerkarte

Da es sich bei FPGAs meist um Bausteine mit einer großen Anzahl von Anschlüssen handelt, war schon zu Beginn klar, daß der Kauf eines einzelnen FPGAs wenig Sinn machen würde. Um ihn zu betreiben, hätte zunächst ein eigene Platine mit einem umfassender Schaltplan erstellt werden müssen. Dabei hätte schon das Auflöten des Chips ein Problem bereitet, da FPGAs der erforderlichen Größe meist in der sogenannten *ball-grid-array*-Bauweise gefertigt werden, welche spezielle Lötmethoden verlangt.

Auf dem Markt gibt es aber zur Entwicklung mit FPGAs diverse Entwicklungsplatinen, die jeweils bereits mit entsprechenden Chips bestückt sind. Da diese Platinen zu Testzwecken gedacht sind, finden sich dort meist zusätzlich eine große Anzahl von Steckverbindungen, Anzeigen und oft auch spezielle weitere Komponenten wie nutzbare Speicherbausteine.

Weiterhin gibt es auch FPGA-Entwicklungsplatinen, welche speziell für die Entwicklung von PCI Geräten gedacht sind. Auf diesen Karten befindet sich dann ein entsprechender Stecker für den PCI-Bus, dessen Anschlüsse vom FPGA aus angesteuert werden können. Im folgenden muß zunächst dieser Zugriff auf den Bus genauer erklärt werden.

Verwendung eines PCI Cores

Bei dem Betrieb des PCI Busses muß die einwandfreie Kommunikation und Koordination der Geräte untereinander sichergestellt werden. Aus diesem Grund sieht die PCI-Spezifikation mehrere Steuersignale vor, welche von allen angeschlossenen Geräten korrekt behandelt werden müssen. Um nun eine einfache Verwendung des Busses zu ermöglichen, werden diese Aufgaben von einer eigenen

Komponente übernommen, welches ein oder mehrere PCI-Geräte repräsentiert und die korrekte logische und elektrische Signalsteuerung auf dem Bus übernimmt, ein sogenannter *PCI-Core*.

Der Core steht logisch gesehen zwischen dem Bus und der eigentlichen Funktion des betreffenden Gerätes. Auf der einen Seite kümmert er sich um die korrekte Signalsteuerung auf dem Bus und repräsentiert dem Bus gegenüber ein bestimmtes Gerät. Für den Fall, daß dieses Gerät durch andere Geräte auf dem Bus angesprochen wird, stellt er auf der Anwenderseite einige einfache Signale über die betreffende Aktion zur Verfügung. Auf diese Weise kann eine Hardwareanwendung den Bus nutzen, indem sie die einfachen Steuersignale des Cores verwendet.

Grundsätzlich gibt es die beiden Möglichkeiten, die Funktion des Cores in einem eigenen Mikrochip in Hardware zu übernehmen oder einen sogenannten *Softcore* zu erwerben. Ein in Hardware implementierter Core ist in der Regel ein eigener Schaltkreis, welcher einen Teil der zur Verfügung stehenden Chipfläche belegt oder separat in einem eigenen Mikrochip realisiert wurde. Ein Softcore liegt dagegen formuliert in einer Hardwarebeschreibungssprache vor und ist dadurch unabhängig von der zu verwendenden Zieltechnologie. Es wird einem Hardwareprojekt hinzugefügt und zusammen mit dem Rest des Projektes synthetisiert. Meist werden Softcores als verschlüsselte Dateien geliefert, so daß sie zwar konfiguriert und verwendet, aber nicht eigenhändig modifiziert werden können.

Der Vorteil eines hardkodierten Cores liegt in erster Linie in der größeren Geschwindigkeit. Der Core wurde bereits für die Zieltechnologie optimiert auf den Chip gebracht und kann sofort zuverlässig verwendet werden. Ein Softcore dagegen muß erst noch zusammen mit dem Rest des Projektes auf die Zieltechnologie abgebildet werden, wobei je nach Optimierungsstrategie unterschiedlich gute Ergebnisse erzielt werden können. Speziell in FPGAs verwendete Softcores müssen sich an die zur Verfügung stehenden Zellen und Verbindungen halten und werden auch deshalb nicht so schnell sein wie eigene Mikrochips. Ein weiterer Vorteil von Hardwarecores liegt darin, daß sie meist vergleichsweise günstig erworben werden können.

Dafür sind Softcores natürlich viel flexibler und vielseitiger konfigurierbar. In der Regel lassen sich vor der Synthese die nicht benutzten Funktionen aus dem Core entnehmen, wodurch letztendlich der Platzverbrauch im Chip beschränkt werden kann. Weil Softcores unabhängig von der Zieltechnologie sind, kann derselbe Softcore sowohl bei der Entwicklung in einem FPGA verwendet werden als auch später bei der Herstellung des fertigen Produktes in einem ASIC.

Bei FPGA/PCI Entwicklungskarten kann in der Regel auch ein entsprechender PCI Core zur Ansteuerung des Busses mit erworben werden. Da der PCI Bus einen recht hohen Speicherdurchsatz erlaubt, findet sich zumeist auch eine größere Menge Speicher mit auf der Karte, welcher über den FPGA genutzt werden kann. Weiterhin verfügen diese Karten über einige Anzeigen, Schalter oder Verbindungsstecker um möglichst flexibel mit ihnen experimentieren zu können.

Die Verwendung einer solche Karte hat also gleich mehrere Vorteile:

- Die wesentlichen Komponenten wie PCI-Stecker, passender FPGA, Speicher und weitere Verbindungsanschlüsse sind bereits vorhanden, aufeinander abgestimmt und vor allem getestet. Dadurch entfällt unter anderem die schwierige Bestückung des FPGAs und die Verlegung der Leiterbahnen für die zeitkritischen PCI-Signale.

- Weitaus geringere Fehleranfälligkeit, da Beispiele von Designs für verschiedene Einsatzbereiche in der Regel mitgeliefert werden und als Ausgangsbasis für die weitere Entwicklung verwendet werden können.
- Eine spezielle Software-Entwicklungsumgebung wird in der Regel mitgeliefert.
- Dadurch große Zeitersparnis bei der Entwicklung von PCI Geräten.

Solche Entwicklungskarten haben zwar einen recht hohen Preis, jedoch wird dieser durch die Zeitersparnis in der Regel wieder wett gemacht. Im Projekt wurde sich deshalb aus den genannten Gründen für den Einsatz einer solchen PCI-Entwicklungskarte mit FPGA für den Aufbau und Test des Prototypen entschieden. Dadurch wurde in Bezug auf die Anforderungen der Arbeit sowohl die Einbeziehung als auch die Beschaltung von PCI Bus und FPGA gleichermaßen auf einer einzigen PCI/FPGA Experimentierkarte integriert.

Erster Ansatz mit ORCA Karte

Zunächst wurde eine bereits im Labor vorhandene FPGA/PCI Experimentierkarte der Firma *Lucent* auf ihre Verwendbarkeit für die Arbeit getestet. Obwohl sich leider herausstellte, daß sie für den Einsatz nicht geeignet war, soll hier kurz auf diese Karte eingegangen werden.

Die Karte war mit einem FPGA vom Typ *ORCA OR3TP12 FPSC* bestückt und verfügte über 1 MB SRAM, einen bestückbaren DRAM Sockel, sowie einige Anzeigen, Schalter und Pfostenleisten, welche alle beliebig vom FPGA angesteuert werden konnten. Das besondere an dem FPGA war, daß dieser den zur Ansteuerung des PCI Busses notwendigen PCI Core bereits als eigenständigen fertigen ASIC im Gehäuse integriert hatte. Als Hardwarecore war dieser entsprechend schnell und die Karte konnte damit auch an einem 66 MHz PCI Bus betrieben werden. Die FPGA Anwendung war durch den Core vollständig von dem PCI Bus entkoppelt, und ein direkter Zugriff auf die Signale des Busses war nicht mehr möglich. Auf diese Weise konnte sich das Hardwaredesign im FPGA auf die Ansteuerung einiger weniger Signale des Cores reduzieren, um mit dem PCI Bus zu arbeiten.

Leider lag gerade im verwendeten PCI Core das Problem, weswegen die Karte nicht in der Arbeit verwendet werden konnte:

In der Projektplanung waren als zwei wesentliche Anforderungen die Möglichkeit der Emulation einer VGA-Karte und das Mitlesen von POST Statusnummern auf dem PCI Bus gestellt worden. Die POST-Statusnummern werden vom BIOS meist an untere Adressen im I/O-Bereich wie 0x80 geschrieben. Schreibzyklen in diesen Bereich sind jedoch an kein spezielles Gerät adressiert und müssen quasi passiv *nebenher* mitgelesen werden. Durch einige Experimente stellte sich heraus, daß die Register der VGA-Karte aus historischen Gründen ebenfalls nicht wie gewöhnliche Adressbereiche von PCI Geräten angesprochen werden. Solche Karten wurden ursprünglich für den alten ISA-Bus entwickelt und reagierten auf Schreib- und Lesebefehle nach *festen* Adressräumen. Der Kompatibilität wegen ist dieses Verhalten für Grafikkarten auf den PCI Bus übernommen worden. Zur Emulation einer VGA-Karte muß der PCI-Core daher genau wie bei POST Statusnummern auf Schreib/Lesezyklen an feste I/O-Bereiche ansprechen können.

Genau dies war aber mit dem verwendeten Core nicht möglich. Weiterhin war er nicht in der Lage, ein Erweiterungs-ROM zu einer PCI-Karte einzublenden, was ebenfalls zur Emulation der VGA-Karte notwendig gewesen wäre. Die Orca-Karte konnte deshalb nicht für die Arbeit verwendet werden.

Auswahl einer FPGA Karte

Da die Orca-Karte nicht verwendet werden konnte, mußte zunächst eine alternative FPGA/PCI Experimentierkarte ausgewählt werden. Um möglichst flexibel mit dem PCI Bus arbeiten zu können, sollte diesmal ein oben beschriebener Softcore verwendet werden.

Da der Softcore nur eine Komponente des Hardwaredesigns im FPGA werden würde, sollte auch der Zugriff auf den PCI Bus am Core vorbei möglich sein. Dadurch würde es möglich werden, die POST Statusnummern auf dem Bus mitlesen zu können, sowie die Schreib/Lesebefehle an eine VGA-Karte zu behandeln.

Weiterhin mußte sich für einen zu verwendenden Speichertyp entschieden werden. Auf der Karte war aus zwei Gründen separater Speicher erforderlich: Zum ersten sollte eine VGA-Karte emuliert werden, die den Textbildschirm und später evtl. auch den Grafikbildschirm speichert, um diesen auf Anfrage zu einem entfernten Rechner eines Administrators zu übertragen. Für einen Textbildschirm werden dabei 4 KByte in der Standardauflösung 80 Spalten mal 25 Zeilen benötigt. Die zweite Anwendung war die Emulation der Floppy. Um dem Wirtsrechner beim Startvorgang eine eingelegte Diskette vortäuschen zu können, müßten sich zumindest einige der Daten schon vor Ort in einem Zwischenspeicher befinden. Zur Speicherung der Daten einer Spur einer Diskette mit weiteren diskettenspezifischen Informationen werden ca. 25 KByte benötigt. Diese Daten sollten aufgrund der begrenzten Größe von FPGAs nicht in diesem selbst, sondern in einem eigenen Speicherbaustein gespeichert werden.

Grundsätzlich gibt es zwei Arten von Speicher: Statisches RAM (SRAM) und dynamisches RAM (DRAM). Bei statisches RAM wird ein Bit durch eine kleine Logik gespeichert, während bei dynamischem RAM ein Bit durch eine Transistor/Kondensator-Kombination gebildet wird. Diese kann sehr klein gebaut werden, wodurch auf DRAM Bausteinen viel mehr Informationen gespeichert werden können als auf SRAMS. DRAMs haben jedoch den Nachteil, daß die Kondensatoren sehr klein sind und regelmäßig aufgefrischt werden müssen, um sich nicht im Laufe der Zeit durch Leckströme zu entladen. Die Ansteuerung von DRAMs ist daher sehr viel komplizierter. Da ihr Aufbau jedoch einfacher ist, sind sie wesentlich billiger als SRAMs, wenn auch langsamer. Da für den Prototyp keine größeren Mengen Speicher erforderlich waren, wurde sich für SRAM entschieden.

Bei der konkreten Auswahl der Karte mußten also vor allem folgende Dinge beachtet werden:

- Der verwendete PCI Softcore sollte möglichst umfangreich und flexibel zu konfigurieren sein. Die Karte sollte zusammen mit dem Core möglichst universell in allen PCI Bussen verwendet werden können, zunächst wurde aber ein Betrieb mit 33 MHz angestrebt.
- Der auf der Karte eingesetzte FPGA sollte genug Platz bieten, um die notwendige Logik aufnehmen zu können. Insbesondere die Komponenten zur Emulation der Peripheriegeräte und zur Ansteuerung des PCI Busses würden eine komplexe Aufgabe darstellen und einiges an Logik erfordern.
- Der FPGA sollte schnell genug sein, um auch bei einem komplexen Design immer noch mit einem 66 MHz schnellen PCI Bus zusammen arbeiten zu können.
- Die Karte sollte mit SRAM, Steckern und Anzeigen bestückt sein.

- Es sollte eine alles umfassende Entwicklungssoftware mitgeliefert werden, inklusive diverser Demonstrationsprogramme, um zügig mit der Entwicklung beginnen zu können.

Zur Auswahl standen einige Karten von den diversen Herstellern von FPGAs wie Altera, Lucent usw. Auf die genaue Auswahl wird an dieser Stelle nicht weiter eingegangen. Im folgenden wird die in der Diplomarbeit verwendete Entwicklerkarte kurz vorgestellt.

Die PLDA Karte

Für die Erstellung des ersten Prototyps wurde sich für eine FPGA/PCI Entwicklerkarte der Firma *PLD Applications* vom Typ *PCI 20K-Prod-B* entschieden.

Der Hauptgrund für die Entscheidung für diese Karte war der eigens von der Firma hergestellte PCI Softcore. Obwohl dieser im FPGA auf dessen Logikzellen abgebildet wird und er kein eigener Schaltkreis wie der PCI Core der Orca-Karte ist, ist mit ihm auch der Betrieb bei 66 MHz möglich, falls ein schneller FPGA verwendet wird. Zusätzlich läßt sich der Core sehr vielseitig konfigurieren, so daß nicht benötigte Teile nicht in das Design eingebaut werden. Dadurch war er ideal für das Projekt geeignet.

Die Karte wurde von der Firma als Entwicklungskarte zusammen mit dem Softcore, einem FPGA und der entsprechenden Entwicklungssoftware verkauft. Der mit der Karte verwendete FPGA und die weitere Ausstattung mit SRAM oder weiteren Anschlüssen konnte dabei aus einer Liste von möglichen Komponenten gewählt werden. Dadurch war die Karte, der PCI Core und der FPGA recht gut aufeinander abgestimmt. Ein Nachteil des Entwicklungspakets insgesamt war zweifellos der recht hohe Preis, der etwa eine Größenordnung über dem angestrebten Preis einer fertig integrierten Lösung lag.

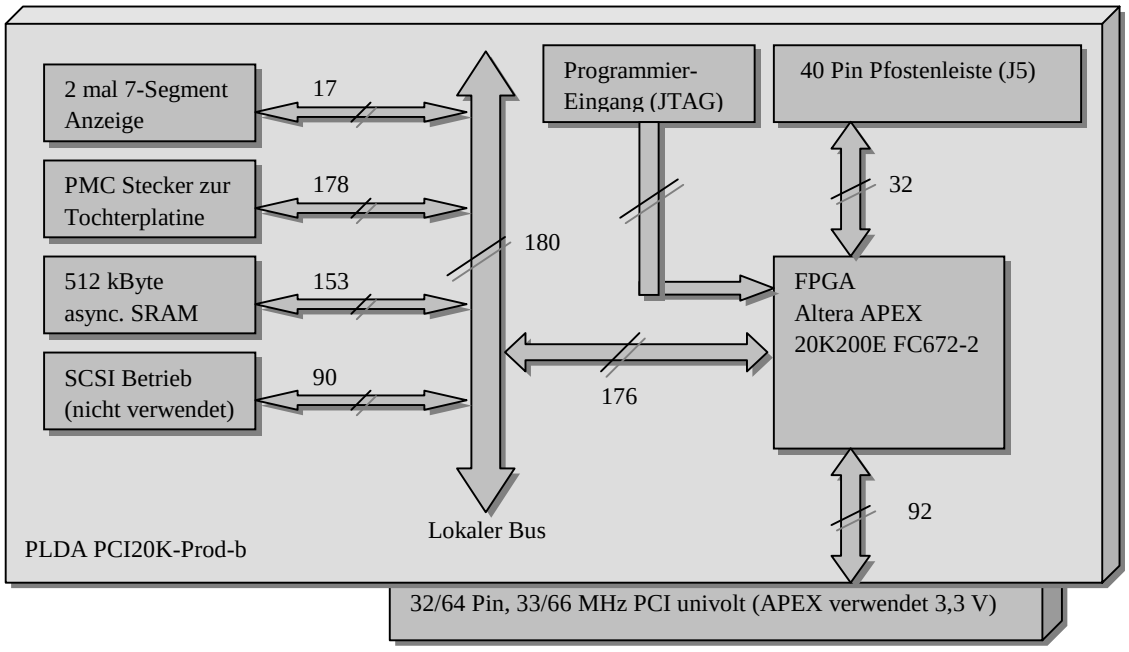


Abbildung 5: Blockschaltbild der PLDA Karte PCI 20K Prod B

Die Platine wird in einem 3,3V-PCI Steckplatz betrieben. Sie verfügt neben dem FPGA weiterhin über 512 KByte asynchronem SRAM, zwei 7-Segment Anzeigen, einem 40-poligen Pfostenstecker und zwei EEPROMs zur Konfiguration des FPGAs nach dem Einschalten. Alle wichtigen Komponenten werden durch einen 180 Bit breiten Bus auf der Platine verbunden. Über vier Steckplätze können alle Signale des Busses auf eine aufsteckbare Tochterplatine geleitet werden. Von hier aus können sie dann bequem durch ein Oszilloskop oder einen Logic-Analyzer abgegriffen werden, was die Fehlersuche während der Arbeit erheblich erleichtert hat.

Zur Verbindung mit weiteren Teilen des Projektes konnte der 40-polige Pfostenstecker verwendet werden, von dem 32 Anschlüsse bidirektional genutzt werden können und direkt mit bestimmten Anschlußbeinchen des FPGAs verbunden sind.

Der FPGA

Konkret wurde sich bei dem FPGA für den Typ *APEX EP20K200E FC672-2* der Firma *Altera* entschieden. Der FPGA ist mit 200.000 Gatter-Äquivalenten schon recht groß, so daß genug Raum für das gesamte FPGA-Design vorhanden sein sollte. Der prinzipielle Aufbau eines FPGAs wurde bereits in Abschnitt 2.8 erläutert.

Der Chip verfügt über 8320 Logikzellen mit denen sich 200.000 typische Gatter darstellen lassen. Zusätzlich verfügt er über 52 spezielle Logikblöcke namens ESB (e*mbedded system block*), in denen sich besonders gut Speicher, FIFOs etc. abbilden lassen. Jeder ESB kann 2048 Bit speichern, insgesamt besitzt der Chip also 106.496 dieser Bits. Jeweils 10 Logikzellen sind in sogenannten LABs zusammengefaßt (*logical array block*). 16 Dieser Blöcke werden zusammen mit einem ESB zu einem *MegaLAB* zusammengefaßt. Der Chip besteht aus 2 Spalten zu 26 Zeilen von MegaLABs. (Die Darstellung und Belegung eines Designs ist im verwendeten Programm *Quartus* im *Floorplan view* besonders schön sichtbar).

Von den 672 Anschlußbeinchen sind 376 frei nutzbar. Der verwendete Geschwindigkeitstyp beträgt 2 und reicht zusammen mit dem PCI-Core für einen PCI-Betrieb mit 66 MHz aus. Der Chip arbeitet intern mit 1,8 V Spannung, extern wahlweise mit 1,8 V, 2,5 V oder 3,3 V Spannung. Mit externen Widerständen läßt er sich auch an 5 V betreiben. Weitere Details über den Chip lassen sich dem Datenblatt [Alt01] entnehmen.

Bemerkung: Da auf der FPGA-Karte von PLDA keine Widerstände zum PCI-Bus verwendet werden, darf die Karte nur im 3,3 V PCI-Steckplatz verwendet werden !

Zur Programmierung des FPGAs wurde eine komplette Entwicklungsumgebung mitgeliefert. Hier fanden sich auch bereits einige Beispiele, wie grundlegende Aufgaben in Zusammenhang mit dem PCI Bus durch Komponenten im FPGA gelöst werden können. Diese Beispiele wurden zunächst als Basis bei der Erstellung des Designs genutzt.

Als Programmiersprache wurde die Hardwarebeschreibungssprache *VHDL* verwendet [Le94]. Eine kurze Beschreibung von VHDL wird im Rahmen der Beschreibung des im FPGA entstandenen Hardwaredesigns in Abschnitt 4.1 gegeben.

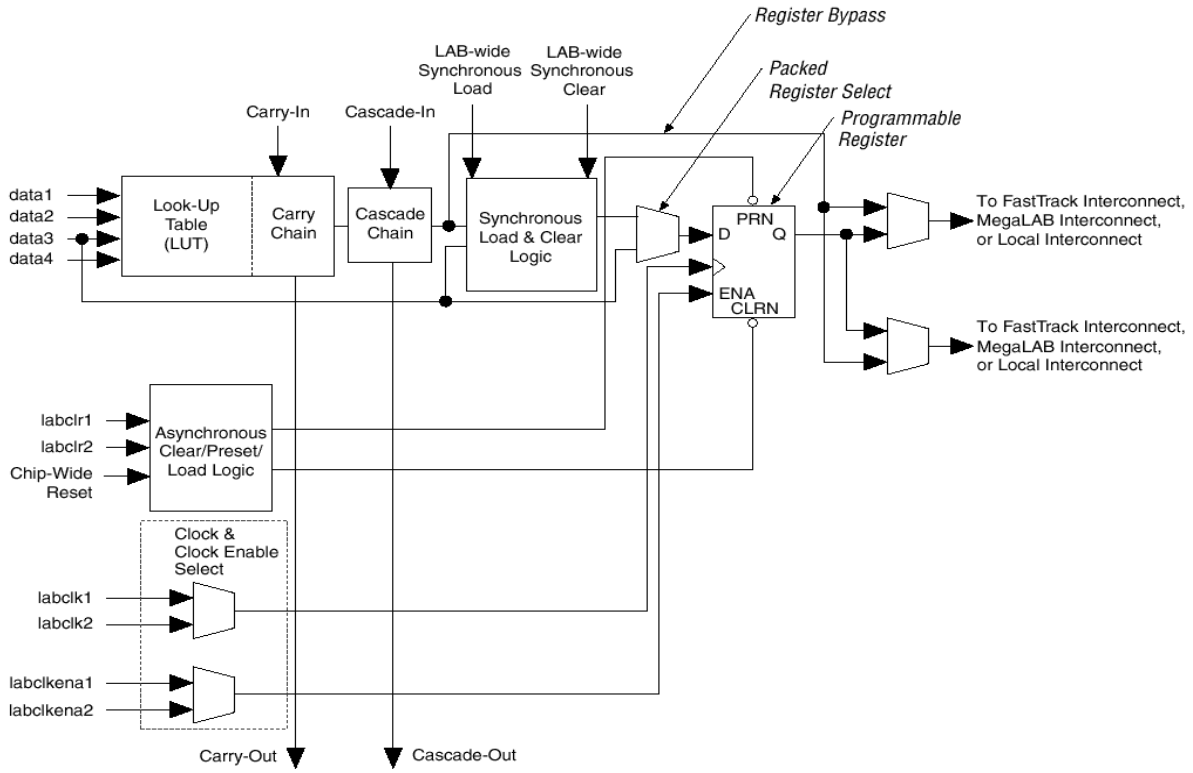


Abbildung 6: Logikzelle des verwendeten APEX 20 K (Quelle: Datenblatt [Alt01])

3.3 Laboraufbau

Zur Programmierung und Test des Hardwaredesigns im FPGA und der Software auf dem CPU Modul wurden weitere Rechner verwendet. Zusätzlich kamen ein Logic-Analyzer und ein PCI-Tracer zum Einsatz. Der folgende Abschnitt gibt einen Überblick über den gesamten Aufbau.

Entwicklung des FPGA Designs

Zunächst wurde die FPGA/PCI Entwicklungskarte in den 3,3V-PCI Bus eines dafür vorgesehenen Testrechners eingebaut. Mit Hilfe dieses Rechners konnte also die Implementierung der den PCI Bus betreffenden Komponenten im Hardwaredesgin des FPGAs getestet werden.

Da bereits etwas Software für diese Zwecke für ein Linux-Betriebssystem zur Verfügung stand, wurde der Testrechner unter diesem System betrieben. Natürlich sollte die Funktionalität der Karte letztlich nicht auf Rechner mit Linux beschränkt sein, sondern gerade einen Rechner auch ohne ein Betriebssystem steuern können. Daher wurden beispielsweise die Tests der Emulation der Peripheriegeräte auch mit anderen Betriebssystemen oder im BIOS des Rechners durchgeführt.

Die Entwicklung wurde auf einem lokalen Rechner unter dem Betriebssystem Windows erledigt. Hier lief die mit der Karte mitgelieferte Software *Quartus*, mit deren Hilfe das Design für den FPGA in der Sprache *VHDL* entwickelt wurde. Um das erstellte Design in den FPGA zu übertragen, wurde ein spezielles paralleles *Byte-Blaster*-Kabel verwendet, welches mit der FPGA-Karte an der *JTAG*-Schnittstelle verbunden wurde. Das Übertragen konnte ebenfalls mit der mitgelieferten Software ausgeführt werden.

Entwicklung der Steuersoftware

Zur Entwicklung der Software für das CPU Modul µCSimm wurde ein Rechner unter dem Betriebssystem Linux benötigt. Die Entwicklungsumgebung wurde deshalb auf einem weiteren bereits an das lokale Netzwerk angebundenen Linux-Rechner installiert, welcher über das Netz vom lokalen Hauptentwicklungsrechner aus bedient wurde.

Der bereits vorhandene Testrechner wurde zwar auch unter Linux betrieben, aus Sicherheitsgründen aber nicht für die Softwareentwicklung verwendet: Durch die Experimente mit der Entwicklungskarte im PCI Bus kam es häufig zu Stillständen, und der Rechner mußte neu gestartet werden ohne daß das Betriebssystem die Gelegenheit hatte, vorher das Dateisystem zu synchronisieren. Dieser Rechner stellte deshalb keine zuverlässige Grundlage für die Softwareentwicklung dar.

Das auf dem Linux-Rechner bestehende Entwicklungsverzeichnis mit den entstandenen Binärdateien wurde zur Verwendung durch das CPU Modul über das System *NFS* in das bestehende Ethernet Netzwerk exportiert. Das auf dem Modul laufende Betriebssystem µCLinux wiederum war in der Lage, dieses Verzeichnis ebenfalls über das Netzwerk mittels NFS zu importieren. Auf diese Weise konnten die neu erstellten Programme vom Modul augenblicklich ohne nötige Kopieroperationen verwendet werden.

Die Ansteuerung des Moduls wurde von dem Hauptentwicklungsrechner aus vorgenommen. Zu diesem Zweck wurde aber keine Verbindung über das Netzwerk hergestellt, sondern der serielle Anschluß des Modules verwendet, da die Rechenkraft der CPU nur eine recht langsame Implementierung des TCP/IP Netzwerkes gestattete. Die erstellten Programme konnten dann an der Eingabeaufforderung des Betriebssystemes µCSimm gestartet werden.

Durch diesen Aufbau konnte sowohl die Entwicklung des Hardwaredesigns für den FPGA, als auch die Erstellung und der Test der Steuersoftware für das CPU Modul von einem Rechner aus geleistet werden.

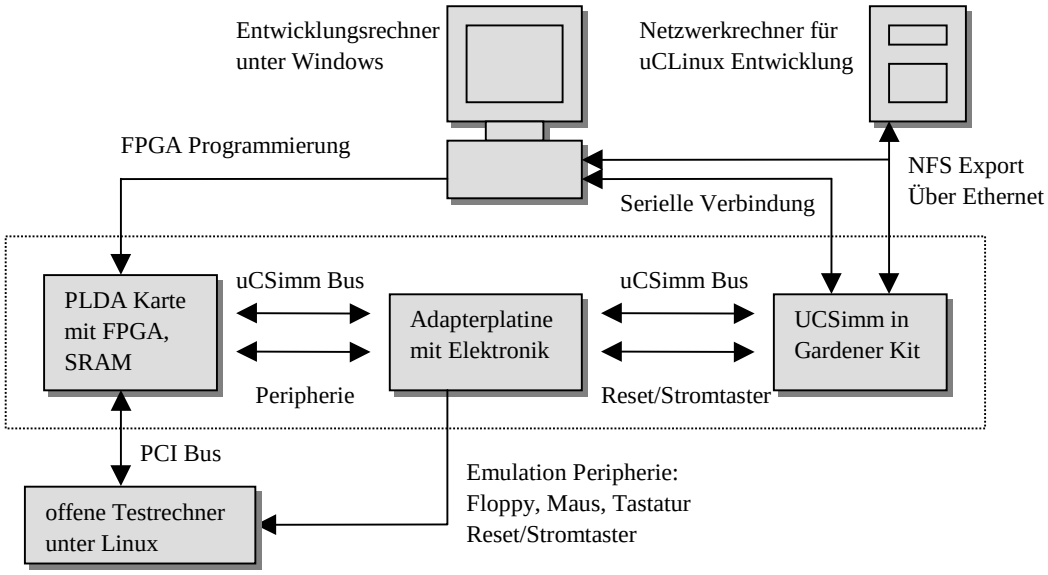


Abbildung 7: Übersicht über den Laboraufbau, die funktionellen Teile der CIA-Karte sind gestrichelt angedeutet

3.4 Aufbau und Funktion des Prototypen

Der Prototyp der CIA Karte bestand aus der FPGA Karte, dem CPU Modul und einer Adapterplatine. Wenn eine gewisse Funktionalität erreicht und die grundsätzlichen Probleme des Projektes gelöst waren, sollte in einem späteren Schritt alles auf einer eigenen Platine mit eingekauften Bauteilen realisiert werden.

Im Folgenden wird ein Überblick über das Funktionsprinzip und die grundsätzliche Zusammenarbeit der einzelnen Teile des erstellten Prototypen gegeben:

FPGA

Um die Steuer-CPU soweit wie möglich zu entlasten, werden alle zeitkritischen oder parallel auszuführenden Aufgaben im FPGA auf der Entwicklungskarte in Hardware gelöst. Unter diesen Bereich fallen sämtliche Aufgaben in Zusammenhang mit dem PCI Bus, sowie die Emulation der Eingabegeräte Maus, Tastatur und Floppy. Die Aufgaben sind in Komponenten zusammengefaßt und werden in ein gemeinsame Konfigurationsregisterbank abgebildet.

PCI Bus

Das SRAM auf der Karte und die Registerbank der Komponenten wird in den Adressbereiche des PCI Busses abgebildet. Dadurch kann die CIA-Karte zusätzlich in der Testphase vom bestückten Testrechner über den PCI Bus gesteuert werden. Weiterhin werden POST-Nummern über den PCI-Bus mitgelesen. Die Emulation einer VGA-Karte konnte nicht mehr fertiggestellt werden, jedoch wurden bereits erste Probleme überwunden.

CPU

Über die Adapterplatine wurde ein Datenbus zwischen CPU und FPGA implementiert. Der Bus besitzt eine Datenbreite von acht Bit und verwendet drei weitere Leitung zur Steuerung. Über diesen Bus ist die CPU in der Lage, einzelne Kommandos zum FPGA zu schicken und Daten in beide Richtungen zu übertragen. Eine weitere Leitung wurde als Interruptleitung konstruiert. Mit ihr kann der FPGA die CPU auf bestimmte Ereignisse aufmerksam machen. Die Programmteile zur Ansteuerung des Busses wurden in eine eigene Softwarebibliothek verlagert. Durch Einbindung dieser Bibliothek können alle Programme auf dem CPU Modul den Bus verwenden. Über den Bus können insbesondere Schreib- und Lesebefehle in die Registerbank des FPGAs ausgeführt werden, um die Hardwarekomponenten zu steuern. Die eigene Stromversorgung der CPU wird durch ein Netzteil geleistet. Die Verwendung eines aufladbaren Akkus muß noch implementiert werden.

Im Betrieb der CIA-Karte nimmt die CPU über die Netzwerkverbindung Steuerkommandos entgegen. Aufgrund der Kommandos werden über den Bus einzelne Module im FPGA angesteuert, welche beispielsweise Eingabegeräte emulieren. Die Signale dieser Geräte werden auf der Adapterplatine den Spannungsbereichen des Rechners angepasst und über Kabel an die entsprechenden Eingänge geführt.

Falls der Wirtsrechner selbst ein emuliertes Gerät anspricht, löst die Komponente im FPGA ein Signal auf der Interruptleitung aus und das Programm auf der CPU muß die Ursache ermitteln und entsprechend reagieren.

SRAM

Es wurde eine eigene Komponente im FPGA zur Ansteuerung des SRAMs entwickelt, welche vom PCI-Bus des Wirtsrechners und der CPU auf dem CPU-Modul angesteuert werden kann. Die CPU kann dadurch direkt Daten von und zum SRAM auf der FPGA-Karte übertragen. Das SRAM wird für die Zwischenspeicherung von Datenspuren einer emulierten Diskette und zur Aufnahme des VGA-BIOSses genutzt.

Adapterplatine

Auf der Adapterplatine befindet sich der Datenbus vom FPGA zum CPU-Modul. Weiterhin wurden hier die Stecker für Maus, Tastatur und Floppy zusammen mit der notwendigen Treiberlogik aufgebracht. Dadurch mußten keine Lötarbeiten auf der FPGA Karte oder dem CPU Modul selbst vorgenommen werden.

Die Hauptentwicklung des Projektes konzentrierte sich dadurch auf die Erstellung des Hardwaredesigns im FPGA, die Erstellung der notwendigen Verbindungselektronik auf der Adapterplatine und die Programmierung der Steuersoftware für die CPU. Jedes dieser Bereiche wird im folgenden in einem eigenen Kapitel behandelt. Am Ende der Arbeit wird eine Übersicht über den Projektstatus in Bezug auf die Anforderungen gegeben und die weiteren Schritte eines möglichen Ausbaus diskutiert.

Speziell die Aufgabe der Steuerung der gesamten CIA-Karte, also primär die Kommunikation von einem entfernten Rechner aus mit der CPU über das Ethernet-Netzwerk, wurde schwerpunktsmäßig in der Arbeit von Martin Kirsch behandelt. Weiterhin entwickelte er einzelne Teile des Hardwaredesigns im FPGA zur Emulation der Peripheriegeräte, sowie Teile der Software auf dem CPU-Modul, insbesondere einen Interrupt-Handler. In der Arbeit wird an entsprechender Stelle darauf hingewiesen.

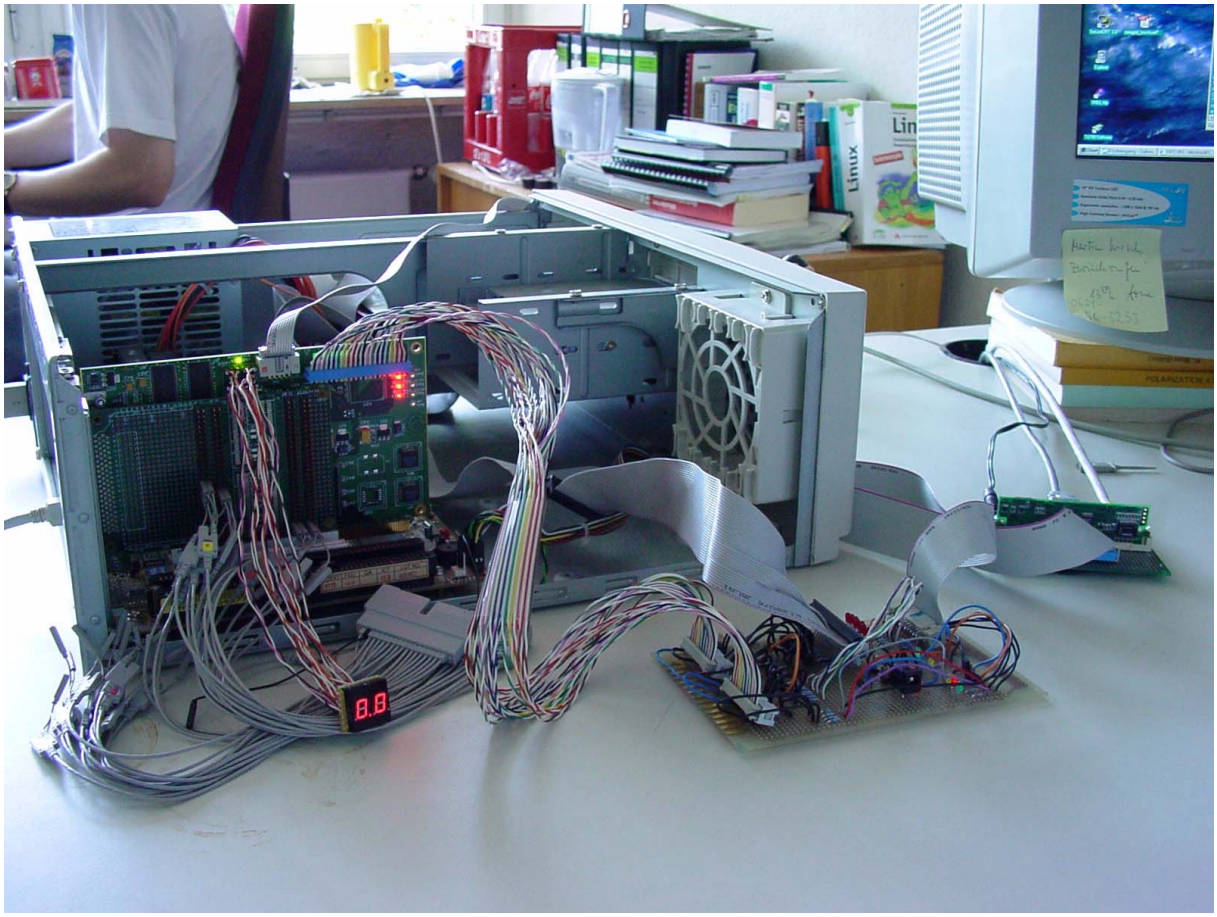


Abbildung 8: **Foto des Laboraufbaues, man sieht die FPGA-Karte im Testrechner, die Adapterplatine, sowie im Hintergrund das Modul µCSimm. Rechts am Rand befindet sich der Hauptentwicklungsrechner. Nicht im Bild ist der Netzwerk-Entwicklungsrechner, sowie die verwendeten Meßgeräte.**

Kapitel 4

Das Design im FPGA

Wie bereits im vorangegangenen Kapitel erläutert, sollten möglichst viele der anfallenden Aufgaben der CIA-Karte als Hardwarekomponente im FPGA realisiert werden. Dieses Kapitel stellt die entstandenen Komponenten vor und beschreibt deren Zusammenspiel. Die elektrische Verbindung des FPGAs zur Steuer-CPU und zum Wirtsrechner wurde über eine Adapterplatine ausgeführt, welche im Kapitel 5 beschrieben wird. Um die verschiedenen Möglichkeiten des FPGA-Designs zu testen wurden Programme erstellt, welche in Kapitel 6 vorgestellt werden.

4.1 Allgemeine Worte zum Design

Zum Verständnis des vorliegenden Kapitels sind einige Grundlagen in Hardwareprogrammierung erforderlich. Das gesamte Design im FPGA wurde mit der Hardwarebeschreibungssprache *VHDL* erstellt. Die Programmiersprache VHDL soll hier nicht näher beschrieben werden, dazu sei auf die Literatur verwiesen [Le94]. Auf die wichtigsten Punkte soll hier aber kurz eingegangen werden.

Entwicklung in VHDL

Die Entwicklung in VHDL verläuft analog zur Programmierung von Software. Die gesamte Aufgabenstellung wird dabei zunächst in Teilaufgaben zerlegt und separat gelöst. Das Projekt besteht dadurch aus einzelnen Teilen, welche im folgenden als *Komponente* bezeichnet werden. Die Beschreibung des gewünschten Verhaltens geschieht dabei grundsätzlich in Textform und wird in verschiedenen Dateien gespeichert.

Beim Erstellen eines Hardwaredesign gibt es einige Besonderheiten zu beachten:

- Die Formulierung geschieht zunächst rein logisch funktionell und damit unabhängig von einer speziellen Hardware. Sie wird später auf die zu verwendende Hardware abgebildet. In dieser Tatsache liegt ein großer Vorteil einer Hardwarebeschreibungssprache.
- Die beschriebene Programmlogik wird in entsprechende elektrische Logik wie Gatter oder Logiktabellen übersetzt. Variablen oder Konstanten werden durch Signalpfade dargestellt. Zur Speicherung von Informationen werden sogenannte *Register* verwendet. Die Register können je nach Breite eine bestimmte Anzahl Bits speichern.

- Eine Komponente kann eine andere Komponente zur Verwendung einbinden, ähnlich einer *Funktion* in Software. Dies resultiert in Hardware in einer elektrischen Verschaltung der beiden Teile. Die zu verbindenden Signale müssen bei der Programmierung in einer sogenannten *Entity* angegeben werden. Die Entity formuliert also die aus der Komponente herausgeführten Signale und entspricht in etwa einem Funktionskopf bei der Softwareprogrammierung. An der Spitze des Designs steht eine oberste Komponente welche das gesamte Hardwaredesign repräsentiert.
- In der Implementierung in Hardware werden alle einzelnen beschriebenen Teile parallel ausgeführt. Die Referenz und Synchronisation der unterschiedlichen Komponenten und deren Register untereinander wird durch ein globales Taktsignal ermöglicht. Das Design besteht damit aus einer Ansammlung von synchron zum Takt angesteuerten Registern, welche durch asynchrone Logik verbunden sind. Wichtig ist in diesem Zusammenhang, daß die Schaltzeit des asynchronen Teils nicht die Zeit zwischen zwei Taktsignalen überschreitet. Aus diesem Grund wird die maximale Geschwindigkeit des Designs durch den langsamsten Weg zwischen zwei Registern vorgegeben (*kritische Pfad*).

Der *Compiler* erzeugt dann aus den erstellten Dateien eine Beschreibung, welche die konkrete Umsetzung in Hardware entspricht. Dieser mehrstufige Prozeß wird im allgemeinen als *Designflow* bezeichnet.

Als ersten Schritt wird die logische Funktionalität des Hardware*designs* untersucht und durch eine Beschreibung von zu verschaltenden logischen Grundelementen dargestellt, eine sogenannte *Netzliste*. Diese wird zunächst hardwareunabhängig formuliert und in einem weiteren Schritt auf die konkret zur Verfügung stehenden logischen Elemente der verwendeten Technologie abgebildet. Den Verlauf bis zu diesem Punkt bezeichnet man als *Synthese*. Im nächsten Schritt muß entschieden werden, an welcher Stelle man die Ressourcen des FPGAs verwendet (*place*) und wie diese dann möglichst direkt über das bestehende Verbindungsnetzwerk verbunden werden können (*route*). Da die Anzahl der Logikressourcen und auch die Anzahl der möglichen Verbindungswege begrenzt ist, erweisen sich diese beiden Aufgaben als hochkomplex und werden in der Regel iterativ abwechselnd berechnet. Die Güte der Berechnung entscheidet über die mögliche Maximalgeschwindigkeit des Designs. Am Ende des Designflows steht ein Bitstrom, welcher mit entsprechender Programmierhardware über spezielle Anschlüsse des FPGAs in diesen übertragen wird.

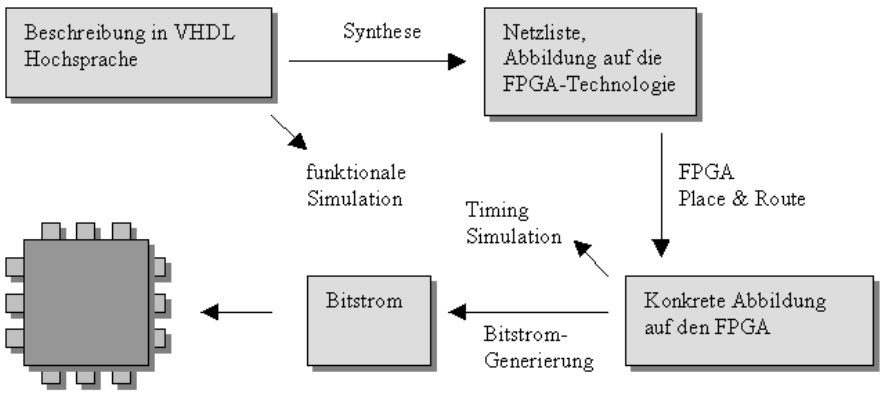


Abbildung 9: Schematische Darstellung des FPGA-Designflows

Überblick über das Design

Zur Entwicklung des Designs wurde das zu der FPGA-Karte mitgelieferte Programm *Quartus* verwendet. Unter diesem Programm konnte sowohl die Bearbeitung des Quelltextes, als auch die Syntaxüberprüfung, die Übersetzung in Hardware und die Übertragung in den FPGA erledigt werden. Die Dateien des Projektes befinden sich auf der beigelegten CD, siehe Anhang D.

Das gesamte Design wird über den PCI-Bus mit dem globalen Taktsignal von 33 MHz versorgt. Die verwendete Synthesesoftware gibt für das gesamte Design mögliche Geschwindigkeiten um 50 MHz an. Bei weiterer Optimierung ließe sich das Design mit dem verwendeten PCI-Core und FPGA auch in einem 66 MHz-Bus betreiben. Zum Zwecke der Datenübertragung zur CPU auf dem CPU-Modul oder für die Emulation der Eingabegeräte des Wirtsrechners werden noch verschiedene weitere Taktfrequenzen verwendet, auf die an entsprechender Stelle eingegangen wird.

Bei der Erstellung des Designs wurde grundsätzlich Wert auf einen modularen Aufbau gelegt und das Projekt den einzelnen Funktionen nach in einzelne Komponenten zerlegt. Die meisten Komponenten dienen der Ansteuerung externer Hardware. Bei diesen werden sowohl die externen Signale an die Hardware als auch die internen Steuersignale über ihre Entity herausgeführt. Während erstere über die Hauptkomponente heraus zu den Anschlüssen des FPGAs geleitet werden, werden die internen Signale auf Adressen in einer gemeinsamen Registerbank abgebildet, wo sie durch die Programme auf dem CPU-Modul angesteuert werden können. Auf diese Weise wird das gesamte Projekt sehr übersichtlich. Einzelnen Funktionalitäten sind in eigenen Dateien verpackt und können modular erweitert werden.

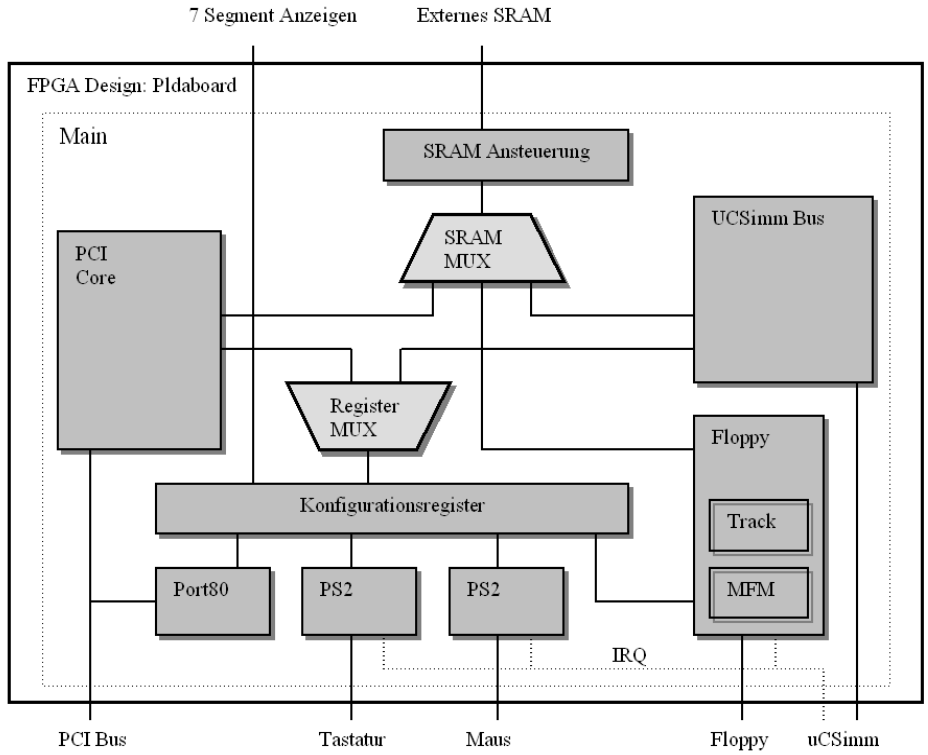


Abbildung 10: Schematische Darstellung der Komponenten des FPGA-Designs

Im einzelnen besteht das Projekt aus folgenden Dateien:

- Der Komponente *pldaboard.vhd.* Sie ist die oberste Komponente in der Projekthierarchie und repräsentiert das gesamte Design im FPGA. Sie bindet die Hauptkomponente ein und nimmt Zuweisung vor, welche sich speziell auf die verwendete FPGA-Karte beziehen.
- Der Hauptkomponente *main.vhd*. Sie bildet den gesamten logisch funktionellen Teil des Projektes. Sie bindet alle weiteren Komponenten ein und implementiert die gemeinsame Konfigurationsregisterbank zu deren Ansteuerung.
- Dem PCI-Core *pcicore.vhd*. Diese Komponente wurde mit dem zur FPGA Karte mitgelieferten PCI-Wizard von PLDA erzeugt.
- Der Komponente *external_sram.vhd* zur Ansteuerung des auf der FPGA Karte vorhandenen SRAMs.
- Der Komponente *µcsimm_ctrl.vhd* zur Implementierung des Übertragungsbusses zur CPU auf dem CPU-Modul.
- Der Komponente *ps2.vhd* um die Eingabegeräte Maus und Tastatur zu emulieren.
- Den Komponenten *floppy_ctrl.vhd, floppy_mfm.vhd* und *floppy_track.vhd* um eine vorhandene Floppy zu emulieren.
- Der Komponente *port80.vhd* um POST-Nummern des Wirtsrechners passiv am PCI Bus mitlesen zu können.
- Weiteren Hilfskomponenten: *digifilter.vhd, hex_decoder.vhd*.

4.2 Die oberste Projektdatei

Die Datei *pldaboard.vhd* bildet die oberste Datei in der Projekthierarchie. In ihr werden alle Einstellungen vorgenommen, welche zwar für die korrekte Funktion des Designs notwendig sind, letztlich aber nur speziell auf die verwendete Hardware zutreffen. Weiterhin wird in ihr die Hauptkomponente *main* eingebunden, welche die gesamte logische Beschreibung des Designs enthält. Eine solche oberste Komponente wird gewöhnlich als *Top-Level* Komponente bezeichnet.

Durch die Trennung von logischer Beschreibung und speziellen Einstellungen muß bei einem späteren Wechsel auf eine andere Hardware nur die Top-Level Komponente modifiziert werden. Um funktionale Teile mittels einer Simulation zu testen kann die Hauptkomponente in eine entsprechende Simulationsumgebung eingebunden werden. Diese Trennung erweist sich als praktisch, ist aber nicht zwingend.

Die folgenden Zuweisungen werden in der Top-Level Komponente vorgenommen:

- Einige Signale, die auf der verwendeten FPGA Karte vorhanden sind werden im Design nicht verwendet. Das betrifft beispielsweise die SRAM Signale für die unbenutzten oberen 384

KByte RAM. Um Logikzellen zu sparen, werden diese Signale nicht mit einen festen Wert belegt sondern einfach nicht verwendet.

- Der PCI Bus hingegen wird von mehreren Geräten verwendet und bei unbenutzten Signalen muß die entsprechende I/O-Zelle im FPGA als *Tristate* geschaltet werden um den Bus nicht zu stören. Leider werden nicht verwendete aber als Eingang beschaltete Signale von der verwendeten Software bemängelt. Aus diesem Grund wurden die Signale komplett auskommentiert und eine entsprechende Projekteinstellung vorgenommen, die global nicht benutzte Anschlüsse des FPGAs als Tristate deklariert. Es handelt sich hierbei um die nicht verwendeten PCI Master-Signale, 64-Bit-Steuersignale und Cardbus-Signale.
- Der auf der Karte befindliche Pfostenstecker *J5* mit 32 Signalen wird für die Verbindung des FPGAs zur Adapterplatine benutzt. Leider können die einzelnen Signale des Busses *ext_j5_io* jedoch nicht beliebig unterschiedlich als Eingang oder Ausgang deklariert werden. Die Signale werden daher einzeln unter eigenem Namen aufgeführt und ihre Belegung auf die jeweiligen Anschlüsse des Pfostensteckers wurde separat in den Projekteinstellungen angegeben.

4.3 Die Hauptkomponente Main

Die Hauptkomponente repräsentiert die gesamte logische Funktionalität des Designs. Sie selbst wird schließlich von der Top-Level Komponente *pldaboard* eingebunden, in welcher spezielle Zuweisungen für die verwendete FPGA-Karte gemacht werden. Durch diese Maßnahme wird die reine Funktion der Schaltung von äußeren Einstellungen getrennt.

Im einzelnen wird in der Hauptkomponente folgendes erledigt:

- Verbindung zum PCI-Bus durch Einbindung der PCI-Komponente.
- Verbindung zur Steuer-CPU durch Einbindung der CPU-Komponente und Herausführen des Interrupt-Signales zur CPU.
- Ansteuerung des SRAMs auf der Karte durch Einbindung der SRAM-Komponente.
- Einbindung der Komponenten zur Emulation von Maus, Tastatur und Floppy.
- Einbinden der Komponente zum Mitlesen von POST-Statusnummern.
- Ansteuerung der beiden 7-Segment Anzeigen.
- Zusammenstellen der Steuersignale der Komponenten in einer Registerbank.

Die weiteren Teile des Projektes werden in den jeweiligen folgenden Abschnitten erklärt. An dieser Stelle soll zunächst die in der Hauptkomponente selbst realisierte Konfigurationsregisterbank besprochen werden.

Implementierung der Konfigurationsregisterbank

Die Konfigurationsregisterbank entstand aus der Notwendigkeit heraus, die Steuerung der einzelnen Komponenten durch die Programme auf dem CPU-Modul auf einfache Weise zu ermöglichen. Alle

Komponenten haben zu ihrer Ansteuerung jeweils einzelne Signale. Um die Komponenten auf einfache Weise ansprechen zu können wurden alle diese Signale in der Hauptkomponente zunächst durch einzelne Register repräsentiert und in einen virtuellen Adressbereich abgebildet, die *Konfigurationsregisterbank.* Schreib- oder Lesebefehle an diese Adressen betreffen also die Steuersignale der Komponenten. Eine Übersicht über die Belegung der Konfigurationsregister findet sich in Anhang A.

Der Speicherbereich der Konfigurationsregister kann mittels eines Multiplexers über zwei Wege angesprochen werden:

- Über den PCI-Bus. Zu diesem Zweck wird das PCI-Adressfenster *BAR0* verwendet.
- Mittels der Programme auf dem CPU-Modul durch die Komponente *µcsimm_ctrl*. Zu diesem Zweck wird der bereits erwähnten Übertragungsbus über die Adapterplatine verwendet.

Die Ansteuerung durch die CPU ist der im Projekt für den Einsatz vorgesehene Fall. Die Ansteuerung über den PCI-Bus ist zu Testzwecken und zur Fehlerbehebung während der Entwicklungsphase gedacht, falls ein Zugriff über das CPU-Modul nicht möglich oder gewollt ist. Zur Verwendung können die unter Linux auf dem Testrechner entstandenen Programme verwendet werden, die in Abschnitt 6.10 beschrieben werden.

Der Multiplexer wurde nicht als Komponente eingebunden, sondern innerhalb der Datei in VHDL beschrieben. Man beachte, daß der Multiplexer bei Zugriffen sowohl durch Programme auf dem CPU-Modul als auch vom PCI Bus aus immer dem PCI-Bus den Vorzug gibt. Bei gleichzeitigem Zugriff werden an der CPU falsche Werte gelesen werden. Dies wird jedoch im Einsatz der Karte nicht der Fall sein, da Zugriffe über den PCI-Bus nur in der Testphase vorgesehen sind. Zu einem späteren Zeitpunkt ist eine Deaktivierung des PCI-Zugriffs leicht möglich.

Da die internen Datenpfade das Designs 32 Bit Breite haben, muß bei den Zugriffen auf die Registerbank das betroffene Byte innerhalb der 32 Bit kenntlich gemacht werden. Zu diesem Zweck werden sogenannte *Byte-enable*-Signale verwendet. Diese Signale liegen auf dem PCI-Bus bereits an, von der Komponente zur CPU-Anbindung werden sie separat erzeugt.

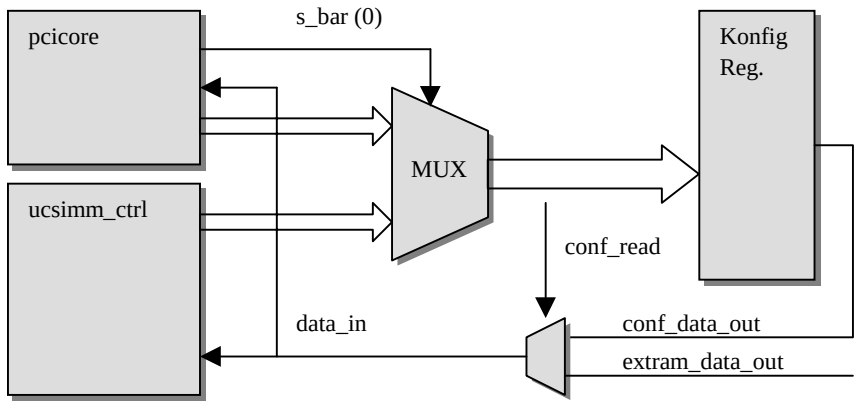


Abbildung 11: Datenpfade beim Zugriff auf die Konfigurationsregisterbank

Bemerkung:

Die Ansteuerung des SRAMs wurde ebenfalls durch einen solchen Multiplexer gelöst, wodurch ähnliche Probleme bei der gleichzeitigen Ansteuerung durch verschiedene Teile des Projektes entstanden. Näheres siehe im Abschnitt über die Anbindung des SRAMs.

4.4 Anbindung an den PCI Bus

Die Anbindung an den PCI Bus sollte neben der Stromversorgung der Karte folgendes leisten:

- Emulation der Karte als VGA-Karte (PCI I/O *Slave*)
- Erstellung einer Übersicht über die im Wirtsrechner installierte Hardware (PCI *Master*)
- Mitlesen der POST Statusnummern während des Selbsttestes (passiv an I/O Adresse 80)
- Zugriff auf das FPGA Design in der Testphase. Diese Möglichkeit war keine ursprüngliche Anforderung des Projekts (PCI *Slave*).

Um die Zusammenarbeit mit dem PCI Bus zu verstehen muß zunächst kurz auf die Arbeitsweise des Busses eingegangen werden. Die Darstellung erfolgt jedoch äußerst knapp. Zum genauen Studium sei auf weitere Literatur verwiesen [Sha98].

Arbeitsweise des PCI Busses

Bei dem PCI-Bus handelt es sich um einen synchronen Bus, dessen Funktion in der Verbindung von systemweiten Hardwarekomponenten und der Integration zusätzlicher Erweiterungskarten besteht. Jedes einzelne am Bus teilnehmende Gerät wird dabei durch eine eindeutige Nummer repräsentiert. Mittels einer sogenannten *Bridge* kann der Bus an weitere Busse angeschlossen werden, die Bridge stellt dabei selbst auch ein besonderes PCI-Gerät dar.

Die Kommunikation auf dem Bus erfolgt in einzelnen Zyklen. Da immer nur ein Gerät auf den Bus zugreifen darf, wird der Zugriff durch einen sogenannten *Arbiter* gesteuert. Ein sendendes Gerät muß also zunächst vom Arbiter die Erlaubnis haben um auf den Bus schreiben zu dürfen. Zu Beginn eines Zyklusses wird zunächst ein Kommando in Verbindung mit einer Adresse auf den Bus gegeben. Diese werden von allen am Bus angeschlossenen Geräten dekodiert. Falls sich ein anderes Gerät auf dem Bus angesprochen fühlt, wird eine Datenübertragung gestartet. Im folgenden kann bis zu einer gewissen Maximalzeit in jedem Bustakt ein Datenwort übertragen werden, dies nennt man einen *Burst*. Anschließend wird der Zyklus beendet und der Bus ist für eine weitere Übertragung frei. Das einen Zyklus beginnende Gerät wird in diesem Zusammenhang *Master* genannt, das antwortende Gerät *Target*.

Um zu entscheiden, ob ein Gerät angesprochen wird oder nicht, verfügt jedes Gerät über eine Ansammlung von Adressfenstern, die *BARs* genannt werden. Die Geräte vergleichen die Adressen auf dem Bus mit den Bereichen der Fenster und melden sich, falls die gelesene Adresse in den Bereich eines solchen Fensters fällt. Jedes Gerät kann mehrere BARs unterschiedlicher Größe und Typs implementieren. Die genaue Lage der Adressfenster im physikalischen Speicherbereich wird jedoch

zum Zeitpunkt der Initialisierung des Busses durch das BIOS eindeutig vergeben und die Anfangsadressen werden in den BARs gespeichert.

Durch diesen Mechanismus kann jedes Gerät des Busses durch Schreibbefehle an bestimmte Adressbereiche angesprochen werden, welche letztlich einem BAR des Gerätes zugeordnet sind. Zusätzlich verfügt jedes Gerät über einen speziellen PCI-Konfigurationsbereich. In diesem sind Informationen zu finden, wie sich die Geräte dem Bus gegenüber repräsentieren. Um den Konfigurationsbereich ansprechen zu können gibt es ebenfalls spezielle Zyklen.

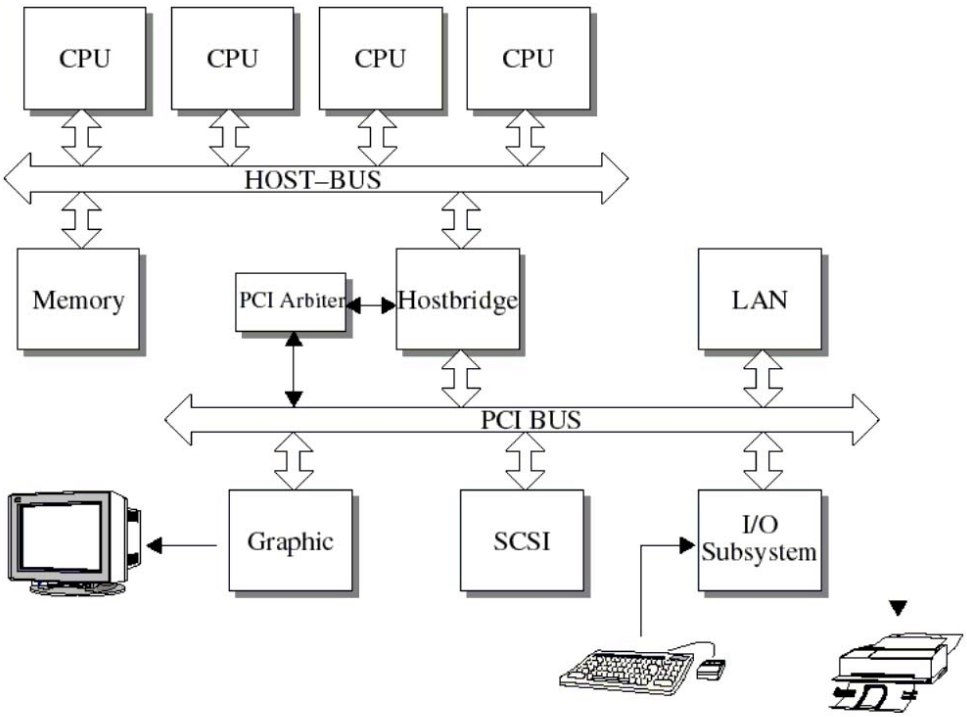


Abbildung 12: Architektur des PCI Busses. (Quelle: Vorlesung Prof. Lindenstruth, Uni-HD)

Anbindung des Cores

Bei dem zu der FPGA/PCI Karte bestellten Softwarecore von PLDA handelte es sich um einen sowohl Target- als auch Master-fähigen Core. Er kann sowohl in 33 MHz als auch in 66 MHz Bussen betrieben werden und unterstützt weiterhin 32-Bit und 64-Bit Betrieb. Einige weitere Eigenschaften sind Unterstützung von Erweiterungs-ROM, Ansteuerung für SRAM, Implementierung mehrerer Funktionen pro PCI-Gerät, Unterstützung von *Plug & Play* und Stromsparfunktionen. Der Core lag als VHDL-Datei vor und konnte somit für FPGAs und ASICs verwendet werden, wobei der verwendete VHDL-Code speziell für die Logikzellen des verwendeten FPGAs APEX 20K optimiert war. Der Core unterstützt die aktuelle PCI-Spezifikation 2.2 und ist damit für den Einsatz in allen PCI-Bussen geeignet. Weiterhin kann der Core auch in Abwandlungen von PCI wie *Compact-PCI* , *Mini-PCI* oder *Cardbus* verwendet werden, falls er mit einer entsprechenden Steckkarte zum Einsatz kommt. Weitere Eigenschaften finden sich im Datenblatt [Pld00].

Der Core lag verschlüsselt vor und mußte zunächst durch das Programm *PCI-Wizard* konfiguriert werden. Durch die weitere Verwendung des Programmes *ACF-Wizard* wurden die Signale des Cores an die richtigen Anschlüsse des FPGAs geleitet, welche weiterhin mit den zugehörigen Anschlüssen

des PCI-Steckers verbunden waren. Die durch den Wizard erzeugte unverschlüsselte VHDL-Datei *pcicore.vhd* repräsentierte den gesamten Core und konnte nun wie andere Komponenten auch von der Hauptkomponente eingebunden werden.

Der Core wurde zunächst so konfiguriert, daß vorhandene Speicherbereiche in BAR0, BAR1 und eine vorhandene ROM-Erweiterung angegeben wurden. Lese- oder Schreibzugriffe auf diese Speicherbereiche werden auf der Anwenderseite des Cores durch entsprechende Signale angezeigt. Der eingebunden Core wurde sodann in der Hauptkomponente mit den Multiplexern der Registerbank und der SRAM Komponente verbunden. Auf diese Weise kann mittels des PCI-Busses über das Adressfenster BAR0 auf die Registerbank und über das Adressfenster BAR1 und die ROM-Erweiterung auf das SRAM der Karte zugegriffen werden.

Damit waren in der Entwicklungsphase alle Funktionen der Karte über den PCI-Bus des Testrechners zu erreichen. In Abschnitt 6.10 werden die zu diesem Zweck unter Linux entstandenen Programme vorgestellt. Um die Einbindung des Cores zu testen können mit ihnen beispielsweise einzelne Speicherzyklen auf den PCI Bus gegeben werden. Weiterhin lassen sich Speicherbereiche anzeigen, beschreiben oder mit verschiedenen Bitmustern auf Zuverlässigkeit überprüfen. Auf diese Weise konnte auch die korrekte Datenübertragung von und zur CPU auf dem CPU-Modul überprüft werden.

Weitere Entwicklung

Um die Logik in der Hauptkomponente möglichst übersichtlich zu halten, wurde der Core zunächst lediglich mit Target-Funktionalität mit 32 Bit breiter Adressierung konfiguriert. Weiterhin wurde durch den verwendeten FPGA die Geschwindigkeit auf 33 MHz begrenzt. Die PCI-Spezifikation fordert jedoch, daß sowohl 32-Bit Karten in 64-Bit Bussen, als auch 33 MHz Karten in 66 MHz Bussen verwendet werden können. Im aktuellen Status kann der Prototyp der CIA-Karte damit auf allen Systemen getestet werden, die wenigstens einen 3,3V-PCI Bus für die FPGA-Karte zur Verfügung stellen.

Um auf den Hauptspeicher oder den I/O-Bereich des Rechners zugreifen zu können, muß selbst als Master auf den Bus zugegriffen werden. Ebenso müssen die einzelnen Geräte auf dem Bus sukzessive durch Konfigurationszyklen angesprochen werden um die im Wirtsrechner vorhandene Hardware zu ermitteln. Aus diesem Grund wird es zu einem späteren Zeitpunkt erforderlich sein, den Core als Master-fähig zu konfigurieren und entsprechend zu betreiben. In der Arbeit blieb dazu leider keine Zeit mehr.

4.5 Anbindung des SRAMs

Die FPGA-Karte ist mit 512 KByte SRAM bestückt, angeordnet zu einem 128 Bit breitem Datenbus und einem 15 Bit breiten Adreßbus. Zur Anbindung des Speichers wurde zunächst die Komponente *external_sram* erstellt, welche die eigentliche Ansteuerung der SRAM-Signale bewerkstelligt. Sie wird von der Hauptkomponente eingebunden und analog zur Ansteuerung der Registerbank mit einem Multiplexer verbunden. Die weiteren Komponenten sprechen das SRAM über diesen Multiplexer an.

Zunächst soll hier auf die Schwierigkeiten der Ansteuerung eingegangen werden.

Realisierung der Ansteuerung

Grundsätzlich gibt es zwei unterschiedliche Typen von SRAM: synchrones und asynchrones, wobei sich synchron hier auf einen gemeinsamen Takt bezieht. Synchrones SRAM hat also einen zusätzlichen Eingang, der mit dem Speichertakt versorgt wird. Alle Eingänge des SRAMs werden synchron zu diesem Takt gelesen und geschrieben, beim Lesen werden die Daten nach einer festgelegten Anzahl Takte auf dem Ausgang gültig. Beim asynchronen SRAM dagegen gibt es diesen Takt nicht. Beim Lesen wird eine Adresse an das SRAM gelegt und nach einer bestimmten *Zeit* können die Daten am Ausgang abgenommen werden. Das Schreiben verläuft analog, wobei mit Hilfe von diversen Steuerleitungen der Lesevorgang vom Schreibvorgang unterschieden wird. Da beim asynchronen SRAM kein Bezug zu einem festen Takt genommen wird, müssen beim Beschalten bestimmte Zeiten beachtet werden, die nicht über- oder unterschritten werden dürfen.

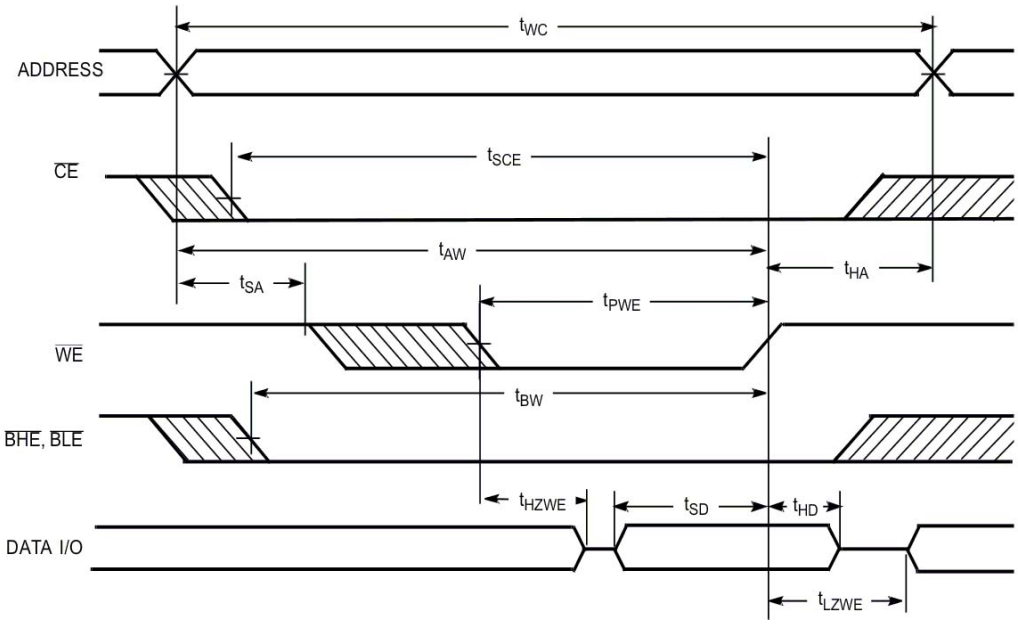


Abbildung 13: Timing des verwendeten SRAM Bausteines bei einem Schreibzyklus (Quelle: Datenblatt)

Bei dem vorhandenen SRAM handelte es sich um asynchrones SRAM. Der verwendete PCI-Core verfügt bereits über einen eigenen SRAM-Controller, der bei Zugriffen auf BARs im PCI-Gerät entsprechende Steuersignale für ein SRAM bereitstellt. Damit wäre es im Prinzip möglich gewesen, das SRAM direkt an den Core anzuschließen, wie auch im mitgelieferten Tutorial zum Core angegeben. Die einzelnen Steuerleitungen vom Core zum SRAM werden jedoch während des Designflows in den Schritten *Place & Route* abhängig vom restlichen Design unterschiedlich innerhalb des Verbindungsnetzwerkes verlegt. Daher sind die Signallaufzeiten ebenfalls unterschiedlich und abhängig vom restlichen Design. Das Problem war, daß die Signale aus dem Core nicht aus Registern kommen, sondern selbst schon untereinander einen Zeitversatz haben. Wenn die Wege zu lang werden, kommen die Signale zu unterschiedlichen Zeiten am SRAM an und die Ansteuerung stimmt nicht mehr. In diesem Zusammenhang haben sich speziell die Adresssignale als kritisch erwiesen. Als Ergebnis stellte sich heraus, daß bei einer direkten Verbindung von Core und SRAM ohne zwischengeschaltete Register kein zuverlässiger Betrieb möglich ist.

Aus diesem Grund wurden Register zur Ansteuerung des SRAMs verwendet. Ihre Aufgabe ist es, alle Signale zum SRAM synchron zum globalen Takt zu speichern, um sie dann beim nächsten Takt mit festem Bezug untereinander zum SRAM durchzureichen. Die Unterschiede durch die verschiedenen Laufzeiten wurden so minimiert. Bei einem Lesebefehl können die Daten nach Anlegen der Adresse an der nächsten Taktflanke abgegriffen werden. Bei einem Schreibbefehl muß zusätzlich ein kurzer Schreibpuls auf der *we*-Leitung des SRAMs erzeugt werden, da Daten bei steigender Flanke dieses Signals in das SRAM übernommen werden. Falls der PCI-Bus in einem Burst bei jeder Taktflanke in Folge Daten schreibt, so muß auch nach jeder Flanke dieser Puls erzeugt werden [Ang].

Durch die Vorschaltung von Registern kann das asynchrone SRAM nun wie ein synchrones SRAM durch das restliche Design verwendet werden. Zu beachten ist, daß nun beim Lesen die gültigen Daten erst einen globalen PCI-Takt später an der Datenleitung anliegen. Aus diesem Grund wird in diesem Fall das *ws*-Signal des Cores verwendet, um auf dem PCI-Bus einen *Waitstate* genannten Wartetakt zu erzwingen. Die momentane Implementierung mit 32 Bit breite und 33 MHz PCI-Takt gestattet damit eine theoretische maximale Datenübertragung von 132 MByte/Sek.

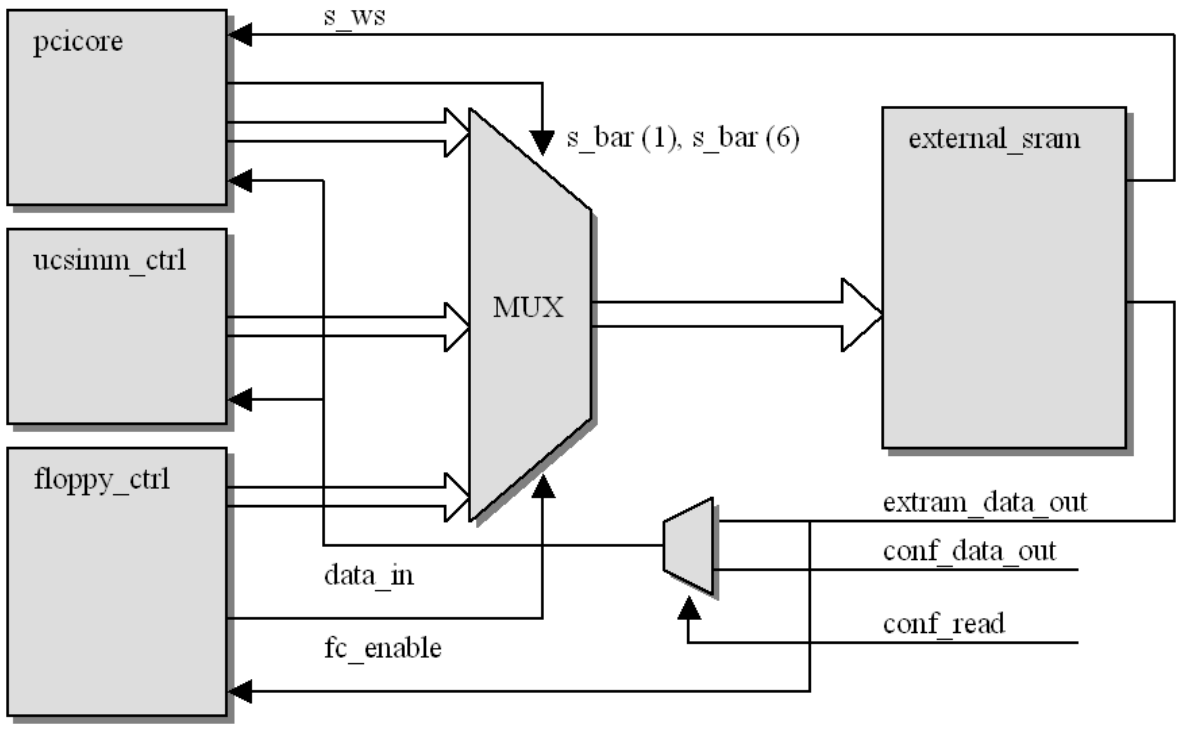


Abbildung 14: Datenpfade bei der Ansteuerung der SRAM-Komponente

Verwendung

Bisher wird die Ansteuerung mit 32 Bit Datenbreite betrieben, was 128 KByte des Speichers nutzbar macht. Schreib- und Lesebefehle betreffen daher immer vier Byte simultan. Die Ansteuerung des SRAMs wurde ebenfalls über einen Multiplexer in der Hauptkomponente angeschlossen. Auf diese Weise kann das SRAM über drei Wege angesprochen werden:

- Über den PCI-Bus. Dies geschieht analog zur Nutzung der Registerbank, jedoch wird nun das PCI-Adressfenster *BAR1* verwendet. Zum Test können die unter Linux auf dem Testrechner entstandenen Programme genutzt werden. Die gesamten verwendeten 128 KByte des SRAMs können angesprochen werden.

  Bei der Emulation der VGA-Karte wird ein vermeintliches VGA-BIOS ebenfalls bereitgestellt, in dem über den PCI-Bus auf den Speicherbereich ab Adresse 0x10000 des SRAMs zugegriffen und 32kByte Erweiterungs-ROM eingeblendet wird. Zu diesem Zweck wird nicht der Adressbereich BAR1, sondern BAR6 des Cores verwendet, was der ROM-Erweiterung von PCI-Geräten entspricht.
- Durch die Programme auf dem CPU-Modul mit Hilfe des Steuerbusses über die Adapterplatine. Die Programme auf der CPU können dadurch ebenfalls das gesamte SRAM ansprechen und das VGA-ROM Abbild oder die Daten einer Diskettenspur schreiben.
- Durch die Komponenten, welche zu ihrer Funktion auf größeren Speicher angewiesen sind. Dies betrifft momentan nur die Floppy-Komponente. Bei einem Zugriff des Wirtsrechners auf seine vermeintliche Floppy werden die Daten der emulierten Diskette im SRAM ab Adresse 0 gespeichert.

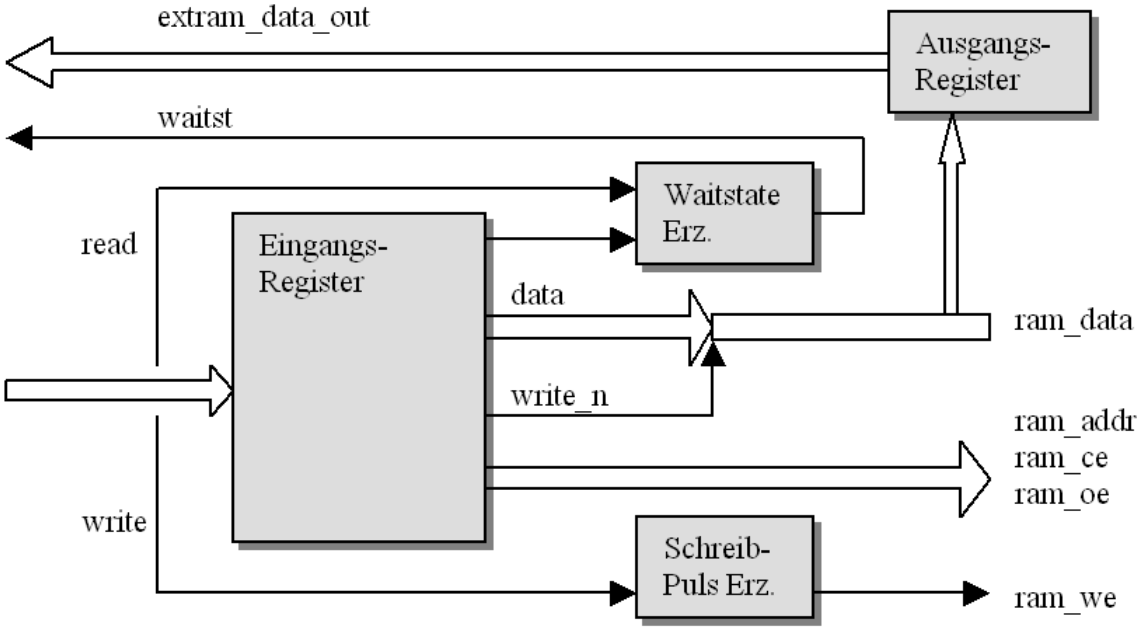


Abbildung 15: Schematische Darstellung der SRAM Ansteuerung durch external_sram

Zu beachten ist, daß der Zugriff auf das SRAM ähnlich dem Zugriff auf die Registerbank nur durch einen Multiplexer realisiert wird und daher nicht durch einen speziellen Mechanismus koordiniert

wird. Im Falle eines gleichzeitigen Zugriffs entstehen daher ebenfalls fehlerhafte Daten für mindestens eine der beteiligten Komponenten. Die momentane Implementierung reicht jedoch aus, um bereits die Funktionen der einzelnen Teile zu testen. Bei der Verwendung der Floppy-Emulation, greifen die Steuerprogramme auf dem CPU-Modul und die Komponente zur Emulation der Floppy nur alternierend aber nie gleichzeitig auf das SRAM zu.

Um die Anbindung des SRAMs zu testen, wurde das Programm *Ramtest* geschrieben und sowohl für das Linux-System des Testrechners als auch für das µCLinux-System auf dem CPU-Modul angepaßt. Dadurch kann die Zuverlässigkeit des Zugriffs auf das SRAM sowohl über den PCI-Bus als auch über den Verbindungsbus über die Adapterplatine vom CPU-Modul getestet werden. Das Programm wird in Abschnitt 6.5 beschrieben.

Weitere Entwicklung

Bei einem Ausbau der Karte werden möglicherweise weitere Komponenten auf das SRAM zugreifen müssen. In diesem Falle muß eine spezielle SRAM-Ansteuerung implementiert werden. Zugriffe von verschiedenen Bereichen werden dann koordiniert der Reihe nach abgearbeitet und weitere zugreifende Komponenten müssen solange warten. Die Entwicklung einer solchen Ansteuerung ist eine noch zu lösende Aufgabe und muß zu einem späteren Zeitpunkt dem Projekt hinzugefügt werden.

Bisher werden nur 128 KByte des SRAMs genutzt. Um die vollen 512 KByte nutzen zu können, müßte die Breite des Datenbusses im gesamten Design auf 128 Bit erhöht werden. Dies würde auch einen größeren Multiplexer nach sich ziehen und einiges mehr der verfügbaren Ressourcen im FPGA verbrauchen. Bisher werden jedoch nur 32 KByte für das Erweiterungs-ROM der VGA-Karte und etwa 24 KByte für die Speicherung einer Diskettenspur benötigt. Möglicherweise könnten in dem freien Platz weitere Spuren der Floppy untergebracht werden. Aber auch die volle Nutzung des SRAMs bietet nicht genug Platz, um die *gesamte* Diskette mit ca. 2 MByte Daten im SRAM abzulegen. Man könnte allerdings einige Spuren vorsorglich zwischenspeichern um die CPU auf dem CPU-Modul zu entlasten.

Im Rahmen der Verbesserung der SRAM-Ansteuerung könnte auch die Ansteuerung auf 128 Bit erweitert und ein globaler Datenpfad von 32 Bit beibehalten werden. In diesem Falle müßte die Adresse entsprechend umgesetzt und mit den jeweiligen *ce*-Signalen der SRAM-Bausteine sichergestellt werden, daß bei einem Schreibzyklus nur die betroffenen 32 Bit geändert werden.

4.6 Anbindung der CPU

Die CPU wurde auf der Karte für Steuerungsfunktionen benötigt, die zu komplex für einen FPGA sind. Sie übernimmt vor allem auch die Anbindung an das Kontrollnetzwerk, über das die Karte und damit letztlich der Wirtsrechner gesteuert werden kann. Durch die Verwendung des µCSimm-Modules war das zum Betrieb nötige RAM, ROM und auch das Netzwerkinterface schon vorhanden. Um die eigentliche Steuerfunktion übernehmen zu können, mußte zunächst der Zugriff auf weitere Komponenten der Karte hergestellt werden.

Die Verbindung der Programme auf dem CPU-Modul zum FPGA wurde mit einen Datenbus über die Adapterplatine hergestellt. Während die Programme die entsprechenden Portregister der CPU beschreiben wird die Ansteuerung des Busses auf FPGA-Seite von der Komponente *µcsimm_ctrl* übernommen. Die Komponente dekodiert die Signale des Busses und liefert auf der anderen Seite die gelesenen Kommandos in Verbindung mit den Adressen und Daten. Ähnlich der Funktion des PCI-

Cores wird sie von der Hauptkomponente eingebunden und mit den beiden Multiplexern für die Ansteuerung der Registerbank und des SRAMs verbunden. Auf diese Weise kann die gesamte Funktionalität der Karte von der CPU genutzt werden.

Die Hardware des Datenbusses

Die CPU war Teil des µCSimm-Modules, welches über seinen Stecker nur eine begrenzte Anzahl von Anschlüssen zur Verfügung stellte. Von der CPU waren 2 frei beschaltbare Ports zu jeweils 8 Bit für eigene Zwecke verwendbar: Port C, der eigentlich für die Ansteuerung einer im Projekt nicht vorhandenen LCD Anzeige gedacht war, und Port D mit weiteren 8 Leitungen, die auch zum Auslösen von Interrupts konfiguriert werden konnten. Zwei weitere Leitungen waren mit den Signalen für einen Timer und eine Watchdog-Funktion belegt, konnten aber ebenfalls frei verwendet werden. Schließlich verfügte die CPU über einen mit drei Leitungen realisierten schnellen seriellen Bus, einen SPI Master (engl: *serial periphal interface*). Dieser konnte mit bis zu 4 MBit/sec betrieben und intern mit 16 Bit Wortbreite angesteuert werden.

Um eine einfache und doch schnelle Übertragung zu erhalten wurde sich dafür entschieden, den seriellen SPI-Bus zunächst nicht zu verwenden, sondern eine parallele Übertragung zu nutzen. Zu diesem Zweck wurde der freie LCD-Port C für 8 parallele Datenleitungen verwendet, während zur Steuerung weitere vier Leitungen des Ports D benutzt wurden. Alle 12 Leitungen wurden über ein Kabel an die Adapterplatine geführt. Über ein weiteres Kabel wurden diese Signale dann von dort an den freien Stecker J5 auf der FPGA-Karte und damit an den FPGA angeschlossen.

Eine Tabelle mit den beschalteten Anschlüssen des FPGAs und der CPU findet sich in Anhang B.

Das Protokoll

Im folgenden wird das implementierte Protokoll vorgestellt. Zum weiteren Studium sei auf die Quelldateien zur FPGA-Komponente *µcsimm_ctrl* oder der Bibliothek *fpga_lib.c* für das µCSimm verwiesen.

Für die Kommunikation wurde ein sogenanntes *Handshake*-Protokoll mit den Signalen Request (*Req*), Acknowledge (*Ack*) und Clock (*Clk*) implementiert. Diese Signale entsprechen Steuerleitungen zur CPU auf dem µCSimm bzw. Register in der Registerbank im FPGA-Design. Während ein Programm auf der CPU die entsprechenden Bits der I/O-Ports der CPU ansteuert, werden die Signale von der Komponente im FPGA dekodiert. Zusätzlich kann der aktuelle Zustand der Signale beispielsweise über den PCI Bus auch in der Registerbank gelesen werden.

Die Signale *Req* und *Ack* sind aktiv im Sinne des Namens bei einem logischen 0-Zustand. Im Ruhezustand des Busse sind sie daher auf logisch 1. Die Übertragung von Daten erfolgt in einzelnen Zyklen und wird von der CPU aus gestartet: Über das Setzen des *Req*-Signales zeigt die CPU an, daß sie einen Zyklus beginnen möchte. Der FPGA bestätigt mittels des *Ack*-Signales. Die Übertragung erfolgt nun taktweise zum Signal *Clk,* welches von der CPU gesetzt wird. Daten werden von der CPU oder dem FPGA auf den Datenbus geschrieben, worauf die CPU die *Clk* Leitung einmal auf 1 und wieder auf 0 setzt. Gültige Daten liegen bei steigender Flanke der *Clk*-Leitung an.

- Die CPU schreibt zunächst ein Kommandobyte. Dieses Kommando gibt an, ob sich der Zyklus auf das SRAM oder die Registerbank bezieht und ob es sich um einen Lese- oder Schreibzugriff handelt. Zugriffe auf das SRAM werden im Folgenden als *Speicherzugriffe*

bezeichnet, bei Zugriffen auf die Registerbank handelt es sich aus Sicht der CPU um *Konfigurationszyklen* des FPGA-Designs. Die Aufschlüsselung der Kommandos zu den geschriebenen Werten kann dem Quelltext entnommen werden.

- Im nächsten Takt wird eine Adresse geschrieben. Bei einem Speicherzyklus muß diese Adresse 32 Bit breit sein, d.h. es werden noch 3 weitere Takte Adressbytes geschrieben. Bei einem Konfigurationszyklus war dies eine Registeradresse und es wird sofort mit der Datenphase fortgefahren. Zu beachten ist, daß das SRAM auf der Karte mit 512 KByte nur 17 Bit breite Adressen besitzt, von denen momentan nur 15 genutzt werden. Ein großer möglicher Adressbereich kann jedoch organisatorische Vorteile haben, die Breite wurde deshalb analog zum PCI Bus zu 32 Bit gewählt.
- In der nun folgenden Datenphase werden die Daten nacheinander bei jedem Takt von *Clk* vom jeweiligen Sender auf den Datenbus gelegt. Der FPGA arbeitet dabei immer mit steigender Flanke von *Clk*. Die wesentlich langsamere CPU schreibt Daten daher vor der Flanke bei Clock logisch 0 und liest Daten nach der Flanke bei Clock logisch 1. Bei Speicherzugriffen wird bei jedem Takt automatisch die Adresse für die SRAM-Ansteuerung inkrementiert. Dadurch können ohne erneute Angabe der Adresse fortwährend Daten in das RAM geschrieben oder aus dem RAM gelesen werden. Der Zyklus wird von der CPU beendet, in dem sie das anfangs gesetzte *Req*-Signal wieder zurücknimmt. Die Komponente im FPGA bestätigt daraufhin mit einer Rückname des *Ack*-Signals.
- Durch das zwischenzeitliche Zurücknehmen des *Ack*-Signales durch die Komponente im FPGA kann der Zyklus eine Zeit lang unterbrochen werden. Mit diesem Prinzip könnte beispielsweise eine Logik zur SRAM-Ansteuerung den Bus in einen Wartezustand versetzen, falls gerade durch andere Komponenten auf das SRAM zugegriffen wird. Wird das *Ack*-Signal nicht nach einer gewissen Zeit wieder gegeben, so wird die Verbindung abgebrochen und die zum Test geschriebenen Programme auf dem µCSimm geben momentan eine entsprechende Meldung aus.

Bei der aktuellen Implementierung wird ein Datenzyklus grundsätzlich von der CPU aus gestartet. Für die Emulation einiger Peripheriegeräte ist es jedoch erforderlich, auf bestimmte Anfragen vom Wirtsrechner entsprechend zu reagieren. Beispielsweise muß bei einem Spurwechsel der Floppy die Übertragung der nächsten Spur veranlaßt werden. Zu diesem Zweck wurde eine vierte Leitung auf CPU-Seite als Interruptleitung konfiguriert. Die CPU reagiert auf diese Interrupts, in dem sie mittels Konfigurationszyklen den Grund in der Registerbank nachfragt. Näheres zu diesem Punkt siehe Abschnitt 6.8.

Implementierung

Die Übertragung erfolgt durch die besprochenen Signale zunächst unabhängig vom globalen PCI-Takt. Aus diesem Grund müssen die Eingangssignale von der Komponente erst in Registern gespeichert werden, damit sie synchron verwendet werden können.

Das Protokoll besteht aus einer Abfolge von einzelnen Zuständen, in denen Kommando, Adresse und Daten übertragen werden. Aus diesem Grund wurde die Implementierung mittels eines speziellen Signales realisiert, welches verschiedene Zustände (Werte) annehmen kann. Man spricht hier von einer sogenannten *State-Maschine*. Dabei wird in dieser Variable der aktuelle Zustand (engl. *state*) gespeichert. Abhängig von den Signalen *Clk* oder *Req* geht ein Zustand in einen Folgezustand über.

Je nach Zustand und vorher gelesenem Kommando werden nun Steuersignale für die Multiplexer zu SRAM oder die Registerbank erzeugt. Wie oben bereits erwähnt wird über das übergebene Kommando die Art der Übertragung bestimmt. Bei SRAM-Zugriffen werden fortlaufend 4 Datenbyte gelesen oder geschrieben, während bei Zugriffen auf die Registerbank nur ein Byte übertragen wird. Weil die internen Datenpfade des Designs 32 Bit Breite haben, muß die Komponente auch die notwendigen Byte-Enable Signale für die Ansteuerung des betroffenen Bytes in der Registerbank erzeugen. Bei SRAM-Zugriffen wird in der Datenphase mit jedem Takt automatisch die Adresse zum SRAM erhöht.

Einige der Zustände und internen Register der State-Maschine wurden zur Fehlersuche auf Register ab Adresse 0x10 in der Registerbank abgebildet und können dort beispielsweise über den PCI Bus gelesen werden. Eine Übersicht über die State-Maschine gibt folgendes Diagramm:

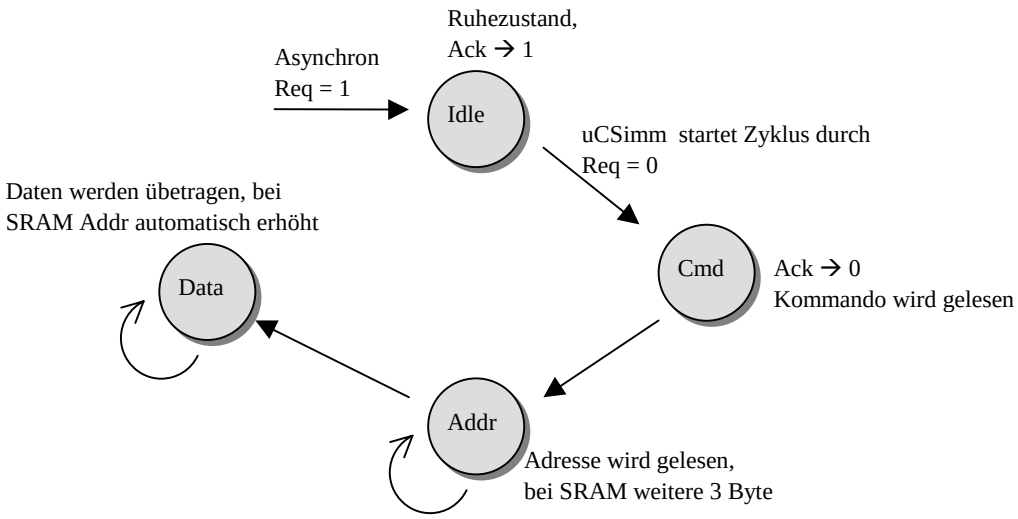


Abbildung 16: Zustandsmaschine beim Datenaustausch mit der µCSimm-CPU

Zur einfachen Verwendung des Busses durch die Programme auf dem CPU-Modul wurde die Funktionsbibliothek *fpga_lib* erstellt, welche von den entwickelten Testprogrammen eingebunden wird. Eine genauere Beschreibung erfolgt weiter unten im Abschnitt 6.4.

Weitere Entwicklung

Die Geschwindigkeit der Schnittstelle bei Zugriffen auf das SRAM wurde zu ca. 100 KByte/Sekunde bestimmt. Dies ist ausreichend für die Verwendung mit der zu emulierenden Floppy, die bisher als einziges Gerät eine zeitkritische Komponente darstellt. Es existieren jedoch einige Möglichkeiten die Geschwindigkeit weiter zu erhöhen, falls dies im Laufe der Zeit erforderlich werden sollte.

Zum einen könnte man die Hardware verbessern. Die CPU ist momentan lediglich mit 15 MHz getaktet, der FPGA verwendet jedoch den mehr als doppelt so schnellen PCI-Takt mit 33 MHz. Mit der aktuellen Implementierung wird die Datenübertragungsrate daher durch die Geschwindigkeit der CPU beschränkt. Eine erste Maßnahme könnte daher sein, die CPU durch ein schnelleres Pendant zu

ersetzen. Tatsächlich gibt es auf dem Markt eine baugleiche CPU mit 33 MHz. Für die Entwicklung des Prototyps war jedoch die integrierte Funktionalität durch das µCSimm ausschlaggebend, welches leider nur mit der halb so schnellen Version geliefert wurde. Wahrscheinlich bietet aber schon die Software Möglichkeiten der Verbesserung, beispielsweise wenn die zeitkritische Schleife der Übertragung direkt in Assembler und nicht in C programmiert wird.

Eine andere Möglichkeit wäre die noch freien Pins des µCSimms zu nutzen, um den Bus breiter als 8 Bit zu gestalten. Es ist jedoch möglich, daß in Zukunft noch einige weitere Pins für andere Zwecke benötigt werden. Bei ausgeschaltetem Wirtsrechner soll später die CPU mit Strom aus einem Akku versorgt werden, der FPGA jedoch nicht. Anschlüsse an Hardwarekomponenten, die auch bei ausgeschaltetem Rechner funktionieren sollen, müssen deshalb direkt mit der CPU verbunden werden, beispielsweise sind jetzt schon die Leitungen, um den Rechner ein- und auszuschalten oder einen Reset auszuführen, direkt am µCSimm und nicht am FPGA angeschlossen.

Schließlich konnte der SPI-Bus des Prozessors aus Zeitgründen nicht mehr getestet werden. Möglicherweise kann die Implementierung einer SPI-Komponente im FPGA eine schnellere Datenübertragung sogar mit nur drei Signalen ermöglichen, da der SPI-Bus direkt vom Prozessor unterstützt wird. SPI arbeitet mit maximal 4 Mbit/sec was 500 KByte/sec entspricht. Mit einem Zyklus kann dabei jeweils ein 16-Bit Wort übertragen werden, d.h. nach jeweils 2 Byte muß etwas Zeit aufgewendet werden einen neuen Zyklus zu starten. Trotzdem ist aber eine Steigerung gegenüber der momentanen 100 KByte/sec zu erwarten.

Im weiteren Verlauf des Projektes ist jedoch geplant, die CPU als eine synthetisierbare Komponente direkt in den FPGA zu integrieren. Es gibt bereits Firmen, die solche Prozessoren als Komponenten für FPGAs anbieten, als Beispiel sie hier die *Excalibur/Nios*-Linie der Firma *Altera* genannt. Dadurch würde sich die Verbindung einfach als ein Datenbus im FPGA realisieren lassen.

4.7 PS/2 Emulation

Heutige Rechner verfügen zum Anschluß von Eingabegeräten meist über einige sogenannte PS/2 Schnittstellen, an denen in der Regel Tastatur und Maus angeschlossen werden. Zur Steuerung des Wirtsrechners sollten diese Geräte selbst emuliert werden, als ob sie am PS/2 Steckplatz vorhanden wären. Der Betrieb der Geräte unterscheidet sich dabei nur in den Daten, welche über diese Schnittstelle gesendet werden. Dadurch reichte für die Emulation dieser Geräte die Entwicklung einer einzigen Komponente aus.

In der Arbeit wurde zu diesem Zweck die Komponente *ps2* entwickelt. Sie wurde von der Hauptkomponente eingebunden und kann von der CPU auf dem CPU-Modul über die zugehörigen Register in der Registerbank angesprochen werden. Die elektrischen Verbindung an den Wirtsrechner wurden über die Adapterplatine hergestellt.

Zum Verständnis der Funktionsweise wird an dieser Stelle zunächst das Protokoll grob erklärt. Eine detailliertere Darstellung findet sich in der Literatur [Cha99].

Das PS/2 Protokoll

Der PS/2 Bus dient zum Anschluß von seriellen Geräten mit niedriger Bandbreite an einen Rechner und besteht lediglich aus drei Leitungen: einer Datenleitung, einer Taktleitung sowie einer elektrischen Masseleitung. Er ist ein schönes Beispiel für eine universelle bidirektionale Übertragung. Sowohl der bestückte Rechner als auch das Gerät können Daten versenden, wobei der Rechner

zusätzlich immer die Möglichkeit zu einem Verbindungsabbruch hat. Das kleine Protokoll beinhaltet sogar eine minimale Fehlererkennung.

Beide Leitungen werden über *Pull-up* Widerstände auf logisch 1 gehalten und können von beiden Seiten gelesen oder auf logisch 0 gebracht werden (vgl. Abschnitt 5.4). Die Kommunikation geschieht in einzelnen Zyklen, welche vom Rechner oder dem Gerät gestartet werden können. Zwischen diesen Zyklen ruht der Bus, d.h. beide Leitungen liegen auf logisch 1. Das Gerät erkennt einen Verbindungsaufbau des Rechners, indem dieser die *Takt*leitung einige Zeit auf 0 hält. Der Rechner wiederum erkennt an der auf 0 gehaltenen *Daten*leitung einen Verbindungsaufbau des Gerätes.

Bei einem Zyklus wird grundsätzlich ein Byte übertragen. Die Daten werden dabei zunächst in ein 11 Bit großes Päckchen gepackt, ein sogenannter *Frame*, welcher aus einem Startbit (logisch 0), den 8 Datenbits (beginnend mit Bit 0), einem ungeraden Paritätsbit (*xnor* über die 8 Datenbits, entspricht logisch 1 bei gerader Anzahl von 1-Bits) und einem Stopbit (logisch 1) besteht. Bei der Übertragung vom Rechner zum Gerät antwortet das Gerät am Schluß mit einem 12. Bit zur Bestätigung (logisch 0). Dieser Frame wird nun vom Sender bitweise auf die Datenleitung gegeben und vom Empfänger mit festem Bezug zum Takt gelesen. Der Rechner liest und schreibt immer bei fallender Flanke der Taktleitung, während das Gerät bei steigender Flanke arbeitet. Der Takt wird immer vom Gerät erzeugt und läuft mit fester Frequenz im Bereich zwischen 10 und 15 kHz, der Rechner kann eine Verbindung jederzeit abbrechen, indem er die Taktleitung auf Masse hält.

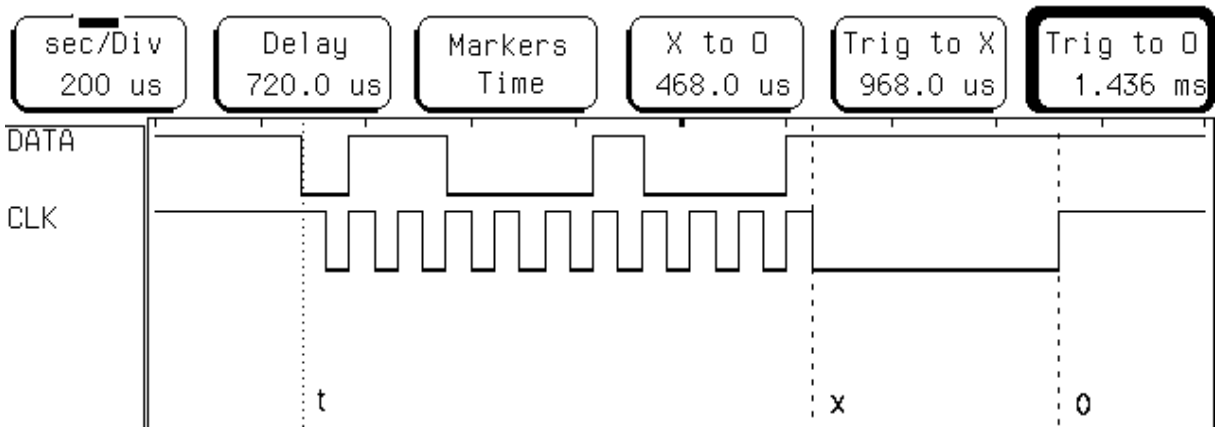


Abbildung 17: Timing Diagramm: PS/2 Komponente emuliert Tastatur und sendet Taste „d“

Implementierung

Um ein PS/2 Gerät zu emulieren, mußte zunächst das oben beschriebene Protokoll als Komponente im FPGA implementiert werden. Dabei wurde analog zur Anbindung an die Steuer-CPU eine State-Maschine konstruiert. Es wurde sowohl das Senden als auch das Empfangen von Daten implementiert. Die Komponente wird von der Hauptkomponente sowohl für die Emulation der Tastatur als auch der Maus eingebunden. Die entsprechenden Register wurden auf die Adressen ab 0x30 bzw. 0x34 in der Registerbank abgebildet, wo sie von einer Anwendung auf der CPU angesprochen werden können.

Die Komponente erhielt zur Steuerung durch die CPU eine Schnittstelle, an der ein Datenbyte übergeben und gelesen werden kann, sowie die Kontrollsignale, *Req*, *Ack, Error* und *Reset*. Ein

gesetztes Bit entspricht einem aktiven Signal. Über das Signal *Irqreq* wird die Interruptleitung zur CPU auf dem CPU-Modul angesprochen, falls Daten empfangen worden sind.

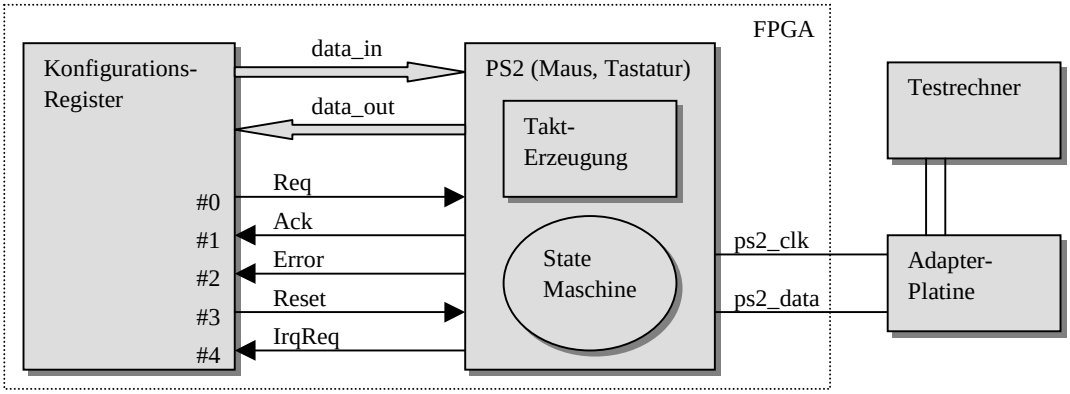


Abbildung 18: Schematische Darstellung und Signale der PS/2-Emulation

Da der PS/2-Takt immer vom Gerät erzeugt wird, muß die Komponente zunächst einen eigenen Takt aus dem PCI-Takt generieren. Zu diesem Zweck wird ein Zähler verwendet (vergleiche auch Abschnitt 4.10). Das letztlich erzeugte Taktsignal besitzt eine Frequenz von 10,8 kHz, was im Rahmen der PS/2-Spezifikation liegt. Ein PS/2 Takt dauert damit 92 µsek.

Zum Senden von Daten schreibt die CPU das Datenbyte in das Register in der Registerbank und setzt das *Req*-Bit. Dadurch geht die State-Maschine in den Sendemodus über. Falls die Signale des Busses nicht beide auf 1 sind, wird der Bus gerade verwendet und es wird ein Fehler über das *Error*-Bit gemeldet. Ansonsten wird das Datenbyte in einem gültigen Frame bitweise zum Rechner übertragen. Während der Verbindung wird ein möglicher Abbruch durch den Rechner abgefragt. Am Ende der Übertragung wird auf die Bestätigung des Rechners gewartet und wieder in den Ruhezustand der State-Maschine übergegangen. Bei einem auftretenden Fehler oder einem Ausbleiben der Bestätigung wird das *Error*-Signal gesetzt. Die CPU kann am *Ack*-Bit eine erfolgreiche Übertragung ablesen oder am *Error*-Bit einen Fehler erkennen. Durch das *Reset*-Bit kann die State-Maschine notfalls zurückgesetzt werden.

Der Empfang verläuft analog. Falls die Komponente einen Verbindungsaufbau durch den Rechner registriert, liest sie das Datenbyte bitweise ein. Am Ende der Verbindung wird für drei PS/2-Takte das *Irqreq*-Bit und damit das Interruptsignal gesetzt, um die CPU auf dem CPU-Modul zu informieren.

Beim Senden wurde auf dem verwendeten Testrechner beobachtet, daß zumindest unter den Betriebssystemen Windows und Linux vom Rechner in regelmäßigen Abständen (ca. 8 mal in der Sekunde) Signale auf den PS/2 Bus gesendet werden, indem die Taktleitung für ca. 240 µsek auf Masse gezogen wird. Diese Signale entsprechen jedoch keinem Sendeversuch nach dem PS2-Protokoll. Möglicherweise versuchen die Systeme auf diese Weise auf elektrischem Wege neue angeschlossene Geräte zu erkennen (Stichwort: *Plug & Play* Fähigkeit). Dieser Effekt verhindert jedoch, daß die Komponente in diesem Moment selbst Daten an den Rechner senden kann. Sie meldet in diesem Fall einen Fehler mittels des *Error*-Signales und die verwendende Software muß einen erneuten Sendeversuch unternehmen.

An dieser Stelle sei bemerkt, daß sich die Bedeutung der einzelnen Datenbytes bei der Verwendung durch Tastatur und Maus unterscheidet. Die Tastatur sendet Tasten-*Scancodes*, während die Maus je nach Typ 3-Byte Pakete oder 4-Byte-Pakete mit Positionsangaben und Informationen über gedrückte Tasten verschickt. Auf diesen Punkt wird in Abschnitt 6.8 im Rahmen der Beschreibung der erstellten Software näher eingegangen.

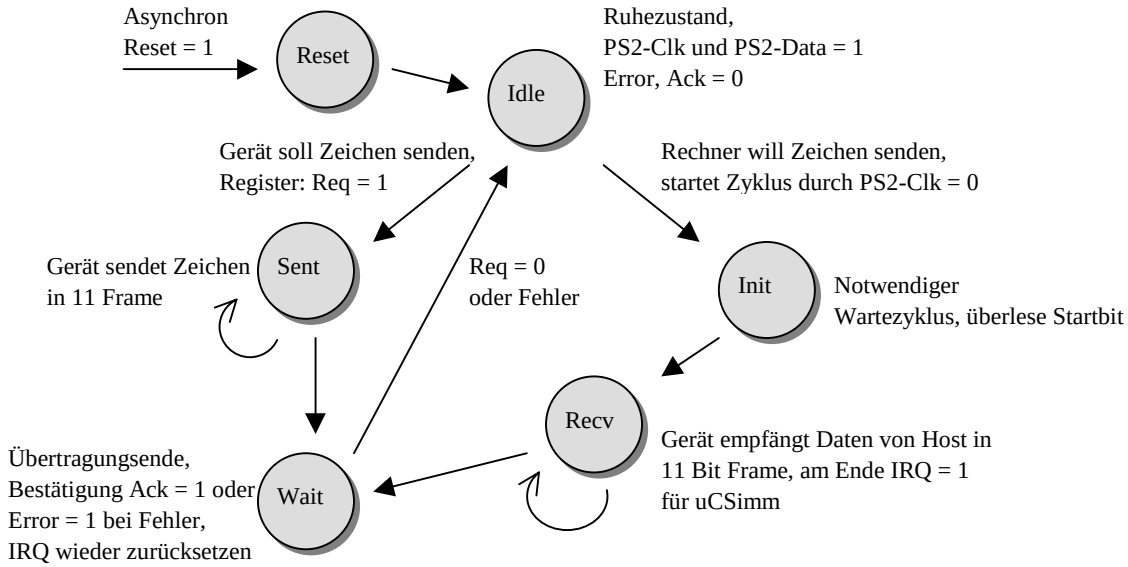


Abbildung 19: Zustandsmaschine bei der Emulation des PS/2 Gerätes

Mögliche Alternativen zu PS/2

Das PS/2 Protokoll ist schon etliche Jahre alt und wird in heutigen PCs nur für die langsamen Eingabegeräte verwendet. Verschiedene Organisationen bemühen sich um eine Ablösung von PS/2 hin zu einem einheitlichen schnellem Standard für serielle Peripheriegeräte mit langsamer und mittlerer Geschwindigkeit. PS/2 unterstützt auch nicht wirklich die automatische Erkennung von eingesteckten Geräten (*Plug-and-Play*) sowie den Austausch von Geräten während der Laufzeit des Rechners (*Hot-Plugging*). An dieser Stelle soll deshalb noch kurz auf mögliche Alternativen zu PS/2 eingegangen werden.

An erster Stelle sollen hier USB genannt werden (eng: *universal serial bus*). USB wurde von einer Kooperation von namhaften Herstellern aus der IT-Industrie wie *Intel*, *Hewlett-Packard*, *Microsoft*, *Net* etc. Mitte der 90er Jahre entwickelt. Ziel war es, einen neuen Standard für die Verbindung von externen Geräten an PCs zu schaffen, der die vielen anderen Steckertypen wie Parallelport, serielle Schnittstelle, Maus, Tastatur, Joystick, Modem etc. mit der Zeit ablösen sollte. Dabei wurde zunächst Wert auf eine leichte Bedienbarkeit gelegt um die Plug-and-Play Fähigkeiten der aktuellen Betriebssysteme mit allen Geräte nutzen zu können. Weiterhin sollte USB eine Leistungssteigerung darstellen, sowohl in Bezug auf Bandbreite, als auch auf Einsetzbarkeit. So lassen sich an einem USB Port mittels eines Hubs insgesamt bis zu 127 Geräte gleichzeitig betreiben. Für USB gibt es inzwischen schon ein Vielzahl von Geräten. Außer Eingabegeräten gibt es auch immer mehr Multimediageräte wie Webcams, Camcorder, sogar Festplatten sind schon für USB verfügbar und lassen sich leicht extern an den PC anschließen. Aufgrund seiner sprichwörtlichen Universalität findet man USB schon bei Geräten, die nicht direkt etwas mit PCs zu tun haben. Die Bandbreite hat sich mit

der USB-Spezifikation 2.0 mittlerweile von ursprünglich 1.5 Mbit/sec über 12 Mbit/sec auf heute 480 Mbit/sec. gesteigert.

Da die Spezifikation von USB sehr komplex ist (die Version 2.0 umfaßt über 600 Seiten) wurde bisher von einer Implementierung innerhalb des Projektes Abstand genommen. Möglicherweise läßt sich mit den am Markt erhältlichen USB-Chips eine preisgünstige Emulation von Peripheriegeräten wie Maus und Tastatur mit vertretbarem Aufwand durchführen. Dies könnte sich insofern als notwendig erweisen, als die heute überall verbreiteten PS/2 Verbindungen mit der Zeit gegenüber USB in den Hintergrund treten werden. Eine Beschreibung von USB findet sich in der Literatur [Usb00].

Eine weitere Alternative für eine universelle Geräteverbindung wäre noch der parallel zu USB zunächst von Apple entwickelte Bus IEEE 1394 (*FireWire*). Er beherrscht mit bis zu 1600 Mbit/sek deutlich höhere Datenraten, hat jedoch bisher jedoch nicht die Verbreitung von USB erlangt [Usb01].

4.8 Floppy Emulation

Zum Konzept der Arbeit gehörte die Möglichkeit, das Startverhalten des Rechners durch die Emulation einer Floppy zu realisieren. Dies sollte ebenfalls durch die direkte Emulation des Gerätes selbst geschehen, welches über ein Kabel an den entsprechenden Platz auf der Hauptplatine des Rechners angeschlossen werden sollte.

In der Arbeit wurde eine solche Komponente erstellt. Sie wird ebenfalls über die Hauptkomponente eingebunden und kann über die Registerbank angesteuert werden. Die Verbindung zum Rechner erfolgt über die Adapterplatine.

Zum Verständnis des Aufbaus muß kurz auf die in PCs verwendete Floppy eingegangen werden. Eine genauere Beschreibung der Floppy, sowie der Organisation der Daten einer Diskette und des Aufzeichnungsformates MFM findet sich in der Literatur [Haa96].

Die Floppy im PC

Die Floppy ist das Lesegerät für Disketten im PC. Zunächst ist zu bemerken, daß die heutige im PC verwendete Floppy selbst nur wenig Logik enthält. Sie besteht im Wesentlichen aus einem Schreib- und Lesekopf für je eine Diskettenseite und einem Steppermotor, um die einzelnen Spuren der Diskette anfahren zu können. Die Ansteuerung der Köpfe und des Motors wird dabei direkt durch einen *Floppy-Controller* auf der Hauptplatine des Rechners übernommen. Als Rückmeldung von der Floppy werden Signale für das Erkennen von Diskettenwechseln, Schreibschutz, Feststellung der Position einer Indexmarke usw. verwendet. Für das eigentliche Lesen und Schreiben von Daten gibt es zwei separate Datenleitungen auf dem Kabel, deren Signale direkt über die Schreib/Leseköpfe auf die Diskette aufgezeichnet werden. Um eine Floppy zu emulieren, ist es also erforderlich, sich im Wesentlichen mit dem Aufzeichnungsformat der Diskette zu beschäftigen.

Bei einer Diskette handelt es sich um ein zylindrisches, magnetisches Medium, das etwa mit einer Frequenz von 5 Hz um seine Mittelachse rotiert. Um sich auf der drehenden Diskette orientieren zu können, befindet sich an einer ganz bestimmten Stelle ein sogenanntes *Indexloch*, das mit einer Lichtschranke geprüft wird. Jedesmal, wenn das Loch sich auf der drehenden Scheibe unter der Lichtschranke hindurchbewegt, erkennt der Controller anhand des Signals den Beginn der Spur.

Die Daten einer heutigen PC-Diskette sind in zwei Seiten mit meist 80 zyklischen Spuren à 18 Sektoren zu je 512 Byte aufgeteilt. Zwischen den eigentlichen Datensektoren befindet sich noch eine nicht unerhebliche Menge Daten zur Synchronisation, Füllzeichen und zur Fehlerbehandlung. Aus diesem Grund steigt die Datenmenge pro Spur und Seite von 18 Sektoren zu 512 Byte, also 9 KByte auf 12.790 Byte an. Die gesamte Diskette enthält damit 2.046.400 Byte Daten, davon sind 1.474.560 Byte Nutzdaten.

Die Daten werden in Form von Flußwechseln der Magnetisierung auf die Diskette aufgezeichnet. Bei diesem Vorgang muß sichergestellt werden, daß unabhängig von der Beschaffenheit der Daten und möglichen Gleichlaufschwankungen des Motors dieselben wieder eindeutig gelesen werden können. Zu diesem Zweck werden sie vor der Aufzeichnung mit einem speziellen Verfahren kodiert und mit einer festen Frequenz auf die Diskette aufgetragen. Die Kodierung wurde von Zeit zu Zeit weiterentwickelt und hat außerdem den Zweck, die Anzahl der Flußwechsel pro aufgezeichnetem Bit zu minimieren, um die Datendichte bei gleichen Medien und beschränkter Aufzeichnungsfrequenz zu erhöhen. Aktuell wird bei Standard-PC Floppys eine Modifikation der *MFM* Kodierung verwendet. Die Hauptaufgabe bei der Emulation einer Floppy besteht also in der richtigen Kodierung der Daten in die entsprechenden Flußwechsel. Diese werden dann auf das Kabel zum Floppy-Controller gegeben.

Das Aufzeichnungsformat MFM

Bei der MFM-Kodierung wird ein Bit durch ein Signal von 2 µsec Dauer dargestellt. Zur Kodierung werden am Beginn oder in der Mitte des Bits Flußwechsel in der Magnetisierung eingebaut, um aufeinanderfolgende 0 und 1-Bits unterscheiden zu können. Da der Lesekopf auf Induktion basiert, finden sich auf dem Floppykabel bei solchen Flußwechseln 1-Impulse wieder, ansonsten ist der Pegel logisch 0. Ein Impuls selbst hat etwa die Länge von 360 µsek.

Die Aufzeichnung geschieht nach folgender Regel:

- Ein 1-Bit wird durch einen Impuls in der Mitte des Bits gekennzeichnet.
- Ein 0-Bit durch einen Impuls zu Beginn des Bits, wenn zuvor ebenfalls ein 0-Bit gesendet wurde. Kein Signal, falls zuvor ein 1-Bit gesendet wurde.
- Zusätzlich wird in der Floppy noch eine Modifikation der MFM-Kodierung verwendet: Bei der Kombination 1-0-0-0-0-1 wird der Impuls in der Mitte, also bei der dritten 0, weggelassen. Diese Ausnahme wird bei der Übertragung der Bytes 0xc2 und 0xa1 in den Sektorheadern zur Synchronisation der Controller-Logik verwendet.

Bei Folgen von reinen 0-Bits oder Folgen von reinen 1-Bits wird demnach alle 2 µsek ein Impuls gesendet. Bei einem Wechsel von 0 auf 1 oder 1 auf 0 liegen 3 µsek zwischen den Impulsen und bei der Kombination 1-0-1 liegen 4 µsek zwischen den Impulsen. Zusätzlich gibt es die erwähnte Ausnahme.

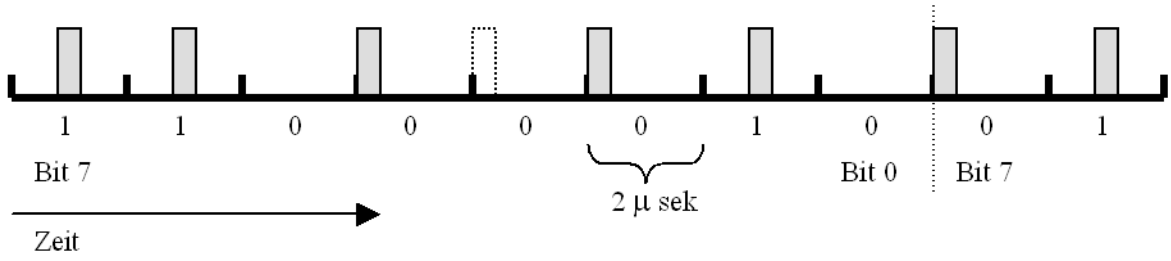


Abbildung 20: Bitsequenz beim Übertragen des Wertes C2, das Auslasen des mittleren Impulses dient der Synchronisation des Controllers.

Implementierung

Bei der Implementierung wurde auf die Arbeit von Martin Kirsch zurückgegriffen, der eine erste funktionsfähige Komponente für die Floppy-Emulation erstellt hat. Speziell die Erkennung von Spurwechseln über die *Step*-Leitungen und besonders die Implementierung der MFM-Kodierung wurde von Martin Kirsch erarbeitet. Die Komponente wurde etwas überarbeitet und um die Möglichkeit erweitert, die zu schreibenden Daten aus dem SRAM zu lesen.

In der Arbeit wurde lediglich eine Lesefunktion implementiert. Daten können also nur *zu* dem Controller auf der Hauptplatine geschickt werden. Dies entspricht einer schreibgeschützten Diskette. Auch wird bisher nur das 1.44-MByte Format (80 Spuren, 2 Seiten, 18 Sektoren zu 512 Byte) unterstützt, das aber heute das Standard-Diskettenformat ist und von allen Rechnern gelesen werden kann.

Die Komponente *floppy_ctrl* ist auf der einen Seite über die Adapterplatine mit dem Floppy-Controller des Wirtsrechners verbunden. Auf der anderen werden Steuersignale an den Multiplexer für das SRAM gesendet, um die Daten der vermeintlichen Diskette aus dem SRAM zu laden. Zur Steuerung der Komponente durch die Programme auf dem CPU-Modul werden weitere Signale in der Registerbank ab Adresse 0x20 abgebildet. Die Komponente bindet selbst zwei weitere Komponenten zur Erkennung der Spurwechsel und der Kodierung in das MFM-Format ein. Die gesamte Koordination wird wiederum über eine State-Maschine gesteuert.

Erkennung von Spurwechseln

Die eingebundene Komponente *floppy_track* emuliert die Arbeit des Steppermotors. Sie ist mit den externen *Step* und *Step-Dir* Leitungen verbunden und speichert die aktuell angefahrene Spur in einem eigenen Register. Jedesmal, wenn der Controller den Steppermotor eine Spur weiter bewegen möchte, werden die Signale von der Komponente ausgewertet und das Register wird inkrementiert oder dekrementiert. Auf einer eigens dafür vorgesehenen Leitung *Track0* wird dem Controller zurückgemeldet, ob sich der virtuelle Kopf auf der untersten Spur befindet oder nicht. Damit hat dieser eine Möglichkeit, sich auf der Diskette zu orientieren. Die Auswertung der Steppersignale wurde aus der Arbeit von Martin Kirsch übernommen.

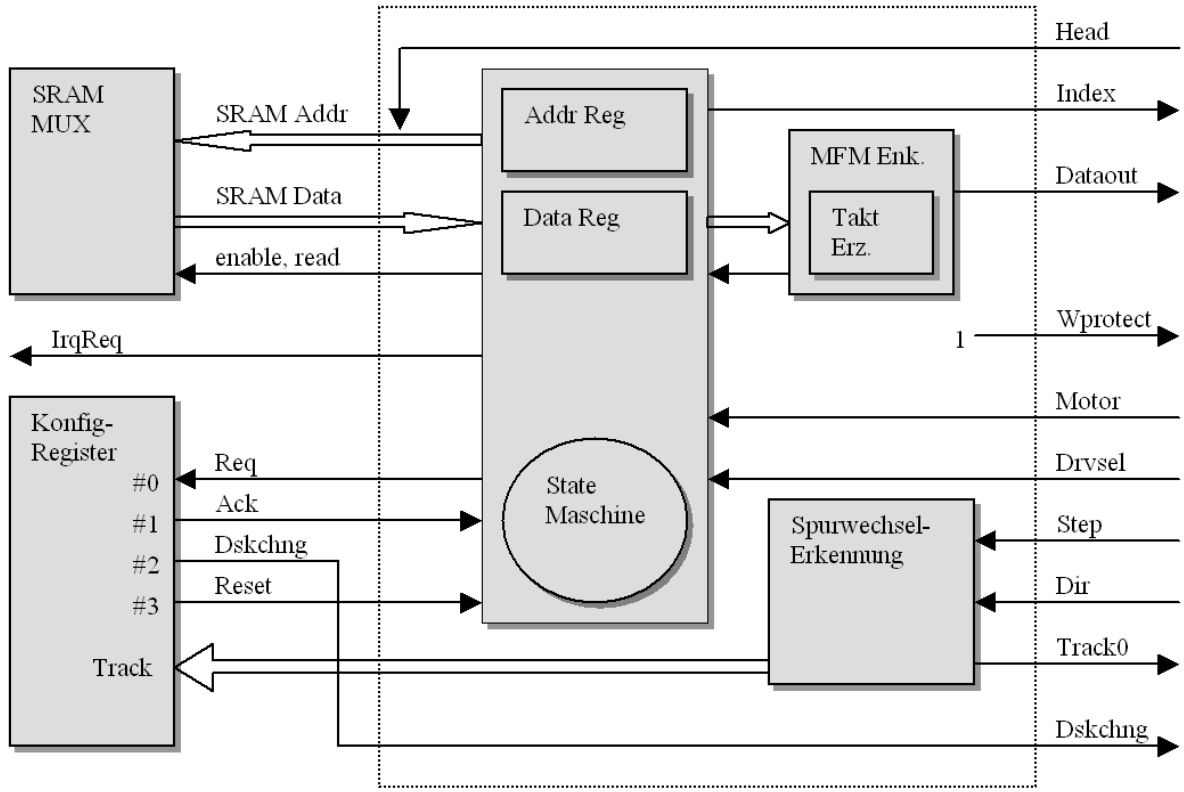


Abbildung 21: Schematische Darstellung der Floppy-Komponente

Datenkodierung

Die ebenfalls eingebundene Komponente *floppy_mfm* übernimmt die Kodierung der Daten in das MFM-Format wie sie auf der Diskette vorliegen und sendet die zugehörigen Impulse mit der richtigen Frequenz auf die Datenleitung zum Controller. Diese Komponente entspricht in etwa einem Lesekopf.

Zur Kodierung wird die Dauer eines Bits in 16 gleichlange Teile unterteilt, die jeweils die Länge von 4 PCI-Takten also 120 nsek haben. Ein Impuls zu Beginn oder in der Mitte eines Bits wird dabei durch Halten der Datenleitung von 3 Teilen auf logisch 1 erzeugt, also 360 nsek. Anschließend folgen 5 Teile logisch 0, also 600 nsek. Falls zu Beginn oder in der Mitte des Bits kein Impuls gegeben werden soll bleibt die Leitung 8 Teile auf logisch 0. Ein Bit kommt damit auf die Zeit von 1,92 µsec, was nicht exakt der Spezifikation entspricht, aber von allen getesteten Controllern problemlos akzeptiert wurde und die einfache Erzeugung über die bereits vorliegende PCI-Clock erlaubt.

Die Komponente übernimmt die Daten als 32 Bit langes Wort und gibt ein Signal zurück, wenn das darauf folgende Wort geschrieben werden kann. Die Kodierung des MFM-Formates wurde ebenfalls zuerst von Martin Kirsch implementiert [Kir].

Zusammenspiel

Bei einem Zugriff auf die Floppy aktiviert der Controller zunächst den Motor und Schreib/Lesekopf und fährt mit dem Steppermotor die gewünschten Spur an. Da sich anschließend der Kopf über der sich drehenden Diskette befindet wartet der Controller auf das Indexsignal und dann auf den eigentlich zu lesenden Sektor.

Die Komponente erkennt dies durch ein logisches 0 auf den Leitungen *Motor* und *Select*. Die aktuelle Spur wird von der Komponente *floppy_track* in Register 0x28 der Registerbank übertragen. Die State-Maschine löst über die *Irqreq*-Leitung einen Interrupt auf der CPU aus. Die Software auf der CPU erkennt am *Req*-Bit im Floppy-Register einen Floppy-Zugriff als Ursache des Interrupts, liest die geforderte Spur aus Register 0x28 und überträgt diese in das SRAM. Da nicht die gesamte Diskette in das SRAM paßt, werden immer nur die beiden Seiten einer Spur dort gespeichert. Diese Daten beinhalten sowohl die 18 Sektoren als auch die zwischen den Sektoren auf der Disketten enthaltenden Steuerinformationen. Nach der Übertragung setzt die CPU das *Ack*-Bit im Floppy-Register.

Die State-Maschine liest nun die Daten der gesamten Spur kontinuierlich mit der Datenfrequenz der Diskette aus dem SRAM aus und sendet diese an die Komponente *floppy_mfm* zur Kodierung, welche den kodierten Bitstrom zum Floppy-Controller schickt. Alle 64 µsek wird also ein neues Datenwort vom SRAM geladen. Das Indexsignal wird dabei gesetzt, falls aus den ersten 128 Byte der Spur Daten gesendet werden. Der Zugriff auf das SRAM wird solange fortgesetzt, bis die Floppy vom Controller deaktiviert wird oder der Controller die Spur wechselt. Falls kein Zugriff auf die Floppy mehr stattfindet, werden auch keine Daten mehr aus dem SRAM ausgelesen.

Zusätzlich stellt die Komponente im Kontrollregister ein *Dskchng*-Bit und ein *Reset*-Bit zur Verfügung. Wird das *Dskchng*-Bit gesetzt und wieder gelöscht, so wird eine neue Diskette emuliert. Ein permanentes Setzen dieses Bits repräsentiert eine nicht eingelegte Diskette. Das Reset-Bit setzt die gesamte Komponente zurück, wenn es kurz gesetzt wird. Zu Testzwecken wurde die Möglichkeit eingebaut, die aktuelle Spur auf der 7-Segment-Anzeige der FPGA-Karte anzeigen zu lassen. Diese Funktion kann über Register 0x0c in der Registerbank ausgewählt werden.

In der Arbeit wurde ein Testprogramm für die Floppy-Komponente erstellt, welches die Spurdaten aus einem vorher erzeugten Diskettenabbild verwendet. Es überwacht Spurwechsel, schreibt die Inhalte der angefahrenen Spuren in das SRAM und gibt die entsprechenden Spurnummern auf dem Bildschirm aus. Das Programm wird in Abschnitt 6.8 beschrieben.

Weitere Entwicklung

Zusätzlich zu den Daten der Sektoren werden auch die Steuerinformationen dazwischen übertragen. Eine Möglichkeit wäre, diese Steuerinformationen gleich automatisch von einer eigenen Komponente im FPGA erstellen zu lassen. Da sie aber auch etwas komplexere Dinge wie CRC-Prüfsummen enthalten wurde zunächst davon abgegangen. Die Diskettenabbilder müssen mit den Steuerinformationen im Vorfeld mittels eines extra Programms erstellt werden, wodurch die Daten einer 1,4 MByte Diskette auf die Größe von 2 MByte anwachsen. Die Erstellung einer solchen Komponente ist jedoch für die weitere Entwicklung des Projektes insofern sinnvoll, als die konkrete Aufzeichnung der Diskettendaten auf der Diskette eigentlich ein Hardwaredetail ist und sich die CPU nicht damit beschäftigen müssen sollte.

Desweiteren sei bemerkt, daß die Floppy-Komponente an den Multiplexer zum SRAM angeschlossen wurde. Bereits bei der Diskussion der SRAM-Anbindung in Abschnitt 4.5 wurde das Problem des gleichzeitigen Zugriffs durch mehrere Komponenten erwähnt. Dies ist jedoch im vorgesehenen

Betrieb nicht der Fall, da CPU und Floppy-Komponente abwechselnd auf das SRAM zugreifen. Eine Weiterentwicklung mittels einer koordinierten Ansteuerung des SRAMs in Verbindung mit einem FIFO für die Diskettendaten wäre aber sinnvoll.

Weiterhin könnte man sich überlegen, verschiedene Diskettenformate oder Aufzeichnungsformate zu emulieren, was aber eher akademischen Anspruch hat. Zwar können einige BIOSse bereits 2.88 MB-Disketten lesen (mit 36 Sektoren anstelle 18 pro Seite), aber das Format wird heute praktisch nicht benutzt. Die Startdateien der diversen Betriebssysteme passen in der Regel auf 1.44 MB-Disketten und laden dann gegebenenfalls größere Komponenten über ein Netzwerk nach.

4.9 POST / Port 0x80 Statusnummern

Bei der Darstellung des Konzeptes der Arbeit wurden bereits in Abschnitt 2.10 die POST-Statusnummern vorgestellt, mit deren Hilfe sich während des Startvorganges des Wirtsrechners eine einfache Möglichkeit der Fehlersuche bietet. In der Arbeit wurde eine Komponente erstellt, welche diese Nummern parallel zum PCI-Core auf dem Bus mitlesen kann. In Abschnitt 4.9 wird die erstellte Software beschrieben, um die Nummern anzeigen zu können.

Implementierung

Die Schreibbefehle der Statusnummern beziehen sich auf den I/O-Speicherbereich. In der Regel wird der I/O-Port 0x80 für diese Zwecke verwendet. Ein für I/O-Speicher konfiguriertes BAR des Cores wird beim Systemstart vom BIOS jedoch an eine *beliebige* Startadresse konfiguriert. Um explizit Schreibbefehle auf *bestimmte* I/O-Adressen lesen zu können, muß daher parallel zum Core auf den PCI-Bus zugegriffen werden. Das reine Lesen stellt kein Problem dar, da keine Signale auf dem Bus gestört werden.

Die Komponente ist auf der einen Seite mit dem PCI-Bus verbunden. Auf der anderen Seite werden die mitgelesenen Nummern in der Registerbank in der Adresse 0x38 abgebildet. Die zu überwachende I/O-Adresse kann in den Registern 0x3a und 0x3b eingestellt werden. Zusätzlich wurde die Möglichkeit eingebaut, den Wert auf der 7-Segment-Anzeige der FPGA-Karte anzeigen zu lassen, was im Register 0x0c eingestellt werden kann.

Um diese Aufgabe zu erfüllen, werden die Signale des PCI-Busses gelesen, die für die Identifikation und Steuerung eines Zyklusses nötig sind. Dabei handelt es sich um die PCI-Signale *CBE, FRAME, TRDY, IRDY, STOP*, sowie die Adress/Datensignale *AD*. Um das Design stabiler zu machen, wurden alle Signale zunächst durch ein Register geleitet.

Um den Status auf dem Bus zu speichern, wurde wiederum eine State-Maschine konstruiert, die sich zunächst im *Idle*-Zustand befindet. Bei fallender Flanke von *FRAME* und einem Schreibkommando auf die gewählte I/O-Adresse geht sie im nächsten Takt in den *Read*-Zustand über. In diesem Zustand wird die vom BIOS geschriebene POST-Nummer vom Bus gelesen. Im nächsten Takt wird auf die Beendigung des Zyklusses gewartet, worauf die Maschine wieder in den Anfangszustand zurückkehrt. Die gelesene Nummer wird im genannten Register in der Registerbank abgelegt. Die Komponente erfaßt dabei ständig geschriebene POST-Nummern, in dem Register befindet sich also immer die zuletzt geschriebene Nummer.

Innerhalb des Projektes war es wichtig zu zeigen, daß ein Zugriff auf den PCI-Bus am Core vorbei realisiert werden kann, weil daraus ein erster Schritt zur Machbarkeit der VGA-Emulation folgte. Sie

konnte im Projekt aber leider nur im Ansatz gelöst werden. Eine genauere Beschreibung der Problematik bei der Emulation einer VGA-Karte erfolgt in Abschnitt 4.11.

Weitere Entwicklung

Eine andere Möglichkeit der Realisierung könnte sein, ein freies BAR des Cores zu verwenden. Dies könnte schon bei der Synthese des Designs fest auf beispielsweise I/O-Port 0x80 konfiguriert werden, um dann Schreibbefehle auf das verwendete BAR abzufragen. Dann wäre aber die betroffene Adresse nachträglich nicht mehr veränderbar. Der Core bietet jedoch die Möglichkeit, sich auch nachträglich durch spezielle Steuersignale innerhalb des Designs konfigurieren zu lassen (*backend core reconfiguration*). Dies müßte zunächst getestet werden.

Die Komponente speichert momentan nur den jeweils letzten gelesenen POST-Wert ab. Als mögliche Verbesserung könnte man einen FIFO-Puffer einbauen, in dem aufeinanderfolgende Nummern zwischengespeichert werden. In diesem Zusammenhang könnte man überlegen, einen Interrupt auf der CPU auslösen zu lassen sobald der Puffer fast gefüllt ist. Andererseits ist prinzipbedingt nur der jeweils letzte POST-Wert von Interesse, da er den aktuellen Zustand des Selbsttestes anzeigt.

Abschließend ist zu bemerken, daß mit der Methode natürlich generell Schreibbefehle an ausgewählte I/O-Adressen überwacht werden können. Man könnte die Komponente auch zum Mitlesen von weiteren Arten von Zyklen wie Schreib/Lesebefehle an physikalische Speicheradressen ausbauen und so ansatzweise die Funktionalität eines PCI-*Tracers* implementieren. Der erreichbare Nutzen in Bezug auf die Steuerung und Überwachung eines Rechners zu Administrationszwecken sei jedoch dahingestellt.

4.10 Sonstige Komponenten

Schließlich enthält das Projekt noch einige weitere Komponenten, die an dieser Stelle kurz vorgestellt werden sollen.

Hexdekoder

Der Hexadezimaldekoder übernimmt die Ansteuerung einer 7-Segment-Anzeige.

Die Komponente enthält als Eingang einen 4-Bit breiten Wert und sorgt für eine hexadezimale Anzeige desselben. In der Hauptkomponente werden 2 dieser Module jeweils für die vier unteren und oberen Bits eines 8 Bit breiten Wertes eingebunden. Auf diese Weise können alle 256 verschiedenen Zustände von 0x00 bis 0xff angezeigt werden. Im Register 0x0c der Registerbank kann die Quelle des Wertes eingestellt werden, momentan die aktuelle Spur des Schreib- und Lesekopfes der emulierten Floppy oder die gelesenen POST Werte. Die Komponente entspricht im Wesentlichen einer Tabelle und wurde einem Beispieldesign der Entwicklerkarte entnommen.

Digifilter

Die Komponente implementiert einen digitalen Filter und wird benutzt, um Störungen bei Flankenwechseln in externen Eingangssignalen herauszufiltern.

Flankenwechsel werden nur akzeptiert, wenn sie eine bestimmte Mindestzeit stabil sind. Die Komponente besteht im Wesentlichen aus drei Registern, welche hintereinander in Reihe geschaltet

sind. Das externe Signal liegt auf dem Eingang der Komponente und perkuliert bei jedem Takt des globalen PCI-Taktes ein Register weiter. Nur wenn der Inhalt aller drei Register gleich ist, hat sich das externe Signal 90 nsek lang nicht geändert und wird entsprechend übernommen. [Ang].

Anlaß der Entwicklung war, daß bei der Übertragung von der CPU auf dem CPU-Modul zum FPGA festgestellt wurde, daß sich die Signale auf den externen *Req* und *Clk*-Leitungen gegenseitig induktiv stören. Aus diesem Grund brach die Übertragung immer wieder ab. Erst nachdem beide Signale zunächst durch die digitale Filterkomponente geleitet wurden, funktionierte die Übertragung dauerhaft stabil. Die Schaltung wird außerdem zur Filterung des *Step*-Signales und des *Select*-Signales der emulierten Floppy verwendet, da sich auch hier einige Instabilitäten zeigten, die nach Einbau verschwanden. Die Filterkomponente sollte damit bei allen asynchronen Eingangssignalen verwendet werden, die eine triggernde Funktion haben, wie beispielsweise zum Zurücksetzen der State-Maschine in der Floppy- oder in der CPU-Komponente.

Die Schaltung verzögert natürlich das eigentliche Signal um 3 Takte, also um 90 nsek. Sie kann damit nur eingesetzt werden, wenn die Störungen wesentlich kürzer als 90 nsek sind und das eigentliche Signal nicht zeitkritisch ist und wesentlich länger Stabil bleibt, um es nicht zu sehr zu verfälschen.

Bei den restlichen Eingangssignalen der Floppy ist die Flanke nicht zeitkritisch und die Signale bleiben lange Zeit stabil. Bei der PS/2 Komponente haben sich bisher keine Instabilitäten gezeigt, aber die Komponente könnte möglicherweise vorsorglich noch in beide Leitungen eingebaut werden. Weiterhin wird von den Datenleitungen der Verbindung zur CPU gelesen. Vor ihnen ist die Komponente insofern nicht erforderlich, da die Daten erst gelesen werden, wenn von der CPU auf der *Clk*-Leitung ein Signal gegeben wurde. Bis dahin sind eventuelle Störungen jedoch längst abgeklungen.

Erzeugung langsamerer Takte aus dem PCI-Takt

Viele Peripheriegeräte wie die Floppy oder die PS/2 Schnittstelle arbeiten mit eigenen, ganz bestimmten Taktfrequenzen, die vom im Design verwendeten PCI-Takt von 33 MHz verschieden und meist wesentlich langsamer sind. Bei der Emulation dieser Geräte müssen diese Takte aus dem PCI-Takt erzeugt werden. In diesem Zusammenhang sind einige Dinge zu beachten, die an dieser Stelle erwähnt werden sollen, da bei der Fortführung des Projektes wahrscheinlich auch weitere Geräte mit unterschiedlichen Takten emuliert werden sollen.

Ein langsamerer Gerätetakt kann leicht aus dem PCI-Takt mit Hilfe eines Zählers erzeugt werden. Immer wenn dieser einen bestimmter Wert annimmt wird das neue Taktsignal negiert und der Zähler zurück auf null gesetzt. Letztlich läuft der neue Takt um den Faktor *zwei mal Zähler* langsamer. Es wäre nun naheliegend, die gesamte Logik, die das entsprechende Gerät emulieren soll, mit diesem neuen Takt zu versorgen, ihn also auf die *clk*-Eingänge der betroffenen Register zu legen, also etwa nach folgendem Schema in VHDL:

```
process (clk)
variable counter : std_logic_vector(3 downto 0);
begin
     if rising_edge(clk) then
          counter := unsigned(counter) +1;
          if unsigned(counter) = 10 then
               new_clock <= not new_clock;
               counter <= (others => '0');
          end if;
     end if;
end process;
```

```
process (new_clock)
begin
     if rising_edge(new_clock) then
     ..
     ..
     ..
     end if;
end process;
```

Das Problem dabei ist, daß Register die durch den PCI-Takt getaktet werden und Register die durch den neuen Takt getaktet werden, ein wenig zeitlich versetzt sind - und zwar etwa um die Schaltzeit eines Gatters von wenigen nsek, da der neue Takt aus dem PCI-Takt selbst erst erzeugt werden muß. Dieser Zeitversatz (engl *clock skew*) sorgt nun für Probleme, wenn zwischen zwei solcher Register Daten ausgetauscht werden. Wenn ein durch den PCI-Takt getaktetes Register Daten anlegt und ein zweites welches die Daten übernehmen soll nur diese kurze Zeit später geschaltet wird, so liegen die Daten womöglich zu kurz an um stabil eingelesen werden zu können. In diesem Fall spricht man von einer Verletzung der sogenannten Setup-Zeit T_{su}.

Um dies zu umgehen, werden im Projekt an den betreffenden Stellen auch weiterhin grundsätzlich *alle* Register durch den Chipweiten dedizierten PCI-Takt *clk* getaktet. Der benötigte langsamere Takt wird zwar weiterhin durch einen Zähler realisiert, doch jedesmal wenn der Zähler einen bestimmten Stand hat wird nun ein Signal auf logisch 1 gesetzt, welches ansonsten auf logisch 0 liegt. Dieses Signal bekommt z.B. den Namenssuffix *Event.* Damit wird an den entsprechenden Registern im VHDL-Quellcode die Abfrage auf eine steigende Flanke lediglich um eine Abfrage auf einen Signalzustand erweitert.

```
process (clk)
variable counter : std_logic_vector(4 downto 0);
begin
     if rising_edge(clk) then
          counter := unsigned(counter) +1;
          if unsigned(counter) = 20 then
               new_clock_event <= '1';
               counter <= (others => '0');
          else
               new_clock_event <= '0';
          end if;
     end if;
end process;

process (clk)
begin
     if rising_edge(clk) and new_clock_event = '1' then
     ..
     ..
     ..
     end if;
end process;
```

Bemerkung: Durch diese Schaltung muß der betroffene Teil des Designs mit schnellem Takt laufen können, obwohl eigentlich nur ein langsamer Takt benötigt würde (*gated clock*). Daraus erfolgt auch ein höherer Stromverbrauch !

Eine andere Möglichkeit wäre gewesen, den gewünschten Takt mit einem sogenannten PLL zu erzeugen (engl: *phase locked loop*), die auf dem verwendeten APEX-FPGA auf der Entwicklerkarte bereits enthalten sind. Die PLLs haben genau die Aufgabe, Takte aus bereits vorhandenen

unterschiedlich schnellen Takten zu erzeugen. Der genaue Taktversatz kann dabei in bestimmten Bereichen eingestellt werden, insbesondere kann er auch null sein. Der neu erzeugte Takt könnte dann auch in den speziellen *global clock* Leitungen auf dem Chip verlegt werden, um an alle Register möglichst gleichzeitig anzukommen. Da PLLs nicht auf allen FPGA-Chips vorhanden sind, hätte dies jedoch die Portabilität des VHDL-Codes eingeschränkt.

4.11 VGA Emulation

Wie bei der Erläuterung des Konzeptes besprochen, war die erfolgreiche Emulation der CIA-Karte als eine im PCI-Bus eingesteckte VGA-Karte ein wichtiger Anforderungspunkt des Projektes. Leider konnte diese Aufgabe im Rahmen der Arbeit nicht mehr vollständig bearbeitet werden, die prinzipielle Machbarkeit wurde jedoch gezeigt.

Zunächst sollen die Probleme, die bei der Emulation auftraten, genauer erläutert werden:

- Die PCI Entwicklungskarte muß die Möglichkeit besitzen, sich auf dem PCI Bus als vermeindliche VGA-Karte zu erkennen zu geben. Dadurch wird der Rechner veranlaßt, entsprechende Befehle auf dem Bus an die Karte zu senden.
- Die spezifischen Befehle und Adressbereiche müssen dekodiert werden können, um die an die VGA-Karte gerichteten Anweisungen wie Einstellung eines Bildschirmmodus oder auch nur die Ausgabe eines Zeichens korrekt zu registrieren.
- Die Inhalte der beschriebenen Bildseiten müssen gespeichert werden. Möglicherweise müssen auch einige der VGA I/O-Register mit minimaler Funktionalität implementiert werden.

Im folgenden wird auf die Problemstellung im Einzelnen eingegangen:

Identifikation

VGA-Karten geben sich im *Class-Code* Register im Konfigurationsbereich des PCI-Cores als *Display/VGA* zu erkennen (Wert 0x030000). Dies reicht jedoch nicht aus. Der Rechner erwartet außerdem noch ein gültiges VGA-ROM über das *ExpROM*-BAR, welches er bei der Initialisierung der Karte zu der Adresse 0xc0000 kopiert und dort startet.

Versuche, einen Rechner mit einer eingesteckten Karte mit korrekter VGA-Kennung aber ohne VGA-ROM zu starten scheiterten an einer Fehlermeldung des Rechners über den Lautsprecher. Dieser setzte den Startvorgang aber in der Regel fort. Bei einem Rechner wurde jedoch beobachtet, daß keine Bildschirmausgaben mehr auf den PCI-Bus gesendet wurden, in der Annahme, keine Grafikkarte zu besitzen. Ein anderer Rechner beschwerte sich zwar ebenfalls per Piepton, sendete aber die Ausgaben aber dann doch.

Das ROM beinhaltet spezielle Initialisierungsfunktionen, beispielsweise um die Funktionen zur Bildschirmausgabe des Rechners auf die Karte anzupassen. In den Versuchen wurde eine Minimalversion eines solchen BIOSses [Pl86] verwendet, in das SRAM ab Adresse 0x10000 kopiert und über den PCI-Core eingebunden.

Anbindung von Registern und Speicher

Wie sich herausgestellt hat, werden VGA-Karten aus historischen Gründen nicht wie gewöhnliche PCI-Geräte angesprochen, sondern belegen *feste* Adressbereiche, die nicht über die BARs eingebunden werden:

- Die I/O-Bereiche 0x03b0 - 0x03bb und 0x03c0 - 0x03df
  für die Steuerung der verschiedenen Controller, z.B. Umschaltung der Bildschirmmodi
- Die Speicherbereiche 0xb8000-0xbffff und 0xa0000 – 0xaffff
  für die Speicherung von Text- und Grafikbildschirm

Die Karten dekodieren also direkt die entsprechenden Adressleitungen des Busses. Zum Lesen reicht es, ähnlich der POST-Komponente, parallel zum PCI-Core auf den Bus zugreifen zu können. Tests mit dem PCI-Tracer haben jedoch gezeigt, daß die Werte an die I/O-Register oder der Inhalt der Bildschirme nicht einfach wie bei den POST-Nummern auf den Bus gegeben werden, sondern die VGA-Karte diese wie ein PCI-Gerät entgegennimmt (*DEVSEL*-Signal), nur eben an festen Adressen.

Zumindest beim Lesen von Registern müssen die Daten auf den Bus gegeben werden, ohne andere Geräte zu stören. Aber auch beim Schreiben beispielsweise von Text in den Textspeicher brach der verwendete Rechner den Vorgang ab (PCI *master abort*). Schreib/Lesebefehle auf I/O-Register und Speicherbereiche müssen also korrekt von dem Gerät mittels eines PCI-Zyklusses angenommen werden.

Möglicherweise geht dies auch *mit* dem verwendeten Core. Aus Zeitgründen konnte nicht mehr versucht werden, die Adressbereiche mit Hilfe der *backend-reconfiguration* Möglichkeit des Cores fest in die BARS einzutragen.

Implementierung von Registern und RAM

Für eine einfache Emulation wird es sicher nicht nötig sein, alle I/O-Register zu implementieren. Die Farbverwaltung auf Palettenebene oder die Wahl des Zeichensatzes im Textmodus könnte beispielsweise vereinfacht werden. Es muß nur getestet werden, inwieweit die BIOSse der Rechner zum Zeitpunkt des Systemstartes die I/O-Register der Karte auf Funktionalität prüfen und bei eventuellen Mißständen Fehler melden. Möglicherweise wird es ausreichen, einige beschreibbare Register zur Verfügung zu stellen, ohne jedoch deren Funktion nachzubilden. Evtl. werden solch grundlegende Funktionen auch direkt in das VGA-BIOS ausgelagert und der Grafikkarte selbst überlassen. Auch hier gilt es, eine Minimalimplementierung zu finden, so daß die Karte von allen Rechnern, in denen sie betrieben werden soll als VGA-Karte erkannt wird.

Die Speicherung der Textseiten kann einfach auf dem SRAM der FPGA-Karte erfolgen und ähnlich wie BAR1 an eine bestimmte Adresse angebunden werden.

Status, Weitere Entwicklung

Bisher wurde die Erkennung als VGA-Karte durch das Setzen der Kennung 0x030000 im PCI Class-Code Register und die Anbindung des VGA-ROMs durch das SRAM erreicht. In der Arbeit wurde zu

diesem Zweck das VGA-ROM von GNU verwendet. Es wurde in der Datei *vgabios_lgpl.bin* gespeichert und kann durch das Programm *write_sram* vom µCSimm-Modul aus in das SRAM ab Adresse 0x10000 geschrieben werden. Von dort wird es über den PCI-Core zugänglich gemacht. Martin Kirsch hat in seiner Arbeit das BIOS leicht modifiziert, um eine korrekte Erkennung zu gewährleisten.

Der bestückte Rechner sollte dann ohne Fehlermeldung starten. Die Zugriffe auf die I/O-Adressen und den Testspeicher werden aber mit einem *master-abort* auf dem PCI-Bus beendet, da der PCI-Core sie nicht behandelt. Die nächste Aufgabe wird also sein, die *backend*-Programmierung des Cores zu versuchen oder eine eigene kleine Logik parallel zum Core zu programmieren, die diese Zyklen korrekt behandelt. Möglicherweise reicht es, das Schreiben von Textzeichen in den Textbildschirm zu Implementieren, um den Inhalt später aus dem SRAM lesen zu können. Damit wäre eine Minimalversion hergestellt. Der weitere Ausbau kann in kleinen Schritten erfolgen.

Weitere Einzelheiten zur VGA-Problematik finden sich in der Arbeit von Martin Kirsch.

VGA-Grafikbildschirm

An dieser Stelle soll noch kurz auf die Frage eingegangen werden, inwieweit auch die Implementierung einer VGA-Grafikdarstellung sinnvoll ist. Dies ist für ein BIOS-Setup zunächst nicht nötig, jedoch könnte dann die Fähigkeit, einen Rechner zu warten, auch auf den Betrieb von möglichen eingesetzten grafischen Betriebsystemen erweitert werden.

Zunächst einmal faßt die Standard-VGA-Auflösung 640 mal 480 Bildpunkte mit 16 Farben, d.h. zu je 4 Bit, man benötigt demnach selbst für diese kleine Auflösung 150 KByte Speicher für den gesamten Grafikbildschirm im Vergleich zu knapp 4 Kbyte für den Textbildschirm mit 80 Spalten, 25 Zeilen und 2 Byte pro Zeichen. Diese 150 Kbyte müssen natürlich zunächst von der CPU auf der Karte ausgelesen und dann über das Netzwerk versendet werden. Für eine einigermaßen häufig aktualisierte Darstellung werden dann etwa 500 KByte bis 1 Mbyte pro Sekunde an Daten anfallen.

Man wird daher bemüht sein, diese Menge einzuschränken, beispielsweise indem man die Daten komprimiert und/oder nur die Unterschiede zwischen den einzelnen Seiten überträgt. Dies würde aber sehr viel komplizierte Logik auf dem FPGA nötig machen, da die verwendete CPU mit dieser Aufgabe hoffnungslos überfordert wäre. Man darf natürlich auch nicht zu stark komprimieren, da sonst der Inhalt des Bildschirmes nicht mehr erkennbar wäre. Kommerzielle Programme wie *Timbuktu* oder *PC-Anywhere* zeigen jedoch, daß dies prinzipiell möglich ist.

Der eigentliche Nutzen bleibt jedoch fraglich. Die Karte wurde im Projekt unter dem Gesichtspunkt konzipiert, einen entfernten Rechner über ein Netzwerk auf Hardwareebene steuern und wichtige Funktionen überwachen zu können. Ziel war die BIOS-Konfiguration des Rechners und die Installation und Auswahl des Betriebssystemes vornehmen zu können. Diese Aufgaben können alle mehr oder weniger im Textmodus ausgeführt werden. Zu weiteren Administrationsarbeiten kann auf die Möglichkeiten des installierten Betriebssystemes zurückgegriffen werden. Da es sich bei dem Wirtsrechner um einen nicht lokalen Rechner handelt wird das eingesetzte Betriebssystem solche Funktionen besitzen. Die Übertragung von Grafikbildschirmen erfolgt dann in der Regel mittels einer Beschreibung der Grafikoperationen, z.B. mit dem unter UNIX verbreiteten System *X*.

Wird das Auslesen des Grafikbildschirms doch zwingend benötigt, so wäre eine Implementierung mit einfacher Komprimierung der Grafikdaten wie z.B. Lauflängenkodierung denkbar. Bei langsamen Netzen kann eben nur mit geringer Geschwindigkeit gearbeitet werden. Dadurch wäre aber notfalls auch ein Rechner mit Betriebssystem ohne Netzwerk-Funktionalität im Grafikmodus bedienbar.

Kapitel 5

Die Adapterplatine

Dieses Kapitel beschreibt den Aufbau der Adapterplatine. Sie stellt die Verbindung zwischen der FPGA-Karte und dem CPU Modul her und nimmt die zur Emulation der Peripheriegeräte notwendige Elektronik auf. Die konkrete Ansteuerung der einzelnen Komponenten wird in den entsprechenden Abschnitten über das FPGA-Design und die Steuerungssoftware erklärt.

5.1 Übersicht

Eine Übersicht über die Anschlüsse von FPGA und CPU findet sich in Anhang B, ein Schaltplan der Adapterplatine in Anhand C.

Die Platine gliedert sich in verschiedene Bereiche: Auf der einen Seite befindet sich die das µCSimm betreffende Elektronik, hierbei handelt es sich um die Steuer- und Datenleitungen zum FPGA, sowie um die vom FPGA unabhängigen Signale für die Kontrolle des Betriebszustandes des Rechners. Auf dem anderen Teil befindet sich Elektronik zum Anschluß der zu emulierenden Eingabegeräte. Die Platine wird mit entsprechenden Kabeln über eigene Stecker an µCSimm, FPGA-Karte und den Rechner angeschlossen.

Ein Großteil der Elektronik besteht aus Widerständen zum Schutz der Bausteine im Signalweg zwischen den Komponenten. Durch diese Maßnahme wird eine Zerstörung der Bausteine auch bei fehlerhafter, logisch entgegengesetzter, Programmierung vermieden. Zusätzlich mußte ein Teil der Signale über Widerstände schwach auf Versorgungsspannung gehalten werden, während bei der emulierten Floppy zusätzliche Transistoren nötig waren, um eine ausreichende Treiberleistung zu erzielen. Die Steuer- und Datensignale des µCSimms wurden zur Fehlerfindung mit kleinen Leuchtdioden versehen.

Um den Aufbau übersichtlich zu halten, wurden bestimmte Signale mit Farben belegt:

- *Blaue* Kabel liegen auf Masse (logisch 0), die Massen von Rechner, µCSimm und FPGA sind dabei elektrisch verbunden.
- *Orange* Kabel liegen auf Versorgungsspannung, wobei das µCSimm mit 3,3V Spannung arbeitet, während an den beiden Steckern des FPGAs *beide* vom PCI Bus gelieferten Spannungen 3,3V und 5V anliegen.

- *Lila* Kabel kennzeichnen 10 kΩ Widerstände gegen 5 V, sogenannte *Pull-Ups*. Sie werden bei der Emulation der Peripheriegeräte verwendet, wenn diese an den Rechner angeschlossen werden.

Die restlichen Farben haben keine bestimmte Bedeutung.

Einige Hinweise zur Beschaltung

Jedes der Signale zum und vom FPGA wurde zunächst über einen 1kΩ Widerstand angeschlossen, um eine Beschädigungen des Bausteines möglichst zu vermeiden. Im Falle eines Kurzschlusses könnten sonst die I/O-Zellen des FPGAs durch den hohen Strom zerstört werden. Der FPGA kann an seinen Ein/Ausgängen bis maximal 6 mA belastet werden. Durch die Widerstände wird die Belastung auf 3,3 mA respektive 5 mA begrenzt. Natürlich begrenzen Widerstände im Signalweg auch die Schaltgeschwindigkeit, bei der betriebenen Frequenz und den vorhandenen Leitungskapazitäten ist dieser Effekt jedoch noch nicht von Bedeutung.

Während der Testphase der Verbindung zwischen µCSimm und FPGA wurde immer wieder festgestellt, daß sich die im FPGA implementierte State-Maschine von Zeit zu Zeit ohne erkennbaren Grund zurücksetzte, was die laufende Kommunikation unterbrach. Eine Analyse mit dem Logic-Analyzer und dem Oszillographen ergab, daß beim Schalten des *Clk*-Signales auch kurze Pulse auf die Nachbarleitung *Req* induziert wurden, dies bezeichnet man als *Übersprechen* (engl *Cross-Talk*). Abhilfe schaffte der digitaler Filter im FPGA Design, der in Abschnitt 4.10 besprochen wurde. Alternativ wäre auch eine elektrische Lösung mit einem Tiefpass-Filterglied möglich gewesen, aber dies hätte zusätzliche Bauteile erfordert. Bei weiteren Störungen könnte man auch zu Flachbandkabeln übergehen, welche zusätzlich eine Masseleitung zwischen den Signalen haben, um die Störungen abzuschirmen, ähnlich dem Floppykabel.

Bei Änderungen der Platine oder beim Bau weiterer Komponenten ist darauf zu achten, daß Signale mit Steuerfunktion entweder elektrisch oder digital gefiltert werden. Bei der vorgesehenen Integration des Projektes auf einer einzigen Platine wird die Notwendigkeit von Filtern aber entfallen, da wesentlich kürzere Leiterbahnen zur Übertragung auftreten werden.

Weiterhin ist darauf zu achten, daß niemals direkt Versorgungsspannungen unterschiedlichen Ursprunges zusammengeschaltet werden, auch wenn sie augenscheinlich dasselbe Spannungspotential aufweisen. Durch die Spannungsschwankungen einzelner Geräte und die Übergangswiderstände der Verbindungen würde trotzdem unnötigerweise ein minimaler Strom fließen, zusätzlich wäre die Gefahr eines unbeabsichtigten Kurzschlusses größer. Es ist im Elektronikbereich dagegen üblich, bei Verbindungen von Geräten deren Massen zu verbinden um ein gemeinsames Nullpotential als Bezug zwischen allen Geräten zu haben. Die Signale beider Geräte werden dann auf diese gemeinsame Masse bezogen.

Im folgenden werden die einzelnen Bereiche kurz besprochen und eventuelle Besonderheiten der Beschaltung erwähnt.

5.2 Datenbus FPGA - µCSimm

Das µCSimm-Modul verwendet eine Spannungsversorgung und einen Signalpegel von 3,3V. Der Verbindungsbus zwischen FPGA und CPU besteht aus 8 Datenleitungen und 4 Steuerleitungen. Die Signale werden zum Schutz des FPGAs über 1kΩ Widerstände zum FPGA geleitet. Jedes Signal ist mit einer Leuchtdiode versehen, um die Übertragung verfolgen zu können. Die Datenleitungen wurden am CPU Port C angeschlossen, für die Steuersignale wird der CPU Port D verwendet.

Da die Verbindung zwischen CPU und FPGA nur über einen Widerstand erfolgt, ist bei der Beschaltung der Signale auf eine Vermeidung von Kurzschlüssen zu achten. Standardmäßig sollten sowohl die I/O-Zellen des FPGAs als auch die Ports der CPU auf Eingangsbetrieb geschaltet sein. Im FPGA-Design und bei der Erstellung der Software für die µCSimm-CPU wurde auf diesen Umstand geachtet. Möglicherweise könnte die Verbindung ebenfalls über die Methode mit Pull-Up/offenem Kollektor wie bei den Eingabegeräten zum Rechner realisiert werden. Bei der späteren Integration auf einer gemeinsamen Platine wird man dies jedoch aus Platz- und Kostengründen nicht wollen.

5.3 Steuerung des Betriebszustandes des Rechners

Eine Anforderung des Projektes war, den zu steuernden Rechner notfalls zurücksetzen oder ganz aus- und wieder einschalten zu können. Zu diesem Zweck wurden zwei verschiedene Möglichkeiten ausprobiert: vor und nach dem Netzteil.

Steuerung über das ATX Netzteil

Heutige Rechner verfügen meist über ATX-Netzteile, bei denen der Rechner durch das Ausschalten nicht vollständig vom Netz getrennt wird. Bestimmte Bereiche der Hauptplatine werden weiterhin mit Strom versorgt, unter anderem die Anschlüsse der beiden Taster zum Ein/Ausschalten und zum Zurücksetzen des Rechners.

Falls der Rechner ausgeschaltet ist, verfügt der FPGA klarerweise nicht über Strom, da sich die Karte im PCI Bus des Rechners befindet. Aus diesem Grund werden drei weitere Signale direkt vom CPU Port D zur Steuerung des Netzteiles verwendet. Es handelt sich bei diesen Signalen um die Emulation der Reset- und Stromtaster sowie um ein Signal zur Überprüfung seiner Stromversorgung.

Die Emulation der vom Benutzer gedrückten Taster erfolgt durch zwei Optokoppler, welche von der CPU angesteuert werden und auf der Gegenseite wie elektrische Schalter wirken. Über die angebrachten Pfostenstecker können die Optokoppler mittels eines gewöhnlichen Steckerkabels an die dafür vorgesehenen Pfostenstecker auf der Hauptplatine des Rechners angeschlossen werden. Auf diese Weise läßt sich der Rechner von der CPU aus sowohl ein- als auch ausschalten und zurücksetzen. Da die Optokoppler auf der Schaltseite eine Diode verwenden, ist beim Aufstecken des Steckers auf die richtige Polung zu achten. Der Vorteil in der Verwendung von Optokopplern liegt in der Tatsache, daß diese auf der Schaltseite einen völlig elektrisch von der Steuerseite getrennten Stromkreis schalten. Sie können also ebenso wie ein Relais verwendet werden, sind jedoch um einiges kleiner und benötigen keine Mechanik.

Unterbrechung der Stromversorgung

Unter bestimmten Umständen möchte man jedoch auch die Möglichkeit haben, auch das Netzteil vom Strom zu trennen und direkt die Hauptstromversorgung des Rechners zu unterbrechen. Der zu diesem Zweck geschaffene und im folgenden beschriebene Aufbau wurde nur aus experimenteller Neugierde geschaffen und wird in dieser Form später sicher nicht eingesetzt werden.

Um schon vor dem Netzteil in die Stromversorgung einzugreifen, mußten keine Veränderungen am Rechner selbst vorgenommen werden. Es wurde eine gewöhnliche Mehrfachsteckdose modifiziert und ein Halbleiterrelais in den Stromweg eingebaut. Die Stromzufuhr des Rechners kann nun über das Relais gesteuert werden. Das Relais wiederum benötigt selbst auch eine Niedervolt-Stromversorgung für den Steuerstrom und kann damit bis zu 10 A bei 230 V Netzspannung schalten.

Das Halbleiterrelais wird ebenfalls durch den erwähnten Optokoppler auf der Adapterplatine geschaltet und wird dadurch über denselben Anschluß am Port C der CPU angesteuert. Die Ansteuerung selbst erfolgt allerdings verschieden, da bei Emulation des ATX-Tasters Impulse gegeben werden, während das Relais Niveau-abhängig arbeitet.

Wie bereits erwähnt, war die Verwendung des Relais nur ein Test und ist aufgrund der notwendigen Größe des Relais nicht für den späteren Einsatz zum Schalten der Stromversorgung des Rechners vorgesehen. Nebenbei muß bemerkt werden, daß das Relais nur *eine* Leitung der Steckdosenleiste unterbricht. Aus sicherheitstechnischen Gründen muß in der Regel aber nicht nur die 230V Phase sondern auch der Nulleiter getrennt werden.

Zu beachten ist, daß in jedem Fall das Vorhandensein einer eigenen Notstromversorgung im Form eines Akkus oder eines eigenen Netzteiles für die CPU notwendig ist, sonst kann die Karte nach dem Ausschalten des Stromes nicht mehr angesprochen werden. Die standardmäßig vorhandenen Stromversorgungskabel innerhalb eines Rechners führen im ausgeschalteten Zustand in der Regel keinen Strom und können für diese Zwecke nicht verwendet werden. Dieser Teil des Projektes ist noch zu erarbeiten. Im Wesentlichen müßte das momentan verwendete Netzteil der CPU durch einen Akku mit entsprechender Ladeelektronik ersetzt werden.

5.4 Anschluß der Peripherie

Die Eingabegeräte werden mittels der bereits erwähnten *Pull-Up* Widerstände mit dem Rechner verbunden. Diese haben den Zweck, momentan nicht verwendete Signale schwach auf Versorgungsspannung zu halten. Sie werden elektrisch gesehen am offenen Kollektor eines Transistors betrieben und können so von beiden Seiten gelesen oder mittels des Transistors auf Masse gezogen werden, beide Seiten können also das Signal auf logisch 0 oder 1 setzen. Ein Kurzschluß durch entgegengesetzte Beschaltung ist dagegen nicht möglich.

Floppy

Zum Anschluß der Floppy kann das üblicherweise im Rechner vorkommende Kabel verwendet werden, ein entsprechender Stecker wurde auf der Adapterplatine angebracht. Von den Anschlüssen wurden 10 Signale verwendet. Fünf werden als Eingang betrieben und weitere fünf als Ausgang, der Spannungsbereich beträgt 0-5 V. Zwischen jeder Leitung auf dem Floppykabel liegt zusätzlich eine Masseleitung, um Störungen der Kabel untereinander abzuschirmen.

Um die fünf als Eingang betriebenen Leitungen zu lesen, war es lediglich notwendig, einen entsprechenden Widerstand zum Schutz des FPGAs in den Signalweg zu schalten. Die Pull-Up Widerstände der Floppy auf der Hauptplatine des Rechners erwiesen sich mit ca. 150 Ω als recht niedrig dimensioniert, so daß der FPGA den Strom von knapp über 30 mA aufnehmen müßte, um diese auf Masse zu schalten. Da die Eingänge des FPGAs dies nicht können wurde ein zusätzlicher Transistor verwendet, der dies übernimmt und vom FPGA angesteuert wird. Aus diesem Grund müssen die Signale invertiert im FPGA-Design angesteuert werden. Im Design wird der aktive Zustand nun mit logisch 1 geschaltet.

Um die Transistoren als Schalter zu betreiben, wurden die Basiswiderstände so dimensioniert, daß die Transistoren im Schaltfall einen recht hohen Basisstrom erhalten, damit der Transistor voll durchschaltet. Aus diesem Grund wird die Ansteuerung leicht gesättigt, und es kommt bei der Wegnahme des Signals zu einer kleinen Verzögerung durch die notwendige Umladung der Sperrschichtkapazität. Bei der Floppy-Emulation spielt dieser Effekt jedoch nur beim Datensignal eine Rolle, da es nur hier auf das richtige Timing ankommt, alle anderen Signale haben eher statische Natur. Um den Effekt in diesem Fall zu eliminieren, wurde ein entsprechender Kondensator über dem Basiswiderstand angebracht (nachzulesen unter [Hin96], Seite 46 ff).

PS2 Geräte

Die Signale der PS2 Geräte werden über Pfostenstecker zur Verfügung gestellt und mittels zweier gebastelter Kabel mit den entsprechenden Eingänge des Rechners verbunden. Jeder PS2 Anschluß verfügt über drei Leitungen, eine von ihnen liegt auf Masse.

Bei der Messung der Größe der verwendeten Widerstände zeigten sich einige Unterschiede zwischen verschiedenen betrachteten Rechnern. Ein Rechner verwendete knapp 10 kΩ große Widerstände auf der Hauptplatine, ein anderer hatte diese nicht und nahm sie offenbar als im Gerät vorhanden an. Bei der Größe des Pull-Ups von 10 kΩ genügt ein einziger 1 kΩ Widerstand zum lesen und schreiben im Signalweg, da die auftretenden Ströme nicht zu groß für den FPGA sind. Da die Größe des Pull-Up Widerstandes unterschiedlich sein kann und keine Norm in diesem Zusammenhang zu finden war, wird es jedoch in Zukunft ratsam sein, die Ansteuerung der Signale analog zur Ansteuerung bei der Floppy mittels eines Transistors zu betreiben.

An dieser Stelle soll nochmals darauf hingewiesen werden, daß beide PS2-Geräte sowohl bezogen auf die Elektrik als auch auf das FPGA-Design gleichwertig sind und auf zwei benachbarte Bereiche im Konfigurationsbereich des FPGAs abgebildet werden. Ein Anschluß wird von der Software auf dem µCSimm Modul zur Maus-Emulation verwendet, der andere zur Emulation der Tastatur.

Kapitel 6

Die Software zu Test und Demonstration

In den vorhergehenden Kapiteln wurde die in der Diplomarbeit entstandene Hardware vorgestellt. Um die Hardware zu testen und deren Möglichkeiten zu demonstrieren, wurde auch Software erstellt, welche in diesem Kapitel beschrieben wird. Der Großteil läuft wie auch später vorgesehen auf der kleinen Steuer-CPU des Modules µCSimm. Zusätzlich wurden Programme für das Betriebssystem Linux geschrieben, mit deren Hilfe über den PCI Bus des Testrechners auf Funktionen der Karte zugegriffen werden kann.

Zunächst handelt es sich dabei um eine Funktionsbibliothek um grundlegende Funktionen zur Ansteuerung der Hardware zu implementieren. Weiterhin entstanden Programme, die jeweils die Arbeitsweise mit einzelnen Teilen der Hardware demonstrieren. Schließlich wird noch eine Möglichkeit gezeigt, die Bedienung der Karte über das Netzwerk durch einen möglichen Administrator zu ermöglichen. Am Ende des Kapitels folgt eine Beschreibung der Linux/PCI Software. Die entwickelte Software befindet sich ebenfalls auf der CD zur Arbeit, vergleiche Anhang D.

Da die Steuersoftware sich jeweils auf einzelne im FPGA erstellte Komponenten bezieht, sollten immer auch die entsprechenden Abschnitte in Kapitel 4 herangezogen werden.

Rahmenbedingungen und Grundlage der Software

Durch den Einsatz des CPU-Modules µCSimm konnte die Software auf die Funktionalität des Netzwerkbetriebssystemes µCLinux zurückgreifen. Dadurch konnte sich die Programmierung auf die Implementierung der wesentlichen das Projekt betreffenden Funktionen beschränken. Für Dateioperationen etc. wird auf Betriebssystemfunktionen zurückgegriffen.

6.2 Konzept der Software

Um die im FPGA implementierten Funktionen anzusprechen, wurde folgendermaßen vorgegangen: Unter dem vorliegenden Betriebssystem war es möglich, von Programmen aus direkt auf spezielle

Register und Adressbereiche der CPU zuzugreifen. Dadurch konnten die in der CPU vorhandenen seriellen und parallelen Ports oder auch der Timer direkt programmiert werden. Bei dem µCSimm Modul sind nun mehrere Pins des Prozessors direkt mit Pins des Simm-Steckers verbunden, welcher seinerseits über die Adapterplatine an den FPGA angeschlossen ist. Um mit dem FPGA zu kommunizieren, mußten also im Wesentlichen die CPU-eigenen Ports programmiert werden. Damit ergab sich für die Software folgendes Konzept:

Demonstration der Funktionen in separaten Programmen

Im Projekt wurden zunächst nur Beispielprogramme zur Demonstration der einzelnen im FPGA implementierten Funktionen geschrieben. Um die Programme kurz und übersichtlich zu halten entstand zum Test jedem Funktionellen Teil des Projektes ein eigenes Programm. Bei der Programmierung wurde bereits auf eine konsequente Fehlerbehandlung Wert gelegt, um bereits eine gewisse Stabilität zu erreichen, wie sie in einem späteren Einsatz nötig sein wird. Beispielsweise kann bei ausgeschaltetem Rechner der FPGA nicht angesprochen werden, da er nicht mit Strom versorgt ist. Die CPU verfügt jedoch über eine eigene Stromversorgung und arbeitet weiter. Der betreffende Programmteil gibt in diesem Fall eine Fehlermeldung zurück. Das Hauptprogramm kann weiterlaufen und diese Meldung korrekt behandeln. Aus diesem Grund nehmen die Testprogramme recht ausführliche Textausgaben vor. In der späteren Verwendung, wenn alle Programmteile ausreichend getestet wurden, wird man anstelle der Ausgaben entsprechende Fehler zurück an den Steuerrechner eines Administrators weiterleiten.

Programmierung einer Bibliothek für den Hardwarezugriff

Die von allen Programmen benötigten Funktionen und Konstanten zum Zugriff auf die CPU-Pins und weiter auf Konfigurationsregister im FPGA befinden sich in einer eigenen Funktionsbibliothek, die von den Programmen eingebunden wird. In dieser Bibliothek finden sich auch Funktionen zur Ansteuerung des SRAMs auf der Karte wieder. Die Bibliothek bildet damit eine Treiberschicht zwischen der Hardware und den Programmen, welche die Funktionalität nutzen.

Erweiterbarkeit

Die Software kann als Basis für eine weitere Entwicklung herangezogen werden. Zu einem späteren Zeitpunkt soll die gesamte Funktionalität der Karte über das Netzwerk mit Hilfe einer einzigen Benutzerschnittstelle einem Administrator zur Verfügung gestellt werden. Die Bündelung der einzelnen Funktionen der Karte sowie die Kommunikation über das Netzwerk wird dann nur noch von einem einzelnen Programm erledigt. Momentan wurde bisher zu Demonstrationszwecken ein kleiner Internet-Daemon geschrieben, um das Prinzip zu veranschaulichen. Diesen Teil der Software weiter auszubauen, war nicht mehr Schwerpunkt dieser Arbeit. Martin Kirsch hat sich in seiner Diplomarbeit näher mit der Kommunikation zwischen der Karte und einem entfernten Administrator beschäftigt und eine Client/Server Software namens *Janus* geschrieben.

6.3 Übersicht

Im Folgenden werden kurz die Arbeitsweisen der einzelnen Programme beschrieben. Dabei wird nur auf die grundsätzlichen Gedanken eingegangen und auf eine ausführliche Dokumentation verzichtet. Zu diesem Zweck können die Quelldateien herangezogen werden.

Die Software gliedert sich in die folgenden Teile:

- Die Bibliothek *fpga_lib*, bestehend aus den Teilen *fpga_lib.h* und *fpga_lib.c*. In ihr werden CPU und FPGA spezifische Konstanten definiert, sowie Initialisierungsfunktionen und Basisfunktionen zum Datenaustausch zwischen CPU und FPGA bereitgestellt. Alle anderen Programme benutzen die Bibliothek um über sie auf den FPGA zuzugreifen.
- Basisprogramme, um einzelne Bereiche der Registerbank im FPGA oder des SRAMs auf der Karte auszulesen oder gezielt beschreiben zu können. In diese Kategorie fallen:

  Programme um auf die Konfigurationsregister des Designs zuzugreifen:

dump_regs	Ausgabe der Konfigurationsregisterbank im FPGA
set_reg	Setzen/Lesen eines Registers

  Programme um das SRAM auf der Karte anzusteuern:

dump_sram	Ausgabe des SRAM Inhaltes
set_sram	setzen/lesen einzelner Speicherzellen
write_sram	SRAM mit konstantem Wert füllen oder eine Datei in das SRAM übertragen
ramtest	ausführlicher Test des SRAMs und der Ansteuerung

- Die Funktionen zur Emulation eines PS/2 Gerätes wurden ebenfalls in einer eigenen Bibliothek namens *ps2_lib* zusammengefaßt. Sie besteht aus den Dateien *ps2_lib.h* und *ps2_lib.c* und wird von den Programmen zur Emulation von PS/2 Geräten verwendet, momentan Maus und Tastatur.
- Beispielprogramme um die eigentlichen Steuerfunktionen der Karte zu testen. Diese Programmteile können später für eine weitere Entwicklung herangezogen werden und zeigen demonstrativ die Funktionalität der einzelnen im FPGA entwickelten Komponenten.

  Die Betriebszustand-Funktionen kommen ohne FPGA aus:

host_reset	Resets des Wirtsrechners über Schaltung des Reset-Pins
host_power	Abschalten und Anschalten des Wirtsrechners über den Strom-Pin, desweiteren Anzeige vom Status der Stromversorgung

  Im FPGA implementierte Module zur Emulation von Peripherie:

p80test	Auslesen der POST-Informationen des BIOSes an I/O-Port 0x80 des Rechners
floptest	Emulation einer Floppy
mousetest	Emulation einer PS/2 Maus, verwendet PS/2 Bibliothek
keybtest	Emulation einer PS/2 Tastatur, verwendet PS/2 Bibliothek

- Der Internet-Daemon

  ciaservd

  Er wird von dem Systemdienst *Inetd* gestartet und stellt eine Beispielentwicklung für ein späteres server-artiges Programm dar. Dadurch wird auf einfache Weise die Bedienung der Karte von einem zentralen Rechner mit einem dort laufenden weiteren Programm ermöglicht.

- Zusätzliche Dateien, die von den Programmen benötigt werden:

Makefile	Steuert die Kompilierung aller Programme mit *make*
award_451pg.txt	Einige POST Codes mit Bedeutung für das Verbreitete Award-BIOS 4.51pg
keytab.h	Übersetzung von ASCII-Zeichen in Scancodes einer deutschen Tastatur
win98boot.img	Abbild einer Windows Bootdiskette mit Sektor-Steuerinformationen
vgabios_lgpl.bin	Abbild des verwendeten VGA-BIOS

Die Programme werden von der Kommandozeile des Betriebssystemes gestartet und nehmen gegebenenfalls Parameter durch Kommandozeilenoptionen entgegen. Alle Programme geben mit der Kommandozeilenoption *-h* Auskunft über die Syntax ihrer Verwendung. Im Programm selbst wird zunächst eine Bibliotheksfunktion aufgerufen, um die verwendeten CPU-Ports zu initialisieren und die Signale zum FPGA mit Defaultwerten zu belegen. Programme die auf FPGA-Funktionen zurückgreifen geben die Version des FPGA-Designs aus. Falls der FPGA nicht unter Strom steht, weil der Wirtsrechner ausgeschaltet ist, brechen die Programme hier ab. Diese Teile müssen bei einer späteren Migration der Software nur einmal zu Anfang ausgeführt werden.

Im folgenden werden die Funktionen der erstellten Programme knapp im einzelnen besprochen.

6.4 Zugriff auf den FPGA - Die Bibliothek *fpga_lib*

Grundsätzliche die CPU-Ports und den FPGA betreffende Konstanten und Funktionen wurden in einer eigenen Bibliothek mit dem Namen *fpga_lib* gekapselt. Die gesamte Steuerung des Hardwaredesigns im FPGA geschieht letztlich über diese Funktionen indem Register im Konfigurationsregisterbereich beschrieben und gelesen werden. Weiterhin werden Funktionen zum Zugriff auf das SRAM der Karte bereitgestellt. Die Bibliothek wird von allen anderen Programmen zu Beginn mittels einer *#include* Anweisung eingebunden und stellt letztendlich die Verbindung zum FPGA her.

In der Datei *fpga_lib.h* befinden sich folgende Deklarationen:

- CPU Portadressen und die Bedeutung deren Bits für die Host- und FPGA-Steuerung zusammen mit einigen kleinen Makros zur einfachen Verwendung, sowie die Konstanten für die Kommandos des Steuerbusses zum FPGA
- Zuweisung von Konstanten und Bedeutung der Bits im FPGA Konfigurationsregisterbereich.
- Funktionsdeklarationen für die Funktionen der Bibliothek.

Die Datei *fpga_lib.c* enthält Funktionen zum Ansteuern des FPGAs. Neben Basisfunktionen wie *Reset()* oder *Version_out()* vor allem Funktionen zum Lesen und Schreiben der Register in der Registerbank im FPGA und um Daten mit dem SRAM auf der Karte auszutauschen. Die Funktionen verwenden dabei den in Abschnitt 4.6 beschriebenen Bus zwischen CPU und FPGA. Schließlich wurden noch Funktionen programmiert um Bereiche im SRAM zu füllen oder eine Datei vom µCLinux System in das SRAM auf der Karte zu übertragen.

Bei der Verwendung des SRAMs ist darauf zu achten, daß die Ansteuerung 32 Bit breit erfolgt, beim Beschreiben von nicht an 4 Byte Grenzen ausgerichteten Adressen werden also die restlichen Bytes mit verändert. Der Grund ist, daß an den SRAM Chips auf der FPGA-Karte keine *byte-enable* Signale angeschlossen sind um einzelne Bytes zu beschreiben und so Schreibbefehle immer alle 32 Bit betreffen. Umgehen ließe sich dies nur mit einem *read-bevore-write* Mechanismus, entweder im FPGA als Modul oder eben vom verwendeten Programm selbst aus. Dieser Mechanismus wurde bisher nicht implementiert, da daß SRAM nur für größere lineare Blöcke von Daten vorgesehen ist, welche einfach an 32 Bit Grenzen ausgerichtet werden.

Zu beachten ist weiterhin die sogenannte *byteorder*, d.h. die Wertigkeit einzelner Bytes innerhalb einer Zahl, die sie darstellen. Im SRAM wird bei Zahlen das niederwertigste Byte zuerst abgespeichert. Dies kommt daher, daß bei PCI-Zugriffen in das SRAM die Datenleitungen direkt an das SRAM geleitet werden und der PCI Bus mit eben dieser Byteorder arbeitet. Die Reihenfolge bei der Steuer-CPU auf dem µCSimm ist jedoch umgekehrt, hier wird das höherwertige Byte zuerst abgespeichert. Wichtig ist, daß beim Schreiben von Datenmengen von der CPU in das SRAM die Reihenfolge der Bytes nicht verändert wird. Lediglich die *Interpretation* dieser Daten als Zahlen liefert auf der CPU andere Ergebnisse als im Design des FPGAs. Dies betrifft z.B. den gewählten Port im Port-80 Modul im FPGA.

Die Geschwindigkeit der Übertragung zwischen CPU und FPGA beträgt momentan etwa 100 Kbyte/Sekunde und ist für die vorgesehenen Zwecke ausreichend. Es bestehen jedoch noch Möglichkeiten, diese Geschwindigkeit zu erhöhen:

Momentan wird bei jedem Takt der Übertragung zunächst ein Datenwort gelesen und auf den externen 8-bit Port gelegt. Weiterhin werden von der CPU durch direkte Registerprogrammierung zwei Flanken am Clock-Pin erzeugt. Eine Implementierung dieser Hauptschleife in Assembler könnte etwas an Geschwindigkeit mit sich bringen. Bei Betrachtung des durch den Compiler erzeugten Assembler-Codes zeigt sich jedoch, daß dieser schon recht gut optimiert ist, wenn er mit entsprechenden Vorschriften an den Compiler erzeugt wurde. Um die Schleife zu beschleunigen könnte man aber im inneren mehrere Befehle direkt hintereinander schreiben und so die Anzahl der Sprünge in der Schleife sparen. Dieses Verfahren wird gemeinhin als *Loop-Unrolling* bezeichnet. Letztlich wäre es auch möglich, Daten an beiden Flanken von *Clk* zu übertragen. Diese Methode wird z.B. in der Ansteuerung des kommerziellen DDR-RAMs verwendet. Eine detailliertere Beschreibung des CPU-FPGA Interfaces siehe Abschnitt 4.6.

6.5 Hilfsprogramme für Basisfunktionen

Die Basisprogramme wurden geschrieben, um die einzelnen Funktionen der Bibliothek auch von der Kommandozeile zugänglich zu machen. Mit ihnen können einzelne Bereiche der Register im FPGA oder des SRAMs angesprochen werden. Da sowohl der Registerbereich als auch das SRAM über ein BAR in den Adressbereich des PCI Busses auf dem Wirtsrechner abgebildet werden, sind diese Funktionen auch vom Wirtsrechner über die Linux-Programme für den PCI Bus zugänglich. Im einzelnen wurden folgende Programme geschrieben.

Dump_regs	Übersichtliche Ausgabe aller Register des Konfigurationsbereiches im FPGA.
Set_reg	Lesen und Beschreiben einzelner Register zum Testen einzelner Module im FPGA.
Dump_sram	Ausgabe angegebener Bereiche im SRAM.
Set_sram	Ausgeben und verändern von SRAM Speicherinhalten.

Write_sram

Zum Füllen des SRAMs mit konstanten Werten und zum Schreiben von Dateien in das SRAM. Das Programm benutzt direkt die Funktionen der FPGA-Bibliothek. Mit ihm kann z.B. ein Diskettenimage oder das VGA-ROM in das SRAM geladen werden.

Speziell die Funktion zum Füllen des SRAMs ist gut geeignet um die Geschwindigkeit des Busses zwischen CPU und FPGA zu messen. Bei Tests ergaben sich dabei Geschwindigkeiten von ca. 100 Kbyte/Sek.

Ramtest

Dieses Programm bietet die Möglichkeit, dem Hardwareteil zum Ansteuern des SRAMs einem ausführlichen Test zu unterziehen. Das Programm wurde ursprünglich auf dem Testrechner unter Linux geschrieben, um über den PCI Bus auf das SRAM zuzugreifen. Später wurde es dann auf das µCLinux System portiert, um auch einen Test über das CPU-FPGA Interface vornehmen zu können. Es ist damit ein schönes Beispiel für eine Portierung eines Linux Programmes nach µCLinux.

Das Programm schreibt Daten blockweise in das SRAM und liest sie zurück. Die zurückgelesenen Daten werden dann mit den ursprünglichen Daten verglichen. Speicherfehler werden vom Programm ausgegeben. Um spezielle Arten von Fehlern zu erkennen, bietet das Programm eine Reihe von Optionen zu testender Bitmuster wie inkrementelle Werte, laufende Bits, speziell gesetzte oder gelöschte Bits und letztendlich Zufallszahlen. Mit dieser Hilfe ist es möglich, bestimmte Regelmäßigkeiten bei Speicherfehlern zu erkennen und deren Ursache zu finden, wie z.B. Gesetzmäßigkeiten in der Adresse oder der Art der Bitmuster wie auch Fehler in Adressen bei der Beschreibung von Nachbarzellen usw. Die Tests können dabei automatisch beliebig oft wiederholt werden. Ein spezieller Debug-Modus erzeugt eine ausführliche Ausgabe über jeden einzelnen Schreibbefehl.

Momentan werden von dem Hardwaredesign nur 128 Kbyte SRAM angesteuert. Diese können mit dem Kommando: *ramtest 0 20000* überprüft werden. Das Programm testet standardmäßig mit Zufallszahlen.

Das Programm sollte in Zusammenarbeit mit einem Logic-Analyzer verwendet werden, der die Signale auf der Karte zur Ansteuerung des SRAMs beobachtet. Die RAM-Signale gehören mit zu einem 180 Bit breiten Bus auf der Karte und können über Stecker auf eine aufsteckbare I/O-Karte geleitet und von dort bequem abgegriffen werden. Mit dieser Kombination wurden während der Arbeit einige Fehler in der Ansteuerung gefunden bis letztlich das aktuelle Modul entwickelt wurde. Ein ausführlicher Test mit dem Ramtest-Programm ist unbedingt nötig, falls sich Änderungen an dem FPGA-Modul zur Ansteuerung des SRAMs als nötig erweisen sollten, z.B. bei der Entwicklung des nötigen SRAM-Controllers. Das momentan implementierte Modul wurde intensiv getestet und funktioniert zuverlässig.

6.6 Stromversorgung und Reset

Die Karte simuliert über die Adapterplatine die Gehäusetaster für Stromversorgung und Reset des Rechners (vgl. Abschnitt 5.3). Im Projekt wurden drei Bits der externen CPU-Ports für diesen Zweck verwendet, die jeweils äußeren Anschlüssen des µCSimm-Modules entsprechen. Mit einem Bit läßt sich ein Reset ausführen, und mit einem weiteren die Stromversorgung schalten. Über das dritte Bit kann der Status der Stromversorgung abgefragt werden. Dies ist nötig, da die Taste für die Stromversorgung bei den meisten Rechnern lediglich Umschaltfunktion hat. Es wurde darauf geachtet, daß die von der Steuer-CPU verwendeten Pins sich nach einem Neustart derselben auf einem Logiklevel befinden, bei dem die Funktionen nicht unabsichtlich ausgelöst werden.

Um die Funktion zu demonstrieren, wurden zwei Beispielprogramme geschrieben. Da der FPGA in diesem Zusammenhang keine Rolle spielt, befinden sich in den Programmen auch keine Funktionen zum Zurücksetzen oder Initialisieren des FPGAs.

Host_reset

Simuliert einen eine Sekunde lang gehaltenen Reset-Taster und löst so einen Reset des Rechners aus.

Host_power

Liefert den Zustand der Stromversorgung der Karte und damit des Rechners. Weiterhin läßt sich der Rechner mit diesem Programm auch ein bzw. ausschalten, indem eine Betätigung des Stromtasters simuliert wird. Bei den aktuellen Betriebssystemen wird jedoch teilweise eine Betätigung dieses Tasters abgefangen, beispielsweise um den Rechner in einen Schlafzustand zu versetzen. Der Rechner würde in diesem Fall also nicht wirklich ausgeschaltet werden, dies ist aber hin und wieder wünschenswert. Die in der Regel verwendeten modernen ATX Netzteile fragen den Taster jedoch ebenfalls ab und erzwingen ein Ausschalten des Rechners, falls er wenigstens 4 Sekunden lang gehalten wird. Nach der Emulation des Tastendruckes wird daher bis zu 6 Sekunden abgewartet, ob sich der Rechner wirklich ausschaltet, d.h. ob die Stromversorgung der Karte noch vorhanden ist.

6.7 Port 80 Statusmeldungen

Im Abschnitt 4.9 wurde die POST-Komponente im FPGA beschrieben, welche die Statusnummern beim Systemstart des BIOSses mitliest und die jeweils letzte geschriebene Nummer in einem Register speichert.

Zur Demonstration der POST-Komponente im FPGA wurde das Programm *p80test* geschrieben, welches in einer einfachen Schleife dieses Register ausliest. Falls eine Änderung des Wertes erkannt wird, wird auf dem Bildschirm dessen Bedeutung ausgegeben. Das Programm stellt zu Beginn im Register 0x0c der Registerbank die 7-Segment Anzeige auf der Karte auf die Anzeige der mitgelesenen Werte um, wodurch man sich einen Eindruck vom Verlauf des Startvorganges machen kann.

Zu beachten ist, daß prinzipbedingt nur der jeweils letzte Wert von Interesse ist, ein Speicher für bisher gelesene Werte besteht daher nicht. Aus diesem Grund ist auch die Abfrage der Werte nicht zeitkritisch und es wurde auf die Verwendung von Interrupts in diesem Zusammenhang verzichtet. Falls der Rechner während des Startvorganges auf Probleme stößt, hält er in der Regel die

Ausführung an und es werden keine weiteren Werte auf die betreffende I/O-Adresse geschrieben. Der letzte geschriebene Wert kann dann aus dem Register ausgelesen werden.

Um die Funktionsweise zu demonstrieren, kann die Karte in einem Rechner betrieben werden, in dem künstlich ein Hardwarefehler herbeigeführt wurde, z.B. in dem der Speicher entfernt wurde. Ist das Programm gestartet und wird der Rechner eingeschaltet, kann der Startvorgang anhand der Ausgabe der entsprechenden Werte verfolgt werden. An einem bestimmten Punkt (dem Speichertest) wird der Rechner stehenbleiben, und die entsprechende Fehlernummer ist der letzte vom Programm angezeigte Wert.

Da einige BIOS-Hersteller in diesem Zusammenhang eine alternative I/O-Adresse verwenden kann über eine Kommandozeilenoption auch die zu beobachtende Adresse vorgegeben werden. Einige Compaq-Rechner benutzen beispielsweise 0x84, ältere IBM-PS/2 Rechner 0x90, IBM Microchannel Rechner 0x680, alte EISA-Rechner 0x300. Der überwiegende Teil der heutigen Rechner dürfte jedoch die Adresse 0x80 verwenden.

Da es keinen Hersteller-übergreifenden Standard für die Bedeutung der Statusnummern gibt, hängt die Interpretation jeweils von dem verwendeten BIOS ab, häufig finden sich selbst Unterschiede in verschiedenen Versionen desselben Herstellers. Aus diesem Grund liest das Programm zunächst eine Liste aus einer Textdatei ein, in der einige Codes zusammen mit deren Bedeutung abgelegt sind. Auf der Kommandozeile kann gegebenenfalls ein anderer Dateiname übergeben werden. Als Beispiel wurde eine Liste für das weit verbreitetes BIOS *4.51pg* der Firma *Award* beigelegt, die standardmäßig eingelesen wird. Eine Auflistung der Bedeutung der POST-Nummern findet sich in der Regel auf den Internetseiten der BIOS-Hersteller oder auf einschlägigen Seiten wie [BioC].

Zu beachten ist weiterhin, daß Schreibbefehle auf den Port 0x80 zur Problembehebung zunächst nur vom BIOS des Rechners verwendet werden. Sobald ein Betriebssystem auf dem Rechner läuft, kann über die Bedeutung von an diese Adresse geschriebenen Werten erst einmal keine Aussage gemacht werden. Im Test wurde unter anderem ein Linux-System beobachtet, welches während des Betriebes fortwährend Werte auf die I/O-Adresse 0x80 geschrieben hat. Das erstellte Programm filtert einige dieser Werte heraus um ständigen Bildlauf zu vermeiden. Vom Konzept her ist die Verwendung der im Projekt erstellten Karte aber nur zum Zweck der Steuerung und Administration eines Rechner gedacht, bis zu dem Zeitpunkt, in dem ein Betriebssystem geladen ist.

6.8 Emulation von Peripheriegeräten

Die wesentliche Aufgabe der Karte ist die Simulation zusätzlicher Peripheriegeräte des Rechners, um die Bedienbarkeit über das Netz zu ermöglichen. Die folgenden beschriebenen Programme steuern die zu diesem Zweck im FPGA entwickelten Komponenten an und zeigen deren Funktion. Die einzelnen funktionalen Teile können später in einem übergeordneten Programm verwendet werden.

Behandlung von Anfragen an Peripheriegeräte

Die meisten Peripheriegeräte erhalten von Zeit zu Zeit Meldungen und Daten vom Rechner auf die sie entsprechend reagieren müssen. Dies gilt selbst für reine Eingabegeräte wie Maus oder Tastatur, welche beispielsweise nach einem Neustart des Rechners von diesem konfiguriert werden. Die Programme auf der CPU müssen in der Lage sein, eine solche Anforderung zu erkennen um wie das emulierte Gerät reagieren zu können. Die Komponenten im FPGA wurden so erstellt, daß sie in diesem Fall ein Signal an einem speziellen Pin an der CPU auslösen.

Um auf solche Ereignisse rechtzeitig reagieren zu können, wäre es zunächst nötig, von der Programmebene aus den logischen Zustand des Pins mit einer bestimmten Frequenz abzufragen, was aber letztendlich eine Verschwendung von Rechenzeit darstellen würde. Die auf dem µCSimm verwendete CPU bietet jedoch die Möglichkeit, bestimmte Pins speziell zu konfigurieren. Eine Änderung des Logiklevels an einem so konfigurierten Pin löst dann eine zu dem jeweiligen Pin zugeordnete Unterbrechung des aktuellen Programmablaufes aus. Diese Unterbrechung wird als *Interrupt* bezeichnet. Im Falle eines Interruptes wird zunächst eine spezielle Funktion des Betriebssystemes aufgerufen, welche ihrerseits einen vorher festgelegten und speziell zu diesem Zweck geschriebenen Softwareteil des Projektes aufruft, ein sogenannter *Interrupt-Handler*. Ein solcher Handler stellt in der Regel zunächst die genaue Ursache des Interruptes fest, um entsprechend darauf zu reagieren. Nach der Abarbeitung wird das ursprünglich laufende Programm fortgesetzt. Mit diesem Mechanismus ist es möglich, ohne regelmäßige Abfragen in relativ kurzer Zeit auf asynchron auftretende Ereignisse zu reagieren.

Da ein Interrupt zunächst vom Betriebssystem bearbeitet wird, muß ein solcher Handler zunächst korrekt in das Betriebssystem eingebunden werden. Einen kompletten Interrupt-Handler zu schreiben lag jedoch nicht mehr im Rahmen dieser Arbeit. Zu diesem Punkt wird auf die Arbeit von Martin Kirsch verwiesen, der einen Handler für das µCLinux Betriebssystem implementiert hat.

Die im Projekt erstellten Programme haben primär den Zweck, die Arbeitsweise der im FPGA implementierten Module zur Emulation von jeweils einem Peripheriegerät zu demonstrieren. Falls die Geräte auch auf Anfragen des Rechners antworten müssen und die zugehörigen Module in diesem Fall einen Interrupt auslösen, wurde in das entsprechende Programm ein Modus eingebaut, in dem der logische Zustand des Pins innerhalb einer Schleife abgefragt wird. Die für die Peripheriegeräte spezifischen Programmteile, welche auf Interrupt-Signale reagieren, wurden der Übersichtlichkeit wegen innerhalb der Programme in eigenen Funktionen untergebracht und können so oder ähnlich später von einem Interrupt Handler aufgerufen werden. Im Quelltext der Programme befinden sich entsprechende Kommentare, an denen der Ablauf verfolgt werden kann.

Bis zum Zeitpunkt der Beendigung der Arbeit wurden im FPGA drei Komponenten für Peripheriegeräte geschrieben, die bei Bedarf einen Interrupt auslösen: Maus, Tastatur und Floppy. Im Falle eines Interrupts kann im Interrupt-Konfigurationsregister im FPGA nachgelesen werden, welche der drei Komponenten den Interrupt ausgelöst hat und der für dieses Gerät zuständige Bearbeitungsteil der Software aufgerufen werden.

Emulation einer Floppy

Zunächst ist zu bemerken, daß auf die simulierte Floppy vom Rechner nur lesend zugegriffen werden kann, die Möglichkeit zu schreiben wurde bisher nicht implementiert. Wird ein solcher Lesezugriff von der Komponente im FPGA erkannt, löst diese zunächst einen Interrupt auf der Steuer-CPU aus. Ein Programm auf der CPU kann die aktuelle Spur der Floppy in einem entsprechenden Register im Konfigurationsbereich nachlesen. Nachdem die Daten dieser Spur vom Programm in das SRAM geschrieben wurden, setzt das Programm das ACK-Bit im Floppyregister und die Komponente im FPGA beginnt diese Daten bitweise zum Wirtsrechner zu senden, der diese als die betreffende Spur der simulierten Diskette interpretiert. Eine genauere Beschreibung wurde im Abschnitt 4.8 gegeben.

Das Programm floptest

Das Programm *floptest* demonstriert diese Funktion. Zunächst wird die auf der Karte befindliche 7-Segment Anzeige auf das Anzeigen der virtuellen Spur der Floppy umgeschaltet und ein Diskettenwechsel simuliert. Weiterhin wird die gesamte zu emulierende Diskette aus einer Datei in den Speicher eingelesen. Der Dateiname kann über eine Kommandozeilenoption angegeben werden. Standardmäßig wird eine Startdiskette mit dem Windows-Betriebssystem emuliert.

Wie oben bereits erläutert wurde kein eigener Interrupt-Handler geschrieben, sondern das Programm fragt einfach in einer Schleife den Zustand des Interrut-Pins ab und wartet auf ein Signal. Ändert sich der Zustand des Pins, so wird zunächst überprüft, ob die Ursache ein Floppy-Zugriff war und in die entsprechende Funktion zu dessen Bearbeitung verzweigt. Diese liest zunächst die angeforderte Spur der Diskette aus dem entsprechenden Register im Konfigurationsbereich aus.

Im weiteren Verlauf muß festgestellt werden, ob der Rechner letztlich von dieser Spur lesen will oder ob die angeforderte Spur nur zwischenzeitlich angefahren wurde, um den Lesekopf zu einer weiter entfernten Spur zu bewegen, die das eigentliche Ziel darstellt. Zu diesem Zweck wird in regelmäßigen Abständen das Spur-Register im Konfigurationsbereich ausgelesen und gewartet, bis dessen Wert eine gewisse Mindestzeit stabil bleibt. In diesem Zusammenhang haben sich 10 msek als brauchbar herausgestellt.

Im Folgenden kann mit der Übertragung der Daten beider Seiten der Diskettenspur über den FPGA in das SRAM begonnen werden. Die jeweils etwa 12 Kbyte Daten werden dabei in 4 Kbyte großen Stücken übertragen. Zwischenzeitlich muß immer wieder überprüft werden, ob sich die angeforderte Spur nicht doch noch geändert hat, in diesem Fall muß die Übertragung abgebrochen werden. Wurden beide Seiten der Spur fehlerfrei übertragen, so wird das entsprechende ACK-Bit im Konfigurationsbereich der Floppy-Komponente gesetzt um die Behandlung des „Interrupts“ zu beenden.

Das Programm gibt während der Bearbeitung die angeforderten Spuren aus und informiert über gegebenenfalls nötige Abbrüche bei der Übertragung von Daten zur Floppy. Gleichzeitig kann die aktuell angefahrene Spur an der Anzeige auf der Karte abgelesen werden. Mit dieser Hilfe kann sich ein guter Eindruck über die Funktionsweise verschafft werden.

Das gesamte Zusammenspiel wurde mit verschiedenen Zugriffsarten auf die virtuelle Floppy getestet: Zunächst mittels dem Linux-Programm *dd*, mit dessen Hilfe einzelne Sektoren einer Diskette gelesen werden können. Auf diese Weise konnte die korrekte Positionierung des emulierten Lesekopfes überprüft werden. Weiterhin wurde mit den Linux Programmen *mdir* und *mcopy* auf die Diskette zugegriffen, eine Standardmethode um unter Linux auf Windows-Disketten zuzugreifen. Die gesamte Diskette konnte ohne Probleme in ein Linux-Verzeichnis kopiert werden. Schließlich wurde der Rechner neu gestartet, und das BIOS begann nach dem Startvorgang erfolgreich von der emulierten Diskette die Windows-Dateien zu laden und zu starten. Die Komponente im FPGA zusammen mit dem Programm hat sich bei den durchgeführten Tests mit 2 unterschiedlichen Rechnern als stabil erwiesen.

Die von dem Programm benötigten Abbilder von Disketten wurden mit einem eigens zu diesem Zweck geschriebenen Programm erstellt. Dabei wurde die Diskette zunächst Sektor für Sektor ausgelesen und in eine Datei geschrieben. Die ist unter Linux mit dem bereits erwähnten Programm *dd* möglich. Unter Windows gibt es diesbezüglich einige Sharewareprogramme, z.B. *Winimage*. Als zweiten Schritt muß das Abbild noch um einige Zusatzinformationen erweitert werden, die auf einer Diskette zwischen den eigentlichen Datensektoren gespeichert werden. Zu diesem Zweck hat Martin Kirsch im Rahmen seiner Arbeit ein Programm geschrieben [Kir] (siehe auch beiliegende CD zur

Arbeit). Diese letztlich erzeugte Datei wird vom Programm verwendet. Ein Diskettenabbild einer ursprünglichen 1,44 MB Diskette belegt dadurch etwas über 2 MB Speicher.

Probleme bei Spurwechseln

Bei Spurwechseln muß besonders auf das Timing geachtet werden. Wenn der Controller eine weiter entfernte Spur anfährt, löst er solange einzelne Stepperbewegungen aus, bis sich der Kopf über der anzusteuernden Spur befindet. Die CPU auf der Karte kann am Track-Register zwar erkennen, daß sich die Spur geändert hat, kann aber zunächst nicht wissen, ob der Kopf schon an der Zielspur angekommen ist oder sogleich weiterbewegt wird. Die Spuren, die von dem Kopf nur überfahren werden, werden vom Controller aber nicht gelesen. Er wartet lediglich einige msek bis der Steppermotor gefahren ist und führt dann den nächsten Schritt aus. Das Programm auf der CPU wartet daher auch erst eine gewisse Zeit ab, ob sich die Spur weiter ändert, bis sie die Daten der Spur zum SRAM sendet.

Eine Schwierigkeit bestand darin, diese Wartezeit richtig zu bestimmen. Zu kleine Wartezeiten beim Spurwechsel erkennt man daran, daß der Controller den Kopf zur nächsten Spur fährt, während das Programm bereits das SRAM beschreibt und den Schreibvorgang abbrechen muß. Zu große Wartezeiten erkennt man an einer Fehlermeldung des Controllers an den Wirtsrechner über eine fehlerhafte Diskette, da die Daten zu spät geliefert werden..

Eine Umdrehung einer Diskette entspricht ca. 200 msek. Mehrere Spurwechsel hintereinander erfolgen in der Regel mit Abständen von 3-4 msek [Haa96], wobei diese Zeit ein wenig von dem verwendeten Controller abhängt. Auf der Zielspur angekommen, erwartet der Controller zunächst das Indexsignal als Beginn der Spur und beginnt darauf hin Daten von der Datenleitung zu lesen. Wartet die CPU zu lange mit dem Senden der Daten an das SRAM, so findet der Controller den gesuchten Sektor nicht und meldet einen Fehler. In diesem Fall werden mehrere Stepperbewegungen ausgeführt, um die Spur erneut anzufahren, letztendlich wird der Vorgang abgebrochen. Erfahrungsgemäß meldet der Controller jedoch erst einen Fehler, nachdem er die gesamte Spur einmal gelesen hat, d.h. von Indexloch zu Indexloch. Damit bleiben der CPU also ungefähr 2 mal 200 msek Zeit, die Daten sollten aber deutlich früher geliefert werden. Das Senden beider Diskettenseiten der Spur von der CPU in das SRAM dauert für 2 mal 12.790 Byte etwa 260 msek. Als Wartezeit zwischen Spurwechsel und Senden der Daten ergibt sich damit ein Wert zwischen etwa 5 bis 140 msek. In der Praxis haben Wartezeiten von 10 msek zu einem stabilen Betrieb geführt. Auch mit Zeiten von 50 msek und mehr funktionierte die Emulation problemlos. Falls sich die Übertragung der Daten von der CPU in das SRAM später als zeitkritisch erweisen sollte, bestehen noch einige Verbesserungsmöglichkeiten, die in Abschnitt 4.8 diskutiert wurden.

Die Bibliothek für PS2-Geräte

Bei heutigen Rechnern werden die Standardeigabegeräte Maus und Tastatur in der Regel jeweils über einen eigenen bidirektionalen sogenannten PS2-Anschluss mit dem Rechner verbunden. Da die zu entwickelnde Karte, um den Rechner zu steuern, dessen Eingabegeräte als angeschlossene Hardware über das Anschlußkabel simulieren sollte, wurde eine eigene Komponente im FPGA entwickelt, mit der sich ein PS2-Gerät emulieren läßt.

Zu beachten ist, daß sowohl die Maus als auch die Tastatur zwar über einen eigenen Bus mit dem Rechner verbunden sind, sich aber die Steuerung lediglich auf Softwareebene in den Daten unterscheidet, welche über den jeweiligen Bus versendet werden. Auf Hardwareebene reichte daher

die Entwicklung einer einzigen Komponente aus, die ein PS2-Gerät simulieren kann. Diese Komponente wurde dann zwei mal im FPGA-Design verwendet. Im Konfigurationsbereich des FPGA gibt es also zwei Bereiche welche jeweils einer eigenen PS2-Komponente und auch einem eigenen Anschluß auf der Adapterplatine entsprechen. Die geschriebene Software verwendet einen für die Emulation der Maus und den anderen für die Emulation der Tastatur.

Den Basisfunktionen von Maus oder Tastatur zum Senden und Empfangen von Daten liegt also dieselbe Komponente zugrunde. Im Prinzip könnten auch andere Geräte, welche PS2 verwenden, über diese Komponenten emuliert werden. Aus diesem Grund wurde eine eigene Softwarebibliothek namens *ps2_lib* geschrieben, welche diese Funktionen implementiert und von weiteren Programmen eingebunden wird. Die Bibliothek besteht aus 2 Teilen:

In der Datei *ps2_lib.h*, befinden sich Definitionen über die Bedeutung der Register und deren Bits im Konfigurationsbereich des FPGAs, sowie spezielle Konstanten, die während der Ansteuerung des Busses eine Rolle spielen. Die Registeradressen verstehen sich dabei als Offsets zu den Basis-Konfigurationsregistern von Maus und Tastatur.

In der Datei *ps2_lib.c* wurde eine Funktion zum Senden von Daten über den PS2 Bus an den Rechner implementiert.

Daten senden

Die entsprechende Funktion schreibt zunächst das zu sendende Byte in das zu diesem Zweck vorgesehene Register und setzt das entsprechende REQ-Bit um den Sendevorgang auszulösen. Die Funktion wartet nun in einer Schleife auf eine positive oder negative Rückmeldung der FPGA-Komponente. Kommt nach einer bestimmten maximalen Anzahl von Durchläufen keine Antwort, so wird ein Fehler zurückgegeben. Meldet die FPGA-Komponente ihrerseits einen Fehler zurück, so wird zunächst ein erneuter Sendeversuch unternommen.

Zwischen zwei Versuchen muß zunächst eine gewisse Mindestzeit gewartet werden, bis der Bus wieder benutzt werden kann. Die in diesem Zusammenhang verwendeten Zeitkonstanten finden sich in der Datei *ps2_lib.h* und wurden experimentell bestimmt um eine stabile und zuverlässige Übertragung zu gewährleisten.

Daten empfangen

Zunächst ist zu bemerken, daß auch reine Eingabegeräte wie Maus und Tastatur in der Lage sein müssen, Daten von dem Wirtsrechner zu empfangen. Bei diesen Daten handelt es sich im Wesentlichen um Initialisierungs- und Konfigurationsbefehle. Bei Tests während der Entwicklung hat sich gezeigt, daß sowohl das verwendete BIOS als auch die Betriebssysteme die Lesefähigkeit der Geräte benötigen, um diese korrekt zu erkennen. Wiederholt kam es vor, daß das BIOS während des Startvorgangs stehenblieb, als diese Fähigkeit noch nicht implementiert war. Wichtig schien zu sein, daß der Schreibbefehl des Rechners auf das Gerät von diesem auf Busebene korrekt bestätigt wurde. Weiterhin bestätigen sowohl Maus als auch Tastatur fast alle vom Rechner empfangene Kommandos noch einmal auf Softwareebene, indem sie das Kommando 0xfa zurücksenden.

Falls der Wirtsrechner ein Byte an das Gerät gesendet hat, löst die FPGA-Komponente für die Zeit von drei PS2-Takten also etwa 270 µsek ein Signal am Interrupt Pin der CPU aus und geht von selbst wieder in einen Ruhezustand über. Die PS2-Komponente arbeitet nicht mit einer expliziten Bestätigung, daß der Interrupt auch korrekt von der Software abgearbeitet wurde wie etwa die Floppy-

Komponente. Die Software stellt auf einen Interrupt hin das verursachende Gerät fest und liest das gesendete Kommando aus dem zugehörigen Konfigurationsregister. Da sich der Lesevorgang für die Software auf das Auslesen des Konfigurationsregisters beschränkt, wurde zu diesem Zweck keine eigene Funktion in der PS2 Bibliothek implementiert.

Eine genauere Beschreibung des PS2-Prokokolls findet sich in der Literatur [Cha99].

Emulation einer Tastatur

Mit Hilfe des Programmes *keybtest* wird die Verwendung einer der beiden im FPGA implementierten Module zum Zweck der Simulation einer an den Rechner angeschlossenen Tastatur demonstriert.

Scancodes

Zunächst ist zu bemerken, daß eine Tastatur nicht die ASCII-Werte der Zeichen der betätigten Tasten an den Rechner sendet. Die Tastatur sendet vielmehr die entsprechende Nummer der Taste selbst, zusammen mit der Information, ob diese gedrückt oder losgelassen wurde. Diese versendete Tastennummer wird auch als *Scancode* bezeichnet. Ein Programm, welches bestimmte ASCII-Zeichen simulieren möchte, muß also die entsprechenden Tastenkombinationen versenden, die eben diese Zeichen erzeugen. Insbesondere muß das Halten und Loslassen von Metatasten wie *Shift* oder *Strg* im einzelnen simuliert werden.

Normalerweise werden zum Dekodieren von Scancodes Tabellen verwendet, in denen abhängig von der verwendeten Tastatur und Landessprache zunächst den von der Tastatur erhaltenen Scancodes bestimmte Zeichen zugeordnet werden. Weiterhin gibt es aus demselben Grund unterschiedliche Tabellen, welches Zeichen im Anschluß welchem ASCII-Wert entspricht. Diese Tabellen müßten vom Programm zu korrekten Emulation einer Tastatur in umgekehrter Reihenfolge durchlaufen werden, wobei sich aber das Problem stellt, daß nicht alle ASCII-Zeichen auch über die Tastatur generiert werden können.

Da das erstellte Programm lediglich das Prinzip und die Funktion demonstrieren soll, wurde jedoch nur eine einzige einfache Tabelle verwendet, welche ASCII-Werte direkt auf an den Rechner zu sendende Scancodes übersetzt und alle alphanumerischen Zeichen korrekt behandelt. Diese Tabelle wurde in der Datei *keytab.h* eingebunden. In ihr sind die zu erzeugenden Scancodes für die meisten ASCII-Zeichen über ein zugrundeliegendes deutsches Tastaturlayout eingetragen.

Eine detailliertere Beschreibung des Tastaturprotokolles findet sich in der Literatur [ChK01]

Das Programm keybtest

Das erstellte Programm nimmt eine Zeichenfolge in einer Eingabezeile entgegen. Diese Zeichen werden dann nacheinander mittels der oben erwähnten Tabelle in Tastatur-Scancodes umgewandelt und zur FPGA-Komponente geschrieben, welche diese weiter an den Rechner sendet. Falls die Zeichen auf einer realen Tastatur die Betätigung von Metatasten erforderlich machen würden, werden zusätzlich noch Scancodes gesendet, die deren Halten und Loslassen simulieren. Wie oben bereits erwähnt, ist die Abbildung von ASCII-Zeichen auf eine Folge von simulierten Tastendrücken nur rudimentär umgesetzt und wurde in einer eigene Funktion untergebracht, die bei Bedarf verbessert werden kann.

Desweiteren kann das Programm in einen Empfangsmodus versetzt werden, in dem es wiederholt den Interrupt-Pin der CPU abfragt und vom Rechner gesendete Kommandos entgegennimmt und mittels des erforderlichen Wertes 0xfa an ihn bestätigt. Dieser Teil kann später im Rahmen einer gesamten Interruptbehandlung eingebunden werden, d.h. ein globaler Interrupt-Handler ruft diesen Teil auf, falls der Interrupt von der PS2-Komponente ausging welche die Tastatur simuliert.

Die Funktion wurde verwendet, um herauszufinden, in welchem Zusammenhang Daten vom Wirtsrechner an die Tastatur gesendet werden. Hierbei zeigte sich, daß dies im Wesentlichen nur beim Initialisieren der Tastatur der Fall ist. Beim Starten des Rechners wird diese durch das BIOS konfiguriert und erhält etwa 10 Kommandos in denen Rücksetz-Befehle gesendet werden und die Abtastrate eingestellt wird. Wie bereits erwähnt, fahren einige BIOSse den Rechner nicht weiter hoch, wenn das korrekte Zurücksenden der Empfangsbestätigung 0xfa ausbleibt. Zu diesem Zweck ist das Programm in den Empfangsmodus zu setzen, da wie bereits erwähnt noch kein Interrupt-Handler vorliegt. Beim Hochfahren des eigentlichen Betriebssystemes wird die Tastatur ein zweites Mal initialisiert.

Weiterhin verfügt das Programm über eine Funktion einen ganzen Block alphanumerischer Zeichen auf einmal zu versenden. Auf diese Weise können einfach sehr viele Zeichen gesendet werden, um die Zuverlässigkeit der Übertragung zu testen. Zu diesem Zweck wurde auf dem Rechner ein Texteditor aufgerufen und die angekommenen Zeichen mit dem gesendeten Block von Zeichen verglichen.

Um der späteren Anwendung gerecht zu werden, wurde sich mit dieser Methode zu Testzwecken auf einem Rechner eingeloggt und einige Befehle abgesetzt. Es hat sich gezeigt, daß auf diese Weise zuverlässig eine Tastatur emuliert werden kann.

Emulation einer Maus

Mittels des Programmes *mousetest* kann eine über den PS2 Bus an den Rechner angeschlossene Maus emuliert werden.

Datenformat von Mauspaketen

Im Gegensatz zur Tastatur versendet die Maus keine einzelnen Zeichen sondern immer Pakete zu drei bzw. vier Byte auf einmal. In diesen Paketen wird sowohl der Status der Maustasten als auch eine eventuelle relative Bewegung in x oder y-Richtung übermittelt. Eine Standard-PS2 Maus verwendet das Paketformat zu je 3 Byte. Zusätzlich gibt es eine *Intellimouse* genannte Erweiterung der Firma Microsoft, bei der die Informationen in einem vier Byte langen Paket versendet werden. Der Standard ist jedoch abwärtskompatibel und so reicht es aus, das 3-Byte Paketformat der Standard PS2 Maus zu unterstützen um einfache Bewegungen und 3 Maustasten zu unterstützen.

Auch eine Maus kann über den PS2-Bus Kommandos entgegennehmen und muß diese in der Regel mit dem Bestätigungsbyte 0xfa beantworten.

Genauere Informationen zum Datenformat bei Computermäusen findet sich in der Literatur [ChM01].

Das Programm mousetest

Das Programm verwendet ebenfalls die PS2 Bibliothek und benutzt die dort implementierte Funktion zum Versenden von Daten. Die zu sendenden Bewegungen und betätigten Tasten werden in einer Kommandozeile ähnlich dem Tastatur-Demoprogramm entgegengenommen. In einer eigenen

Funktion innerhalb des Programmes wird aus diesen Angaben ein entsprechendes Datenpaket zusammengestellt und versendet. Außerdem besteht die Möglichkeit durch Senden mehrerer Pakete Doppelklicks mit einer Taste zu emulieren. Zu Demonstrationszwecken kann außerdem der Mauszeiger entlang eines Viereckes über den Bildschirm bewegt werden.

Genau wie eine Tastatur wird auch eine Maus sowohl vom BIOS als auch vom Betriebssystem beim Systemstart mit Initialisierungskommandos konfrontiert. Daher befindet sich auch im Mausprogramm eine Lesefunktion, die am Interrupt-Pin auf ankommende Mauskommandos wartet, diese mit 0xfa an den Rechner bestätigt und auf dem Bildschirm ausgibt.

Die Implementierung der Maus-Emulation wurde erfolgreich getestet. Über die Softwaresteuerung konnten auf einem Rechner mit Windows-Betriebssystem über die emulierte Maus der Mauszeiger bewegt und durch simulierte Klicks Aktionen ausgelöst werden. Für die korrekte Erkennung der Maus durch Windows war es allerdings notwendig, das Programm in den Empfangsmodus zu schalten, während das Betriebssystem startete. Diesen Umstand wird jedoch später ein Interrupt-Handler auf dem µCSimm beseitigen, der im Hintergrund auf empfangene Kommandos reagiert und daher immer präsent ist.

Zu beachten ist, daß auf dem Betriebssystem bereits ein entsprechender Treiber für eine Maus installiert war. Falls die Maus korrekt erkannt werden soll (inklusive der automatischen Installation eines kompatiblen Standard PS2 Maustreibers) muß unter Umständen mehr Wert auf die richtige Beantwortung von Initialisierungskommandos gelegt werden. Dies sind letztendlich jedoch Softwaredetails, die sich problemlos dem Programm hinzufügen lassen.

6.9 Daemon als minimales beispielhaftes Benutzerinterface

Wie oben bereits erwähnt, wurden die einzelnen Programme zum Zweck der Demonstration der Arbeitsweise der einzelnen Komponenten im FPGA erstellt. Um die Karte wie vorgesehen betreiben zu können, bedarf es aber eines übergeordneten Programmes.

Auf der einen Seite muß einem entfernten Administrator ein Benutzerinterface angeboten werden, um den Rechner mittels der Karte zu steuern. Auf der anderen Seite muß eine Software auf der Steuer-CPU laufen, welche diese Kommandos entgegennimmt und entsprechende Programmteile startet, welche ihrerseits die notwendigen Befehle an die Hardwarekomponenten im FPGA senden. Wie bereits erwähnt, war dieser Teil nicht mehr im Rahmen der Arbeit und wurde von Martin Kirsch in seiner Arbeit behandelt. An dieser Stelle soll nur eine Mögliche Realisierung skizziert werden.

Im Grunde bietet schon das Betriebssystem eine minimale Möglichkeit, die Funktionen der Karte aus der Ferne zu nutzen. Das auf der CPU laufende Betriebssystem µCLinux beinhaltet einen Telnet-Daemon, über den sich ein Benutzer mittels eines gewöhnlichen Telnet-Programmes auf dem System im Textmodus einwählen kann. Nach erfolgreicher Authentifizierung können von der Kommandozeile des µCLinux-Systems die einzelnen Programme gestartet werden. Dies ist die Methode, die auch in der Entwicklungsphase zum Test der Software verwendet wurde, sie hat jedoch einige offensichtliche Nachteile. Zum einen wird dem Benutzer Zutritt zum gesamten Betriebssystem gewährt, obwohl eigentlich nur eine begrenzte Liste von notwendigen ausführbaren Befehlen zugänglich sein soll. Zum anderen ist die Einwahl und die Bedienung der Programme von Hand im Textmodus recht mühsam. Eine Möglichkeit, das erste Problem zu umgehen, könnte sein, das Betriebssystem soweit zu reduzieren, daß keine unerwünschten Aktionen mehr möglich sind. Es gibt jedoch eine elegantere Lösung, die aufgrund ihrer Einfachheit beispielhaft implementiert wurde um das Prinzip zu zeigen.

Hierbei handelt es sich um einen im Internet recht verbreiteter Mechanismus, nämlich die Steuerung mittels eines *Internet-Daemons.*

Prinzip

So wie jeder Rechner im Internet durch eine eindeutige Adresse repräsentiert wird, so wird auch ein auf diesem Rechner verfügbarer Dienst wie Email oder Webseiten durch eine bestimmte sogenannte *Portnummer* repräsentiert. Ein Programm, welches einen bestimmten Dienst auf einem Rechner nutzen will, muß bei der Verbindungsaufnahme diesen Dienst durch die Angabe seiner Portnummer benennen.

Das µCLinux System bietet nun wie ein Standard-Linux System die Möglichkeit, Programme an beliebige Portnummern zu binden und damit eine Textverbindung über das Netzwerk zu diesem Programm zu ermöglichen. Verbindet sich nun ein Benutzer von außen mit einem Terminalprogramm auf den Rechner und gibt als gewünschten Dienst eine solche Portnummer an, so wird das zugehörige Programm gestartet. Das besondere dabei ist, daß die Ausgaben und Eingaben des Benutzers auf die Standardeingaben und Ausgaben des Programmes umgeleitet werden. Ein Text, den das Programm also normalerweise auf dem Bildschirm ausgeben würde, wird nun über das Netz zum Benutzer geleitet und von dessen Terminalprogramm angezeigt. Ebenso gelangen Ausgaben des Benutzers über das Netz letztlich an die Standardeingabe des Programmes. Der gesamte Mechanismus wird durch einen meist *Inetd* genannten Dienst des Linux-Betriebssystemes geleistet.

Die Vorteile dieser Methode ist klar, daß eine Verbindung mit einem Rechner möglich ist und dem Benutzer eine Reihe von Kommandos angeboten werden können, ohne ihm jedoch Zugriff auf die Betriebssystem-Kommandozeile zu geben. Ein weiterer Vorteil besteht in der einfachen Entwicklung der Programme. Solange diese nur die Standard Ein/Ausgaben verwenden können sie sowohl von der Kommandozeile gestartet und bedient werden, als auch über das Netz, wenn sie durch Inetd aufgerufen werden.

In diesem Zusammenhang sei erwähnt, daß viele der im Internet genutzten Dienste wie z.B. FTP oder Sendmail auf diesem Prinzip beruhen und ihre Kommandos letztlich als lesbare Textbefehle entgegennehmen, dies aber von komfortablen Anwendungsprogrammen versteckt wird.

Der Daemon

Um das Prinzip zu veranschaulichen, wurde das Programm *ciaservd* erstellt. Wie beschrieben kann es sowohl von der Kommadozeile als auch durch Inetd gestartet werden. Eine genaue Beschreibung, wie das Starten mittels Inetd in das System eingebunden wird, findet sich auf der CD zur Arbeit, vgl. Anhang D.

Das Programm selbst nimmt im Wesentlichen Kommandos über eine Eingabezeile von der Standardeingabe entgegen. Die Kommandos werden bearbeitet, indem die entsprechenden oben bereits beschriebenen Programme gestartet werden. Implementiert wurde Ein- und Ausschalten des Rechners, Neustarten des Rechners sowie Anzeige des FPGA-Konfigurationsregisterbereiches. Das Programm kann aber leicht ausgebaut werden, um alle bisher implementierten Fähigkeiten der Karte verfügbar zu machen. Beispielsweise könnte eine Konfigurationsdatei erstellt werden, in der alle verfügbaren Kommandos zusammen mit den daraufhin zu startenden Programmen und ihren Optionen aufgelistet werden.

Um die Programme zu starten, muß zunächst ein zweiter Programmkontext erzeugt werden, wobei speziell unter µCLinux zu diesem Zweck die Funktion *vfork()* anstelle der sonst in diesem Zusammenhang üblichen Funktion *fork()* verwendet werden muß. Hierbei handelt es sich um eine Besonderheit im Gegensatz zum Standard-Linux System, daß bis zum eigentlichen Start des auszuführenden Programmes oder eine Fehlers das aufrufende Programm angehalten bleibt. Dies sind jedoch programmtechnische Details und können besser im Quelltext studiert werden. Der Programmteil zum Starten anderer Programme wurde in einer eigenen Funktion implementiert.

Beim Ausgeben von Meldungen ist zu beachten, daß Schreibbefehle über die Standardausgabe nicht unmittelbar auf das Netzwerk gesendet sondern zunächst einmal in einem Puffer zwischengespeichert werden, bis sich eine genügende Anzahl Daten angesammelt hat. Um eine Übermittlung z.B. nach Ausgabe einer wichtigen Information zu erzwingen besteht die Möglichkeit, den bisherigen Pufferinhalt mittels *fflush()* sofort zu senden. Weiterhin müssen Zeilenumbrüche aus Kompatibilitätsgründen zusätzlich mit dem Zeichen *\r* ausgeführt werden, sonst werden sie vom Terminalprogramm nicht als solche erkannt. Am besten kann auch dies direkt im Quelltext nachgesehen werden.

Alternative Implementierung

Bezogen auf das Projekt bietet der Dämon zunächst eine bequeme Möglichkeit, die geschriebenen Programme über ein einfaches Interface einem Administrator zugänglich zu machen, ohne kompletten Zugriff auf die Kommandozeile zu geben. Auf Seiten des Administrators kann eine Benutzerschnittstelle geschrieben werden, die Anweisungen auf Anwender-freundliche Weise entgegennimmt, beispielsweise über eine grafische Bedienoberfläche. Dieses Programm kann letztlich Kommandos an die Karte ebenfalls über den Dämon abgeben. Da die Übertragung im Textmodus abläuft ist eine einfache Fehlerbehebung möglich.

Der Nachteil der Methode liegt sicherlich zum einen in der Unsicherheit der Übertragung gegenüber externen Beobachtern auf dem Netz, da alle Anweisungen und auch die Einwahlprozedur mit Paßwort im unverschlüsselten Textmodus übertragen werden. Dies ließe sich mit einer zusätzlichen Verschlüsselungsschicht umgehen. Ein weiterer Nachteil ist das größere nötige Datenaufkommen, welches zur Absetzung eines Befehles nötig ist.

Aus diesem Grund kann die Übertragung auch Paketorientiert implementiert werden, beispielsweise über das Internetprotokoll UDP. Jeder Befehl an die Karte kann dann mit einem einzelnen Paket versendet werden, welches Kennwort oder Verschlüsselung gleich mit implementiert haben kann. Diese Methode ist programmtechnisch jedoch sicherlich aufwendiger als ein textorientierter Dämon. Martin Kirsch hat im Rahmen seiner Arbeit eine Client-Server Software zu diesem Zweck namens *Janus* geschrieben.

6.10 Linux PCI Programme

Auf dem verwendeten Testrechner unter Linux wurde ebenfalls Software entwickelt, um über den PCI-Bus auf die Karte zugreifen zu können. Bei der Entwicklung wurde von einem bereits bestehenden Treiber zur PCI-Ansteuerung ausgegangen und die oben besprochenen Basisprogramme zur Ansteuerung des Designs auf das System übertragen. Der Treiber wurde ursprünglich von verschiedenen Mitarbeitern des Lehrstuhls für Technische Informatik (weiter)entwickelt, namentlich seien hier Markus Schulz und Timm Steinbeck genannt.

Die Bedienung der Programme erfolgt analog den oben beschriebenen Programmen. Sie werden hier nur kurz aufgeführt. Sie befinden sich auf der CD zur Arbeit im Verzeichnis **\linuxpci**.

- Befehle, um die Karte auf dem Bus verfügbar zu machen:

list_pci	Listet alle Geräte am PCI-Bus auf
pldareset	Konfiguriert die BARs der PLDA-Karte mit Startadressen, über die sie mit den folgenden Befehlen angesprochen werden können

- Befehle, um spezielle Zyklen auf den Bus zu geben:

set_config	Verändern von Werten im Konfigurationsbereich der PCI-Geräte
set_io	Lesen und Setzen von 32-Bit-Werten an I/O-Adressen
setw_io	Selbiges für 16 Bit Werte
setb_io	Selbiges für 8 Bit Werte
dump_io	Zum Auflisten des Inhaltes von I/O-Speicherbereichen
set_mem	Setzen und Lesen von 32-Bit werten an physikalische Speicheradressen
dump_mem	Zum Auflisten des Inhaltes von physikalischen Speicherbereichen

- Programme, um das Design zu testen:

ramtest	Beschreibung analog Abschnitt 6.5. Das Programm wurde zuerst für Linux geschrieben und später auf µCLinux portiert.

Bemerkung:

Um die Programme nutzen zu können, sind *root*-Rechte auf dem System erforderlich, sonst kommt es zu einer Fehlermeldung.

Kapitel 7

Zusammenfassung + Ausblick

7.1 Zusammenfassung der Diplomarbeit

In der vorliegenden Diplomarbeit wurde der erste Prototyp einer Fernwartungskarte für Rechner mit PCI-Bus konstruiert. Die Arbeit wurde in Kooperation mit der Diplomarbeit von Martin Kirsch ausgeführt.

Die Anforderung der Steuerung des bestückten Rechners konnte durch die Emulation der Eingabegeräte Maus, Tastatur und Floppy realisiert werden. Weiterhin läßt sich der Rechner über die Karte zurücksetzen oder auch ganz aus- und wieder einschalten. Die Bedienung der Karte über das Netzwerk wurde mit einer seriellen Verbindung vor Ort und mit einer Steuerung über ein Ethernet-Netzwerk realisiert. Der Prototyp arbeitet korrekt mit dem PCI-Bus zusammen und kann die POST-Werte des BIOSses mitlesen und anzeigen. Die Karte gibt sich auf dem PCI-Bus als VGA-Karte zu erkennen. Die vollständige Emulation der VGA-Karte konnte jedoch nicht fertiggestellt werden.

Die Implementierung erfolgte durch Verwendung einer FPGA/PCI Entwicklerkarte der Firma PLDA und des CPU-Moduls *µCSimm* der Firma RT-Control. Beide Bausteine wurden über eine Adapterplatine verbunden, auf der auch die benötigte Elektronik für die Verbindung der emulierten Geräte zum Rechner Platz fand. Die logischen Funktionen für die Emulation der Eingabegeräte und die Anbindung an den PCI-Bus des Rechners konnten vollständig innerhalb des FPGAs auf der PCI-Karte realisiert werden. Um die verschiedenen Möglichkeiten der Karte zu demonstrieren, wurden erste Programme für das CPU-Modul geschrieben, die sich separat starten lassen. Zur Repräsentation über das Netzwerk wurde ein Internet-Daemon für das Modul erstellt. Mit diesem konnte gezeigt werden, wie sich eine mögliche einheitliche Darstellung und das zur Verfügung stellen der Funktionalität der Karte über das Netzwerk realisieren läßt. Auf diese Weise ist die Funktionalität der Karte sehr leicht zu bedienen.

Die Entwicklung des FPGA-Designs erfolgte in der hardwareunabhängigen Beschreibungssprache VHDL. Der gesamte Designflow konnte mit dem zur FPGA-Karte mitgelieferten Programm Quartus der Firma Altera geleistet werden. Die verschiedenen Aufgabenbereiche des FPGAs wurden in einzelne Komponenten zerlegt und getrennt gelöst. Die Funktionalität der Komponenten wird in einer Konfigurationsregisterbank zusammengeschaltet. Auf diese kann von dem CPU-Modul aus zugegriffen werden. Mit dem momentanen Design ist der verwendete FPGA zu 25 Prozent belegt. Das Design kann später in die Flash-Speicher auf der Karte eingebracht werden, um es dauerhaft zu speichern.

Die Entwicklung der Software für das CPU-Modul wurde unter Linux mit einem Cross-Compiler für das auf dem Modul verwendete Betriebssystem µCLinux durchgeführt. Die Programme werden über

das Netzwerk geladen und können ebenfalls später auf dem Flash-ROM des Modules untergebracht werden. Dadurch kann der Prototyp vollständig unabhängig von weiteren Rechnern eingesetzt werden.

Die Arbeit zeigt, daß die Konstruktion einer Fernsteuerkarte für Rechner mit PCI-Bus grundsätzlich möglich ist. Durch die Integration möglichst viel der erforderlichen Logik im FPGA und die Verwendung einer eigenen CPU besteht eine flexible Lösung, welche die volle Kontrolle über den administrierten Rechner erlaubt. Auf diese Weise kann eine universell einsetzbare und gleichzeitig kostengünstige Lösung zur Administration von Rechenclustern oder entfernten Rechnern realisiert werden.

7.2 Ausblick

Durch den modularen Aufbau ist der erste erstellte Prototyp sehr leicht erweiterungsfähig. In diesem Kapitel soll die weitere Entwicklung diskutiert werden, dabei werden allgemeine Schritte angedeutet. Hinweise zur speziellen Verbesserung von Komponenten im FPGA oder einzelnen Teilen der Software wurden bereits an entsprechender Stelle in der Arbeit gegeben.

Zunächst sollten die bereits bestehenden Teile des Projektes besser zusammengefaßt werden, um eine einfachere Zusammenarbeit sicherzustellen. Innerhalb des FPGA-Designs betrifft dies in erster Linie die unzureichende Implementierung der Ansteuerung des SRAMs, die bisher lediglich durch einen Multiplexer gelöst wurde. Hier sollte ein Controller geschrieben werden, der gleichzeitige Zugriffe von verschiedenen Seiten auf das SRAM koordiniert, um die Richtigkeit der übertragenen Daten sicherzustellen.

Bezogen auf die Softwareentwicklung betrifft das die Erstellung und Einbindung eines Interrupt-Handlers, um verschiedene Eingabegeräte gleichzeitig und zuverlässig emulieren zu können. Hier wird auf die Arbeit von Martin Kirsch verwiesen [Kir]. Ein weiterer Schwerpunkt der Entwicklung wird die Verbesserung der Kommunikation mit einem entfernten Administrator sein. Der geschriebene Dämon zeigt zwar bereits eine einfache und zuverlässige Implementierung, doch ein entfernter Administrator hat bisher keine Möglichkeit, auf übersichtliche Weise mehrere Rechner auf einmal zu verwalten oder statistische Informationen zu sammeln. Zu diesem Gebiet hat Martin Kirsch ebenfalls in seiner Arbeit eine Software namens *Janus* entwickelt.

Als nächstes können der Karte weitere bisher nicht entwickelte Komponenten und Funktionen hinzugefügt werden. Die oberste Priorität wird die Weiterentwicklung der Emulation einer VGA-Karte sein, die in Abschnitt 4.11 eingehend diskutiert wurde. Weiterhin fehlen noch fast alle Kontrollmöglichkeiten für bereits intern im Rechner vorhandene oder zusätzliche Sensoren. Auch die Erstellung einer Geräteliste im Rechner über einen Scan des PCI-Busses konnte noch nicht implementiert werden. Für den Betrieb als Steuerkarte ist auch die eigene Energieversorgung durch einen Akku bisher nicht gelöst, dieser müßte anstelle der momentanen Stromversorgung des CPU-Modules angebunden und parallel zum Betrieb geladen werden. Für die Zukunft sollte auch die Notwendigkeit neuer Eingabegeräte der Rechner beachtet werden, insbesondere USB hat sich inzwischen als Standard etabliert.

Um die Administrationsmöglichkeiten vor Ort zu verbessern, könnten auch weitere Steuerzugänge zu der Karte geschaffen werden. Palmtops und Handhelds sind inzwischen weit verbreitet und so kann eine direkte Anbindung an diese die Administration vor Ort sehr erleichtern, denkbar wäre beispielsweise eine Infrarot-Schnitstelle in Form von *IrDA*.

Um die Karte zu steuern, wäre zusätzlich die Nutzung des Web-Servers auf dem CPU-Modul denkbar, dadurch würde eine Administration noch leichter über einen Browser ermöglicht werden. Bei der Verwendung der Karte über das Internet, muß in diesem Zusammenhang auch an Sicherheitsaspekte gedacht werden, beispielsweise durch den Einbau einer Verschlüsselung in das Protokoll.

Ein wichtiger Schritt in der weiteren Entwicklung wird die Integration des momentanen Prototypen auf einer einzigen Platine sein. Dies hilft auch bei der Kostenabschätzung für weitere Stadien der Entwicklung. Die Kosten können weiter durch die Integration des CPU-Moduls in den FPGA deutlich gesenkt werden. Von der Firma Altera gibt es die Produktlinie *Excalibur*, die für den FPGA synthetisierbare und vielseitig konfigurierbare Prozessoren anbietet. Für diese Prozessoren ist ebenfalls eine Version des verwendeten Betriebssystemes µCLinux zusammen mit einer Unterstützung für 10-Mbit Ethernet verfügbar. Dadurch wäre letztlich außer dem FPGA und den notwendigen Verbindungssteckern für die Emulation der Eingabegeräte nur noch der notwendige RAM-Speicher auf der Karte vorhanden.

Die Endkosten bei der Herstellung einer integrierten Lösung auf einer einzigen Karte reduzieren sich auf die erwähnten Bausteine FPGA und SRAM sowie natürlich der Herstellung der Platine. Momentan würden weiterhin die Kosten für CPU mit DRAM, Flash-ROM und dem Netzwerkchip hinzukommen. Sie liegen damit bei den momentanen Preisen (Ende 2001) unter $ 100, bei einem angesetzten Preis von $ 35 für den FPGA in großen Stückzahlen. Durch den Einsatz der Karte kann sowohl die Grafikkarte als auch die Floppy des bestückten Rechners eingespart werden, was die effektiven Kosten für den Einsatz weiter senkt. Bei der Herstellung von größeren Mengen, können durch den Übergang vom FPGA zum ASIC die Kosten weiter gesenkt werden, sie machen damit nur einen Bruchteil der Kosten eines typischerweise in Rechenclustern verwendeten Rechners aus.

Anhang A: Registerbelegung FPGA Design

Adresse (hexadezimal)	Bit	Beschreibung
0x00 – 0x02		Kennung CIA
0x03		Version im BCD Format, momentan 1.0 = 0x10
0x08		Interrupt-Register, Bits = 1 falls Interrupt bei Gerät aufgetreten
	0	Floppy
	1	PS2 Tastatur
	2	PS2 Maus
0x0c		7 Segment Anzeige Kontrolle, 0 = POST Werte, 1 = aktuelle Floppy-Spur
0x10 – 0x1f		uCSimm Komponente, zur Fehlerfindung via PCI eingebaut
0x10		Kontrollsignale für MUX-Ansteuerung SRAM
	0	uc_enable
	1	uc_read
	2	uc_write
0x12		aktueller Zustand der State-Maschine
0x13		aktuelles Kommando
0x14 – 0x15		aktuelle Adresse
0x18 – 0x1b		uc_data_in, Daten zum uCSimm
0x1c – 0x1f		uc_data_out, Daten vom uCSimm
0x020 – 0x2f		Floppy Komponente, Signale zur Steuerung / Fehlerfindung
0x20		Steuerregister für uCSimm, Bits = 1 für aktives Signal
	0	fc_req
	1	fc_ack
	2	fc_dskchng

	3	fc_reset
0x23		Signale vom Floppy-Stecker, Bits = 0 für aktives Signal (verwenden Pull-Ups)
	0	fe_drvsel
	1	fe_motor
	2	fe_dir
	3	fe_step
	4	fe_head
0x28		aktuell vom Controller angefahrene Spur
0x29		aktueller Zustand der State-Maschine
0x2a – 0x2b		aktuelle SRAM Adresse der Diskettendaten
0x2c – 0x2f		aktuelles Datenword, 32 Bit
0x30 – 0x33		PS2 Komponente zur Tastatur-Emulation
0x30		Tastatur-Komponente Kontrollregister, Bits = 1 für aktives Signal
	0	kb_req
	1	kb_ack
	2	kb_error
	3	kb_reset
	4	kb_irqreq
0x31		kb_data_in, an Rechner von Tastatur zu sendendes Zeichen (Scan-Code)
0x32		kb_data_out, vom Rechner empfangenes Kommando
0x33		aktueller Zustand der State-Maschine
0x34 – 0x37		PS2 Komponente zur Maus-Emulation
0x34		Maus-Komponente Kontrollregister, Bits = 1 für aktives Signal
	0	ms_req
	1	ms_ack
	2	ms_error
	3	ms_reset
	4	ms_irqreq

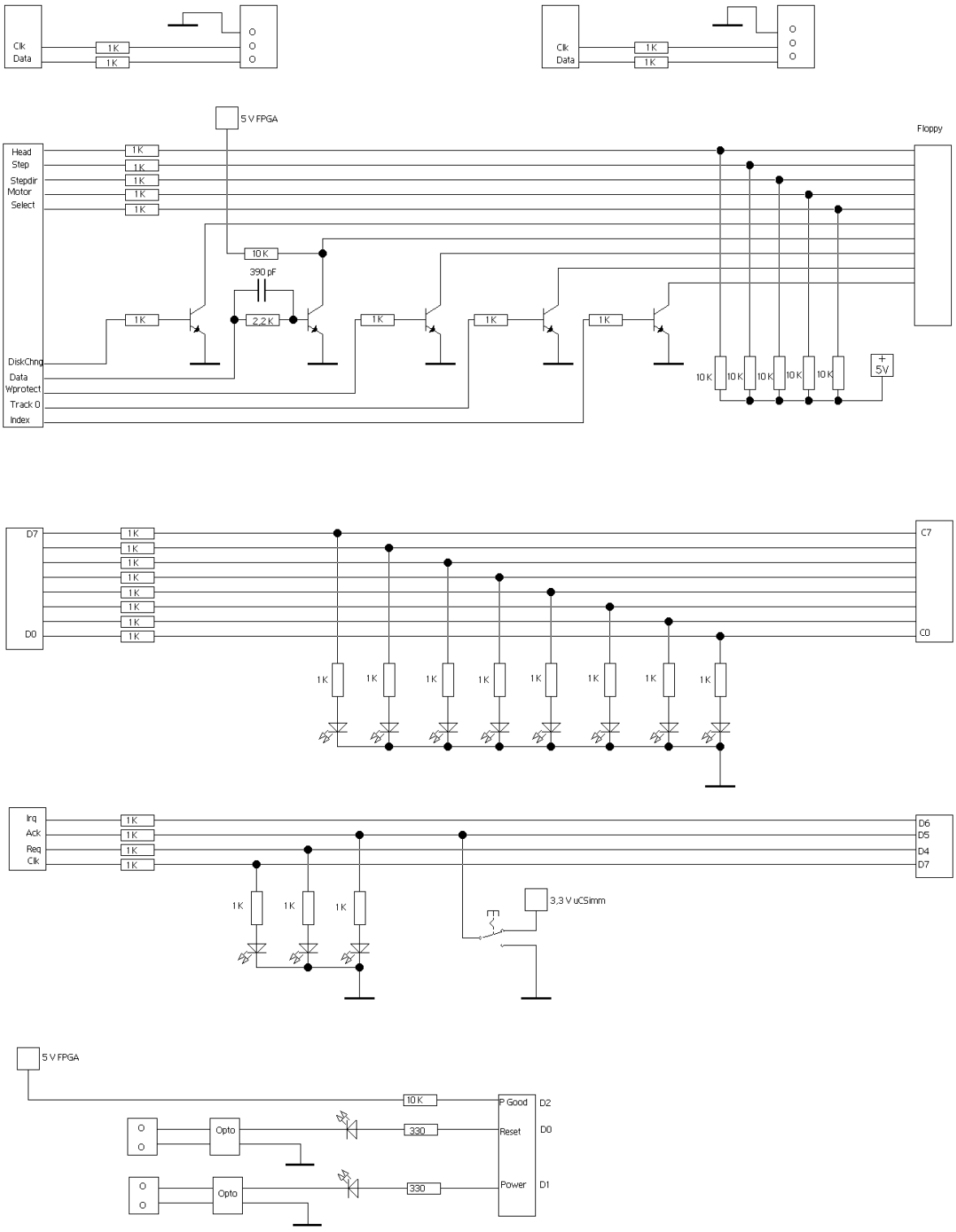
0x35	ms_data_in, an Rechner von Maus zu sendendes Zeichen
0x36	ms_data_out, vom Rechner empfangenes Kommando
0x37	aktueller Zustand der State-Maschine
0x38 – 0x3b	POST Komponente
0x38	letzter an gewählter I/O-Adresse geschriebener Wert
0x3a – 0x3b	zu überwachende I/O-Adresse, Voreinstellung 0x80

Anhang B: Externe Signale FPGA und CPU

Signal	FPGA Signal	FPGA Pin	J5 Pfostenl. Pin	Port, Bit	fpga_lib Name
Datenbus	data0	H 16	12	Port C, Bit 0	Pdata_data[0]
CPU<-> FPGA	data1	G 16	13	Port C, Bit 1	Pdata_data[1]
	data2	E 22	14	Port C, Bit 2	Pdata_data[2]
	data3	H 18	15	Port C, Bit 3	Pdata_data[3]
	data4	K 18	16	Port C, Bit 4	Pdata_data[4]
	data5	E 19	17	Port C, Bit 5	Pdata_data[5]
	data6	F 19	18	Port C, Bit 6	Pdata_data[6]
	data7	G 19	19	Port C, Bit 7	Pdata_data[7]
Steuerleitungen	us_clk	F 16	2	Port D, Bit 7	FPGA_Clk
für Bus	uc_req	H 17	3	Port D, Bit 4	FPGA_Req
CPU <-> FPGA	us_ack	G 7	4	Port D, Bit 5	FPGA_Ack
	us_irqreq	F 17	5	Port D, Bit 6	FPGA_Irq
Maus Emulation	ps2_ms_clk	F 15	36		
	ps2_ms_data	K 19	37		
Tastatur Emul.	ps2_kb_clk	J 21	38		
	ps2_kb_data	E 21	39		
Floppy Emulation	fe_index	J 19	22		
	fe_drvsel	F 21	23		
	fe_motor	G 20	24		
	fe_dir	F 22	25		
	fe_step	G 21	26		
	fe_track0	H 21	27		
	fe_wprotect	G 24	28		
	fe_dataout	J 22	29		
	fe_head	F 9	32		
	fe_dskchg	E 20	33		
Emulation der				Port D, Bit 0	Host_Reset
Gehäusetaster				Port D, Bit 1	Host_Repower
+Check Stromvers.				Port D, Bit 2	Host_Pwrgood

Siehe auch: Datenblatt FPGA Karte, uCSimm

Anhang C: Schaltplan der Adapterplatine



Anhang D: CD Inhalt

- Im Verzeichnis **\doc** befinden sich verschiedene, der im Projekt herangezogenen und in der Literatur angegebenen Webseiten und Dokumente. Nicht aufgenommen wurden die Startseiten größerer Projekte oder Firmen.
- Das Verzeichnis **\datasheets** enthält die Datenblätter zu FPGA-Karte, µCSimm und weiteren verwendeten elektronischen Bausteinen.
- Die Projektdateien und Quelltextdateien zur Hardwareentwicklung des FPGA-Designs befinden sich im Verzeichnis **\design**. Die weiter beiliegenden Konfigurationsdateien enthalten Pinbelegungen und Timinganforderungen für das verwendete Softwareprogramm *Quartus II.*
- Die Entwicklungsumgebung für die µCSimm-Software befindet sich im Verzeichnis **\ucdev**. Um sie verwenden zu können, ist eine Installation der µCLinux-Entwicklungstools erforderlich, auf die in [BeLo] eingegangen wird.
- Im Verzeichnis **\linuxpci** befinden sich die Quelltexte und Dateien der Linux-PCI-Tools.
- Im Verzeichnis **\etc** befinden sich vereinzelte weitere Dateien des Projektes, wie beispielsweise das Programm zur Erzeugung von Diskettenabbildern.

Literatur / URL Verzeichnis

[Kir] Diplomarbeit Martin Kirsch, bis zum Druckzeitpunkt nicht vollendet
Kontakt: martin.kirsch@siemens.com

[Ali] ALICE, *A Large Ion Collider Experiment,* physikalisches Experiment am Ring LHC
Web: http://alice.web.cern.ch/Alice/

[Alt01] Altera, USA: *Apex20K Programmable Logic Device Family Data Sheet*, 09/2001
http://www.altera.com/literature/ds/apex.pdf

[Ami] American Megatrends: *MegaRAC (Series 780) Remote Access Companion*, 2001,
http://www.ami.com/support/doc/megarac_spec.pdf

[Ang] Persönliches Gespräch Venelin Angelov

[BeLo] µCLinux Installation: *uClinux - Setting up the Development Environment*,
http://www.beyondlogic.org/uClinux/uClinux.htm

[BioC] BIOS-Central Webseite: Listen und Informationen zu POST-Codes
http://www.bioscentral.com/

[Bo85] Bill Croft, John Gilmore: *RFC 951, Bootstrap Protocol (Bootp)*, Internet, 1985, vgl auch RFC 1542

[Cer] CERN, heutiger offizieller Name *Organisation européenne pour la recherche nucléaire*
Web: www.cern.ch

[Cha99] Adam Chapweske: The PS/2 Mouse/Keyboard Protocol, 1999
http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm

[ChK01] Adam Chapwesk: *Interfacing the AT Keyboard*, 2001
http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/keyboard/atkeyboard.html

[ChM01] Adam Chapwesk: *The PS/2 Mouse Interface*, 2001
http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/keyboard/atkeyboard.html

[Clus98] Gregory F. Pfister: *In Search Of Clusters*, second edition, Prentice Hall PTR, 1998

[Dh93] R. Droms: *RFC 1531, Dynamic Host Configuration Protocol*, Internet, 1993, vgl auch RFC 2131

[En01] Michael Engel: *Klein aber Linux- Linux für MMU-lose Prozessoren*, Linux-Magazin 3/2001

[Grd] EU-Data-Grid, http://www.eu-datagrid.org/

[Grid99] Ian Foster, Carl Kesselman: *The Grid: Blueprint Of a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc, 1999

[Haa96] Michael Haardt, Alain Knaff, David C. Niemi: *The floppy user guide*, 1996
http://www.ipblythe.com/pdfs/floppy.pdf

[Hin96] Hermann Hinsch: *Elektronik – Ein Werkzeug für Naturwissenschaftler*, Springer, 1996

[Hof01] S. Bethke (Chair), M. Calvetti, H.F. Hoffmann, D. Jacobs, M. Kasemann, D. Linglin: *Report of the steering group of the lhc computing review*, CERN/RRB-D 2001-3

[Le94] Lehmann, Wunder, Selz: *Schaltungsdesign mit VHDL*, Franzis´, 1994

[Lhc] LHC, *Large Hadron Collider*, Beschleunigerring am CERN
Web: http://lhc.web.cern.ch/lhc/

[LinB] Linux BIOS Homepage (Linux im ROM-BIOS, Ansteuerung von Sensoren)
http://www.acl.lanl.gov/linuxbios

[LmS] Sensoren mit Linux auslesen: lm_sensors, Smbus, I2C-Bus
http://www.netroedge.com/~lm78

[Pl86] Plex86 VGA Bios, Gnu LGPL BIOS, http://cbothamy.free.fr/projects/vgabios/

[Pld00] Pld Applications, *PCI Core Users Guide*, Version 5.1.1, 2001
http://www.plda.com/dwn/userguide.pdf

[Sha98] Tom Shanley, Don Anderson: *PCI System Architecture*, Third Edition, Addison-Wesley 1998

[Si00] H. Singpiel: *Der ATLAS LVL2-Trigger mit FPGA-Prozessoren*, Dissertation, Nat/Math Gesamtfakultät Ruprecht-Karls-Universität Heidelberg, 2000

[Ti92] M. Tischer: *PC Intern 3.0 Systemprogrammierung*, Data Becker, 1992

[Tie99] U. Tietze, Ch. Schenk: *Halbleiter-Schaltungstechnik,* 11. Auflage, Springer-Verlag, 1999

[uCL] µCLinux Homepage: *µCLinux - Embedded Linux/Microcontroller Project Home Page*, http://www.uclinux.org/

[uCS] µCSimm Homepage: *µCSimm - Embedded Linux/Microcontroller Project*, http://www.uclinux.org/ucsimm/

[Usb00] Universal Serial Bus Specification, Revision 2.0, April 2000,
http://www.usb.org/developers/data/usb_20.zip

[Usb01] Marcus Stöbe, *USB 2.0 kontra fire wire,* c‘t 15/2001, Seite 128 ff, Heise Verlag 2001

Danksagung

An dieser Stelle möchte ich allen denjenigen danken, die dazu beigetragen haben, daß ich meine Diplomarbeit habe durchführen können:

Zuerst danke ich Herrn Prof. Dr. Volker Lindenstruth, daß ich die Arbeit an seinen Lehrstuhl ausführen konnte. Durch viele interessante Aufgabenstellungen auch außerhalb meiner Arbeit habe ich meinen Horizont auf vielfältige Weise erweitern können und eine sehr interessante Zeit am Lehrstuhl verbracht.

Herrn Prof. Dr. Männer danke ich für die freundliche Übernahme der Zweitkorrektur.

Weiterhin möchte ich mich der gesamten TI-Arbeitsgruppe am Institut bedanken. Ich habe mich während der Zeit meiner Arbeit durch die angenehme Atmosphäre sehr wohl in der Arbeitsgruppe gefühlt.

Ich danke auch Martin Kirsch für die gute Zusammenarbeit. Ich wünsche ihm viel Erfolg bei der Fortführung des Projektes !

Besonders danken möchte ich den Mitarbeitern und Freunden, die mir in der heiklen Endphase der Arbeit mit guten Tips und Korrekturlesen zur Seite standen, an dieser Stelle besonders meinem Bruder Martin Philipp.

Weiterhin möchte ich mich bei den Angestellten der Verwaltung des Institutes für die immer freundliche Atmosphäre Bedanken.

Der größte Dank geht an meine Eltern Christa und Gerd Philipp, die mir das Studium ermöglicht haben und mich dabei unterstützt haben.

Erklärung:

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 4.10.2001 Stefan Philipp