Fakultät für Physik und Astronomie

Ruprecht-Karls-Universität Heidelberg

Diplomarbeit

im Studiengang Physik

vorgelegt von

Martin Kirsch

aus Grünstadt

2001

Entwicklung einer PC PCI-Einbaukarte zur Clustersteuerung

Die Diplomarbeit wurde von Martin Kirsch ausgeführt am
Kirchhoff-Institut
unter der Betreuung von
Herrn Prof. Volker Lindenstruth

Entwicklung einer PC PCI-Einbaukarte zur Clustersteuerung:

Eine PCI-Einbaukarte für PC's soll entwickelt werden. Sie soll zur vollständigen Remote-Steuerung eines Rechners dienen, angefangen bei den BIOS-Einstellungen, über das Booten von einer virtuellen Diskette und das Aufsetzen eines Betriebssystems bis zur Bedienung von Programmen bei laufendem Betriebssystem. Sowohl an das Betriebssystem als auch an den PC sollen keine speziellen Anforderungen gestellt werden, ein Standard-PC soll ausreichend sein. Teil der Diplomarbeit war das Herausarbeiten von Modulen zur Emulation von Floppy, Tastatur und Maus. Ein Grafikkarten-BIOS soll ebenfalls gefunden werden. Zur Remote-Darstellung und Bedienung der Karte muss ein entsprechendes Programm entwickelt werden. Als Verbindungsprotokoll kommt TCP/IP zum Einsatz. Weitere Komponenten der Karte werden von anderen Diplomand realisiert.

Development of a PC PCI-Card for Cluster Monitoring:

A PCI-Card for PC's should be developed. It must be useful for a remote administration of a computer, beginning by configuring the BIOS, over booting from a virtual floppy-disc and installation of an operation system and ending by normal operation of programms and operating systems. No restrictions should be needed for the OS and the PC. A standard PC should be enough. Part of the diploma was developing modules for floppy, keyboard and mouse emulation. Even a VGA-BIOS should be found. For remote monitoring a programm has to be written. The basic network protocol is TCP/IP. Other components will be developed by other students.

Vorwort

An den Leser dieser Diplomarbeit wird die Anforderung gestellt, die Grundlagen von Elektronik und Programmentwicklung zu kennen. Er sollte auch Erfahrung mit binären und hexadezimalen Zahlen haben. Hexadezimale Zahlen werden in Großbuchstaben mit einem kleinen nachgestellten “h” angegeben. Binäre werden direkt als 1 oder 0 dargestellt, da sie aus dem Sinnzusammenhang eindeutig hervorgehen. Der Unterschied zwischen Byte und Bit erfolgt durch Groß-/Kleinschreibung: b-Bit, B-Byte.
Auf einen Auszug aus dem Quelltext wird in der Arbeit verzichtet, da dies zu weitreichend wäre und nicht zu mehr Verständnis der Software führen würde.

Danksagungen

Dank sei insbesondere meinem Teamleiter Thomas Weickel von der Firma Siemens gewidmet. Er hat mich immer wieder ermuntert, weiterzumachen und neue Ansätze auszuprobieren. Des weiteren stand er mir stets mit Rat und Tat zur Seite und organisierte das eine oder andere für mich. Dank auch an Benjamin Pfuhl, der mir mit dem Java Client geholfen hat, sowie an Heinrich Mainka, der mich – trotz seiner scherzhaften Art, die mich manchmal eher von der Arbeit abhielt – vor allem mental unterstützte und zum Gelingen dieser Arbeit beitrug. Simone Dederichs kämpfte mit mir gemeinsam beim Korrekturlesen mit der neuen deutschen Rechtschreibung. Ebenfalls möchte ich meiner Mutter und Oma danken, die mich versorgt haben, da ich sonst ab und zu verhungert wäre.

Inhaltsverzeichnis

Abkürzungen und Begriffserläuterungen

Eigene Abkürzungen:

CIA	**C**luster **I**nterface **A**dapter steht für die eigentliche PCI-Karte, welche die komplette Steuerung des Rechners übernimmt. Auf ihr sind das µCsimm, eine FPGA und weitere Bauteile zur Ansteuerung von Floppy, PS/2, ... zu finden. Der CIA wird das Endprodukt von dieser und weiteren Diplomarbeiten werden.
Host	Der Begriff „Host“ bezeichnet den fernzusteuernden Rechner, also denjenigen Rechner, in dem sich das CIA-Board befindet. Zu beachten ist, dass das µCsimm auf dem CIA-Board eine andere IP-Adresse hat als der Host.
Janus	**J**ava **A**dministration **N**etwork **U**niversal **S**ervice. Dieser dient der Steuerung der CIA-Karten. Janus kommt aus dem Lateinischen. Es ist die Bezeichnung eines römischen Gottes. Der Gott hat zwei Gesichter, die auf die zwei Seiten der Dinge hinweisen, zum Beispiel die gute und schlechte Seite oder auch Vergangenheit und Zukunft (folglich **Janu**ar). Der Name wurde gewählt, da der CIA-Adapter zwei voneinander unabhängige Seiten besitzt, die jedoch intern miteinander zusammenhängen. Auf der einen Seite stellt er sich als PCI-Karte, Floppy- und PS/2-Schnittstelle für den Host-Rechner dar, so dass auf dem Host normal gearbeitet werden kann, auf der anderen Seite bietet er die Möglichkeit, den Rechner fernzusteuern.
TCP-Server	Der TCP-Server ist das Hauptprogramm auf dem µCsimm. Er verwaltet bisher Maus, Tastatur und Floppy und stellt eine binäre TCP-Schnittstelle auf Port 5432 zur Verfügung, über welche die CIA’s bedient werden können.
CIA-Server	Dies ist ein von Stefan Philipp geschriebenes Programm, das zum Steuern der Reset-Funktion und dem Power-Zustand des Hosts dient. Er kann über eine normale Telnet-Sitzung auf Port 77 angesprochen werden.

Allgemeine Abkürzungen:

API	Das **A**pplication **P**rogrammer **I**nterface bezeichnet die Programmier-Schnittstelle, über die mit einem Programm kommuniziert werden kann.
BIOS	**B**asic **I**nput **O**utput **S**ystem. Es ist die niedrigste Software-Schicht eines Rechners und es ist hardwaretechnisch auf dem Mainboard implementiert. Beabsichtigt ist, einen ersten Zugriff auf die Hardware, speziell auf Laufwerke zu ermöglichen, um das Betriebssystem laden zu können. Die heutigen Betriebssysteme nutzen die Funktionen des BIOS nur noch selten, statt dessen bringen sie eigene Treiber mit, welche die Hardware effizienter ausnutzen.
Cluster	Ein Rechnercluster besteht aus mehreren Computern, die alle an der gleichen Aufgabe arbeiten. Absicht ist es, “billige” Standardcomputer in einem Netzwerk zusammenzuschalten, um keinen großen, teuren Rechner kaufen zu müssen. Außerdem ist die Störanfälligkeit (Redundanz) wesentlich geringer in direktem Vergleich zu einem einzigen Rechner.
DNF	**D**isjunktive **N**ormal **F**orm. Eine mathematische Darstellung von Bedingungen, die nur aus einer Kombination von einer in zwei Stufen zu bewältigenden Operation bearbeitet werden können. Die erste Stufe beinhaltet nur UND-Verknüpfungen, die zweite nur ODER-Verknüpfungen.
Excalibur	Ist eine FPGA von Altera mit integriertem StrongARM Prozessor. Dieser soll eine maximale Taktrate von 200MHz haben. Die FPGA soll einem Apex 20K entsprechen.
FIFO	**F**irst **I**n **F**irst **O**ut. Ein FIFO ist ein Silospeicher, welcher die Daten, die zuerst gespeichert wurden, wieder als erste ausgibt. Er wird meist als Puffer eingesetzt, um Datenströme gleichmäßiger zu gestalten.
FPGA	**F**ield **P**rogrammable **G**ate **A**rray. Eine Weiterentwicklung von GAL’s. Sie sind wesentlich komplexer aufgebaut und verfügen daher über mehr Fähigkeiten. Eine genaue Einführung zu diesem Thema gibt es in Kapitel 1.5 Hilfsmittel
GAL	**G**ate **A**rray **L**ogic. Dies sind programmierbare Logikbausteine, die eine Matrix von UND- und ODER-Gattern besitzen, welche die DNF darstellen können.

MMU	**M**emory **M**anagement **U**nit ist ein Baustein, meist im Prozessor integriert, der Speicherzugriffe verwaltet. Die Einheit ermöglicht, virtuellen Speicher zu erzeugen und einen Speicherschutz zu realisieren. Die Funktionsweise in Kürze: Die angesprochene Adresse wird mit einer internen Tabelle verglichen. Aufgrund des Ergebnisses werden entsprechende Reaktionen, wie Page Fault Auslösung oder Adressraumkonvertierung ausgeführt.
NIOS®	NIOS® ist ein Soft-Prozessor von Altera.
PCI	**P**eripheral **C**omponent **I**nterconnect, ein Standard für Erweiterungskarten für den PC.
PS/2	ist ein Standard, welcher von IBM ins Leben gerufen wurde. Er beinhaltet mehrere Teilgebiete (Hard- und Software) und bezeichnet auch den Namen eines PC-Modells von IBM. In der Diplomarbeit wird der Name hauptsächlich für die Schnittstelle von Maus und Tastatur verwendet.
PXE	**P**re e**x**ecution **E**nvironment. Ein von IBM entwickelter Standard, um das BIOS eines Rechners von der Ferne aus zu verwalten. Updates des BIOS sind dann ebenfalls möglich.
SIMM	ist eine früher gängige Bauform für Speicherbausteine (**S**ingle **I**nline **M**emory **M**odule). Nachfolger sind PS/2 und DIMM (**D**ual **I**nline **M**emory **M**odule).
SPI	**S**erial **P**eripheral **I**nterface. Ein von Motorola entwickelter Standard zur seriellen Übertragung von Daten. Das Interface besteht unter anderem aus einer Takt- und Datenleitung.
µCsimm	**µ**(mikro) **C**ontroller **S**ingle **I**nline **M**emory **M**odule ist ein Mikrocontroller in der Bauform eines SIMM's. Der Prozessor ist ein Motorola Dragonball mit 16 MHz Taktfrequenz. Des weiteren sind 8 MB Ram, 2 MB Flash RAM und eine 10-BT Netzwerkkarte auf dem Modul vorhanden.
µClinux	**µ**(mikro) **C**ontroller **linux**. Dies ist eine „abgespeckte" Variante von Linux. Auf dem µCsimm wurde die Version 2.0.38 verwendet. Ein 2.4.x Kernel existiert bereits, läuft allerdings zum Zeitpunkt der Diplomarbeit noch nicht stabil genug, so dass hier die ältere Version benutzt wurde. Beim NIOS® Prozessor wird dann wahrscheinlich eine Umstellung erfolgen. Der Unterschied zum normalen Linux besteht darin, dass dieses Linux auf Rechnern ohne MMU läuft. Es ist speziell für integrierte Systeme angepasst.
VGA	**V**ideo **G**raphics **A**dapter. Hierbei handelt es sich um einen IBM-Standard für Grafikkarten.

1 Einführung

1.1 Idee

Die Idee, eine solche Karte zu entwickeln, wurde von zwei Seiten unabhängig geboren. Das erste Mal, dass ich mit der Idee konfrontiert wurde, war bei der Firma Siemens. Dort sind viele Abteilungen über mehrere Standorte verteilt, und es ist nicht immer möglich, entsprechend geschultes Personal vor Ort zu haben. Die Abteilung, in der ich schon während des Studiums nebenbei arbeitete und auch meine Diplomarbeit absolviert habe, ist über sieben Standorte verteilt, von denen jedoch nur drei ständig administrativ besetzt sind. Da es immer wieder vorkommt, dass sich an einem nicht ständig besetzten Standort ein Rechner „aufhängt", ist es sinnvoll, eine Möglichkeit zu haben, solche Rechner auch von der Ferne aus bedienen zu können. Eine solche Karte muss mindestens folgende Basisfunktionalität besitzen: Grafikkarten-, Tastatur- und Floppyemulation. Mit diesen drei Funktionen ist es bereits möglich, einen Rechner, sofern nicht gerade ein Teil der Hardware wirklich defekt ist, komplett neu zu installieren und wieder in Betrieb zu nehmen. Besonders interessant ist hierbei das BIOS, welches leider nicht mit Standard-Software von der Ferne aus steuerbar ist. Es gibt zwar industrielle Lösungen, nur weisen diese zwei Nachteile auf: zum einen sind sie zu teuer (bis zu 8000,-€) und zum anderen an bestimmte BIOS-Versionen oder Hardwareanforderungen gebunden. Ein allgemeinerer Ansatz ist PXE, jedoch wird sich dieser Standard wohl erst in den nächsten Jahren durchsetzen. Ob PXE auch von allen Computerherstellern unterstützt werden wird, ist bis heute unklar.
Die andere Idee, diese Karte zu entwickeln, entsprang dem LHCb Projekt, welches im Jahre 2005 starten soll. Das Projekt wird vom Cern-Institut in der Schweiz geleitet. Hierbei handelt es sich um ein Kollisionsexperiment, bei dem Elementarteilchen untersucht werden. Es fallen Unmengen an Daten an, die ausgewertet werden müssen. Daher ist ein Rechnercluster mit über 1000 Knoten geplant. Die Administration eines solch großen Clusters ist aufwändig, speziell das Auffinden einzelner Knoten ist nicht einfach. Eine entsprechende Administrationsmöglichkeit ist hier sehr sinnvoll.
Die Karte lässt sich sicherlich auch unter dem Gesichtspunkt sehr gut vermarkten, da sie in jedem x86 Rechner funktioniert und somit auch normalen Heimanwendern die Möglichkeit gibt, ihren eigenen Rechner von der Ferne zu verwalten. Sinnvoll ist hier auch die Möglichkeit, endlich den Bootloader von jedem Betriebssystem aus steuern zu können.
Ein weiterer Einsatz ist auch bei Siemens geplant: da es an den oben genannten Außenstandorten neben Servern auch viele Rechner von Anwendern gibt, besteht nun die Möglichkeit, diese Rechner ebenfalls zu verwalten. Hierzu wäre eine standalone-Lösung nötig, die nur die Karte enthält und an den zu administrierenden Rechner extern angeschlossen werden kann. Das Problem mit Grafikkarten in Notebooks lässt sich auf folgende Weise lösen: Integration schneller ADC's auf der Karte zur Digitalisierung des Videosignals.
Zu erwähnen wäre auch noch, dass es mit dem Wissen dieser Diplomarbeit auch möglich ist, einen Maus-/ Tastaturumschalter zu bauen, was sicherlich als Praxisarbeit eine gute Übung ist.

1.2 Aufgabenstellung

Aufgrund des Umfangs der Diplomarbeit wurde sie in zwei Teile aufgespaltet. Ein Teil wurde im Rahmen dieser Diplomarbeit behandelt, der andere Teil von Stefan Philipp. Die Aufteilung im Einzelnen:

Stefan Philipp:

VGA-Grafikkarte emulieren
Schnittstelle zwischen µCsimm und FPGA
Erste Module in der FPGA implementieren

Martin Kirsch:

Client zur Steuerung der Karte
Floppyemulation
Tastatur-/Mausemulation
VGA-BIOS (zu Anfang nicht geplant)

Die beiden Diplomarbeiten haben zwar Schnittstellen, jedoch sind die einzelnen Teile sehr gut voneinander abgrenzbar. Dies ist auch sinnvoll, um gegenseitige Abhängigkeiten zu vermeiden. Zum kompletten Verständnis müssen einige Aspekte der Arbeit von Stefan Philipp erwähnt werden. An den entsprechenden Stellen wird hierauf hingewiesen.
Zusätzlich zu den in dieser Arbeit erwähnten Tatsachen wurden noch weitere Aufgaben erledigt, von denen hier nur eine kurz angesprochen werden soll. Hierzu zählt die Entwicklung eines Schaltplanes der CIA-Karte und das Selektieren von verschienen Bauteilen, die benutzt werden können. Vollendet wurde diese Arbeit nicht und wird im Moment von der Werkstatt des physikalischen Instituts weiterverfolgt.
Aufgrund der Komplexität dieser Karte wurde bisher nur ein Teil aller geplanten Funktionen entwickelt. Die wichtigsten Grundlagen wurden in den beiden Diplomarbeiten herausgearbeitet. Auf diesem Wissen muss in Zukunft aufgebaut werden. Es werden noch weitere Praktikumsarbeiten/Diplomarbeiten/Doktorarbeiten nötig sein, um die Karte zu vervollständigen.
Zum zeitlichen Ablauf gibt es noch folgendes zu sagen: die erste Aufgabenstellung war die Floppyemulation, die nach vielen Rückschlägen doch bewältigt werden konnte. Anschließend wurde das Protokoll der Tastatur entschlüsselt. Da das Protokoll der Maus fast dem der Tastatur entspricht, wurde dies auch noch hinzugefügt (ursprünglich sollte es ein Praktikant erledigen). Am Ende der Diplomarbeit wurde noch ein weiteres Thema behandelt, das vorher nur teilweise gelöst werden konnte: ein BIOS für eine VGA-Grafikkarte zu finden, welches als Quelltext vorliegt, komplett verstanden und auf einer FPGA-PCI-Karte eingesetzt werden kann.

1.3 Pflichtenkatalog

Hier sind alle Teilaufgaben aufgelistet, die bearbeitet und gelöst wurden.

- PS/2 Modul: Dieses Modul soll sowohl für Tastatur als auch für Mäuse verwendet werden können. Es soll sich so verhalten, dass der Rechner „denkt“, er habe das entsprechende Gerät wirklich angeschlossen. Auch eine Steuerung des Rechners durch diese Geräte soll möglich sein.
- Floppy Modul: Das Modul soll dem Rechner ein komplettes 3½” Diskettenlaufwerk vortäuschen. Der Host soll von demselben booten können, ein Schreibzugriff auf dieses Laufwerk ist nicht notwendig.
- Grafikkarten-BIOS: Es soll ein BIOS erstellt werden, welches einem Host eine Grafikkarte simuliert, von der ein normales Booten möglich ist. Der notwendige PCI-Core wurde von Stefan Philipp konfiguriert.
- µCsimm Programm: Der TCP-Server soll alle hardwaretechnischen Module verwalten und eine Schnittstelle zu einem TCP-IP Netzwerk bereitstellen. Als Programmiersprache soll C dienen.
- Janus Client: Dieser soll der Verwaltung aller µCsimm’s dienen und auch die Möglichkeit bieten, sich auf einen Rechner aufzuschalten und dort alle Funktionen fernzubedienen (Maus/Tastatur/Floppy/ Grafik). Das Programm soll in Java geschrieben werden, um eine Portierung auf andere Systeme zu erleichtern.

1.4 Überblick

In diesem Abschnitt soll ein Überblick über die Komponenten der einzelnen Teile und deren Aufgaben gegeben werden.

1.4.1 Hardware der CIA-Karte

Technische Daten:

- PCI-Karte
- Altera® Apex® FPGA
- µCsimm; später ein in der FPGA integrierter NIOS®
- RAM (circa 8MB DRAM für den Prozessor und 256 kB für Grafikkarte und Floppy)
- Flash-ROM (circa 2MB für den Prozessor)
- ADC’s zur Messung von Signalen (Spannungen, Temperatur, Ton, ...)
- DAC’s zur Ausgabe von analogen Signalen (Ton, ...)
- Digitale Ein-/Ausgänge (Lüfterdrehzahl, USV, Netzteil, ...)
- Anschlüsse für Maus, Tastatur, Floppy, Netzwerk, IDE und Monitor

Nachfolgende Abbildung zeigt einen schematischen Aufbau der CIA-Karte:

1.4.2 Geplante Funktionen für den Host-Rechner

Technische Daten:

- Linux, Novell®, Windows NT® 4.0, Windows 2000® (**Betriebssystemunabhängigkeit!)**
- Den Host bezüglich Temperatur, Programme*), und so weiter ständig überwachen, eventuell Gegenmaßnahmen einleiten und den Operator informieren
- Komplett scriptsteuerbar
- Ständige Kommunikation der Karten untereinander zur Redundanzbildung
- Gezielte Aktionen ausführbar*). (Datenbank beenden und anschließend wieder starten)
- Treiber für Datenbanken*)
- Treiber für Programme*)

*)gelten nur, falls für das entsprechende System die Treiber existieren

1.4.3 Janus-Client

Der Client ermöglicht es, sich auf den entsprechenden µCsimm’s aufzuschalten. Er verwaltet die Eingabe von Maus und Tastatur und stellt die Grafik auf dem Host-Rechner visuell dar. In einer Liste sollen alle µCsimm‘s erscheinen und ausgewählt werden können. Ein Mehrfachaufschalten ist ebenfalls für später geplant. Das Programm soll direkt nach dem Start einen Scan des kompletten Netzwerkes durchführen und gegebenenfalls Probleme bei einzelnen Rechnern anzeigen, so dass entsprechende Gegenmaßnahmen einfach eingeleitet werden können. Der Client soll in Java erstellt werden. Dies bringt zwar Einbußen bei der Performance mit sich, hat jedoch eine hohe Portabilität zum Guten, was als positiv zu bewerten ist.

1.5 Hilfsmittel

In diesem Abschnitt soll eine kurze Einführung in die verwendete Software und die Geräte gegeben werden.

1.5.1 Einführung in FPGA’s

FPGA’s (**F**ield **P**rogrammable **G**ate **A**rray‘s) sind sogenannte programmierbare Elektronikbausteine. Sie sind eine Weiterentwicklung der GAL’s. Jede FPGA besteht aus vielen einzelnen Logikzellen, die alle gleich aufgebaut sind, aber für verschiedene Aufgaben konfiguriert werden können. Am Eingang einer jeden Logikzelle sitzen frei verdrahtbare UND- und ODER-Gatter, aus denen sich Produktterme mit Hilfe der DNF (**D**isjuktive **N**ormal **F**orm) zusammenbauen lassen. Die so entstehenden Terme können entweder als kombinatorische Logik benutzt und zum Beispiel an andere Logikzellen weitergegeben oder auf die Eingänge eines Registers gelegt werden. Ein solches Register kann in verschiedenen Modi arbeiten: es kann zum Beispiel ein RS-, JK- oder D-Flip-Flop sein. Die Ausgänge einer Logikzelle sind weiter zu Eingängen anderer Logikzellen verdrahtbar oder auch zu den Ausgängen des Bausteins (FPGA). Die Zuordnung zwischen den Logikzellen

untereinander und den Ein-/Ausgängen der FPGA ist komplett frei. Das Verdrahten der einzelnen Bausteine geschieht definitiv nicht von Hand, sondern durch einen sogenannten Fitter, der versucht die eingegebene Logik auf die Logikzellen abzubilden. Dieser Vorgang kann je nach Größe mehrere Minuten in Anspruch nehmen. Als Beschreibung der Logik kommt die Sprache VHDL (**V**ery **H**igh **S**peed **H**ardware **D**escription **L**anguage) zum Einsatz. Auf diese Sprache soll hier nicht weiter eingegangen werden, da sie in vielen Büchern gut erklärt ist und auch den Rahmen dieser Arbeit sprengen würde. Dem interessierten Leser sei hier die Literatur [V1] und [V2] empfohlen.
Als FPGA's kamen die Familien Max, Flex und Apex der Firma Altera zum Einsatz. Altera ist neben Xilinx und Lattice einer der größten Hersteller von programmierbaren Logikbausteinen. Die Wahl fiel auf die Firma Altera, da sie zur Zeit der einzige Hersteller ist, der sogenannte Soft-Prozessoren für seine Produkte anbietet und eine Integration eines richtigen Prozessors (ARM) als ASIC in Kombination mit einer FPGA in einem einzigen Baustein auf dem Markt hat (Excalibur). Speziell NIOS ist hier von Interesse, da er ein relativ kleiner, aber leistungsfähiger Prozessor ist, der nur circa 2000 Logikzellen verbraucht und auch mehrfach in einer FPGA synthetisierbar ist. Des weiteren bietet Altera eine µClinux-Version für den NIOS an. Ein Netzwerkkartenadapter ist ebenfalls direkt mit in der FPGA synthetisierbar, so dass lediglich ein externes RAM und ein ROM (besser Flash-ROM) angeschlossen werden müssen. Ein einziger Chip senkt den Preis für die spätere Produktion. Zudem ist er platzsparender. Zum Zeitpunkt der Diplomarbeit wurde jedoch noch mit dem µCsimm gearbeitet, das ebenfalls unter µClinux betrieben wird. Daher dürfte eine Portierung auf NIOS später nicht allzu komplikationsbehaftet sein.

1.5.2 Verwendete Gerätschaften

Da es sich bei der Diplomarbeit um ein sehr komplexes Thema handelt und aus diesem Grund Spezialwerkzeug benötigt wird, geht dieses Kapitel insbesondere auf die verwendeten Geräte und Entwicklungstools ein.

µCsimm:

Dieses Modul ist einer der kleinsten autonomen Rechner der Welt. Es besteht aus einem Motorola Dragonball EZ-Prozessor mit 16 MHz Taktfrequenz, einem 8MB großen Ram, 2MB Flash-Ram und einem Netzwerkchip Crystal LAN CS8900A. Als Betriebssystem dient µClinux 2.0.38. µClinux ist eine Portierung des standard Linux für Kleinsysteme, die keine MMU (Speicherschutz) besitzen. Das Modul ist neben dem Nachbau des Digilab Max das Kernstück der Arbeit. Der TCP-Server und der CIA-Server wurden hierfür entwickelt.

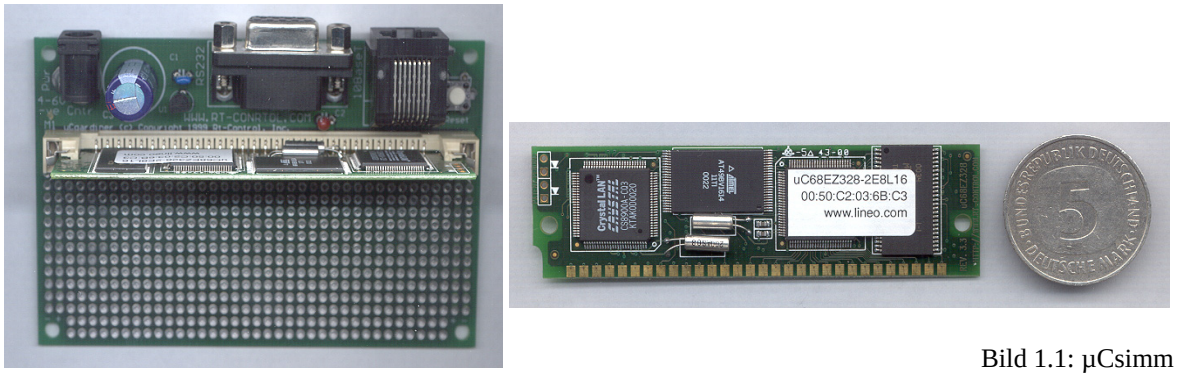


Bild 1.1: µCsimm

Digilab Max von ElCamino:

Das Digilab ist ein Entwicklungsboard von El Camino mit einem Altera Flex 10K10LC84-4. Die FPGA besitzt 576 Logikzellen und sitzt in einem 84-poligen PLCC-Gehäuse. Auf diesem Board wurde zu Beginn die eigentliche Hauptentwicklung vorangetrieben. Später wurde das Board nachgebaut, da in der Version von ElCamino nicht alle Ein- und Ausgänge zur Verfügung stehen.

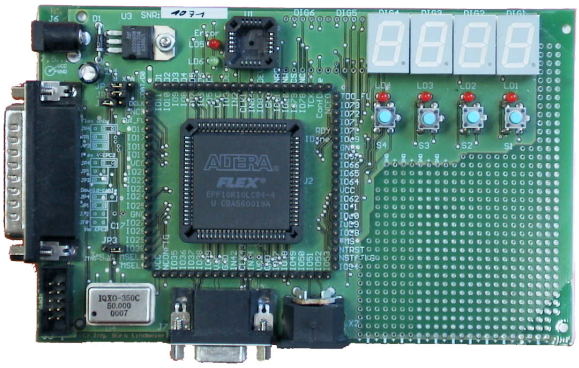
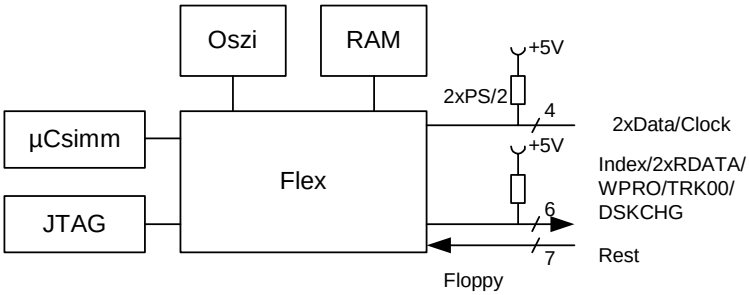


Bild 1.2: Digilab Max

Eigener Nachbau:

Dieser besteht aus einer Lochrasterplatine, auf der die folgenden Bauteile aufgelötet wurden: FPGA Altera® Flex® 10K10LC84-4, 32kB SRAM UM61256AK-15, 33MHz Quarzoszillator, Widerstände für PS/2 und Floppy, Verbinder zum µCsimm, JTAG, PS/2 Maus, PS/2 Tastatur und Floppy Adapter. Dieses Board wurde benötigt, da jenes von El Camino nicht genügend Ein- und Ausgänge bereitstellte.



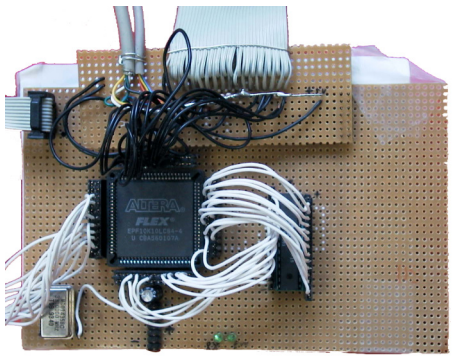


Bild 1.3: Nachbau

Offener Testrechner:

Hierzu diente ein alter ausgemusterter Dell PC mit einem PentiumPRO® 60MHz. Um eine bessere Zugänglichkeit der einzelnen Komponenten zu erreichen, wurde der PC nur als Platine offen betrieben. Hierdurch wurde ein einfacher Wechsel der Karten und Steckverbinder ermöglicht. Der Rechner wurde mit minimalem RAM betrieben, um die Boot-Zeiten möglichst kurz zu halten.

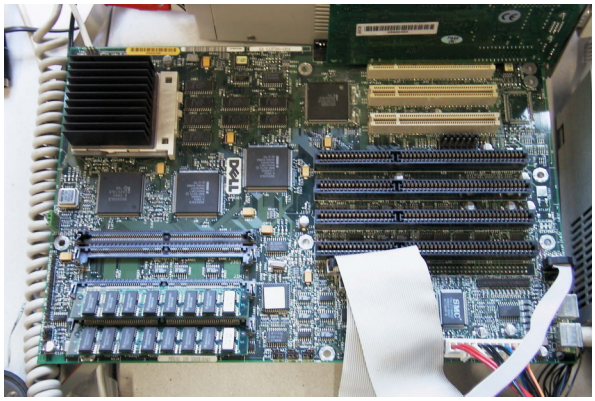


Bild 1.4: Testrechner

Speicheroszilloskop:

Für die Messung schneller analoger Signale stand ein digitales Speicheroszilloskop mit 20 MHz von GoldStar zur Verfügung (OS3020).

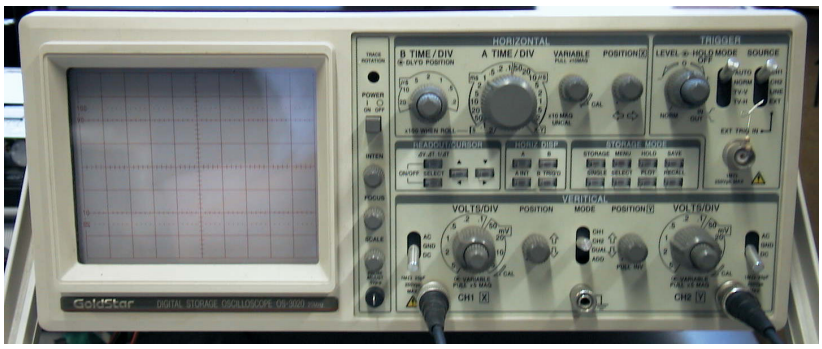


Bild 1.5: Oszilloskop

Logikanalysator LA-100:

Dieser aus dem Hause BrainTechnology stammende Logikanalysator arbeitet mit einer maximalen Abtastfrequenz von 100MHz. Er kann 16 Bits entweder mit 5V oder 3,3V gleichzeitig einlesen und hat eine Speichertiefe von 8kB pro Kanal. Außerdem ist er extern taktbar und kann auf bestimmte Bitmuster triggern.

Bild 1.6: LA-100

Eprom-Simulator:

Der Eprom Simulator aus dem Hause ELV wurde zur Entwicklung des BIOS der Grafikkarte benutzt. Er wird in das Zielsystem gesteckt und verhält sich dort wie ein Eprom. Die darin gespeicherten Daten sind durch eine serielle Schnittstelle veränderbar, so dass eine einfache Neukonfiguration der Daten schnell und komfortabel möglich ist.

„Port 80“ ISA Karte:

Diese Karte, ebenfalls aus dem Hause ELV, diente in erster Linie dem Überprüfen des Host-Rechners. Sie zeigt den aktuellen Status von Port80 an. Dies war eine gute Debugmöglichkeit, sowohl für das VGA-Bios, als auch für Maus und Tastatur. Der angezeigte Port lässt sich auch auf andere Adressen einstellen, so dass zusätzlich weitere Informationen angezeigt werden können. Um das VGA-BIOS zu testen, wurden zu Debugzwecken Daten auf einen freien Port geschrieben und mit der Karte angezeigt.

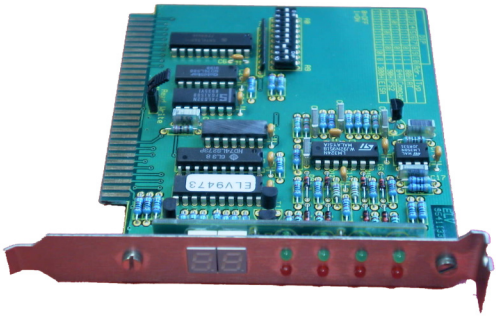


Bild 1.7: „Port 80“ ISA-Karte

Weitere Meßgeräte und Hilfsmittel:

Digitalmultimeter, Eprommer, Epromlöschgerät, Labornetzteil.

1.6 Stand der Technik

Zum Zeitpunkt der Diplomarbeit gab es eine Karte (mit entsprechendem Funktionsumfang) von AMI (**A**merican **M**egatrends **I**ncorporation). Diese Karte (AMI MEGA Rac), speziell für Server entwickelt, kostet circa 1000,-€. Über die Karte ist es möglich, einen Server komplett fernzusteuern. Das Problem hierbei liegt in einer eng mit dem BIOS abgestimmten Software, so dass diese Karte nicht in jedem Rechner funktionieren kann. Der Vorteil ist allerdings, dass keine externen oder internen Steckverbindungen benötigt werden. Alle Daten laufen über den PCI-Bus. Die Managementsoftware, ohne die die Karte nicht sinnvoll einsetzbar ist, kostet circa 7500,-€.
Es existieren am Markt noch weitere Karten, die diese Funktionen übernehmen können; allerdings beläuft sich auch hier der Preis auf mindestens 5000,-€.
Am Ende der Diplomarbeit kam noch eine weitere Karte auf den Markt. Diese kostet circa 1000,-€ und ist bei der Firma Black-Box (http://www.black-box.de) erhältlich. Sie bietet bis auf die Floppyemulation dieselben Funktionalitäten.

2 Floppyemulation

Zu Beginn der Diplomarbeit stand das Thema Floppyemulation im Vordergrund mit dem Ziel, dem Host vorzutäuschen, dass ein Diskettenlaufwerk vorhanden ist und von diesem auch Daten gelesen werden können. Es sollte eine reine Hardwarelösung sein, die keine zusätzliche Software oder Treiber benötigt und die Standardfunktionalität der PC-Hardware ausnutzt. Ursprünglicher Hintergrund war das Aufsetzen neuer Rechner. Speziell bei einem Cluster ist dies sehr sinnvoll, da nicht jeder PC einzeln von einer Diskette gestartet werden muss, sondern auch über einen Batchbetrieb konfiguriert werden kann. Ein Schreibzugriff ist vorerst nicht geplant, da beim Aufsetzen eines Rechners normalerweise keine Schreiboperationen durchgeführt werden. Sobald Netzwerktreiber installiert sind und funktionieren, besteht die Möglichkeit, die Daten über das Netzwerk abzuspeichern. Sollte es zu einem späteren Zeitpunkt nötig sein, Daten über das Diskettenlaufwerk zu speichern, so wird dies mit den Erkenntnissen dieses Kapitels problemlos möglich sein. Lediglich das Wissen muss in die Praxis umgesetzt werden. Im folgenden wird zuerst auf die Hardware und den physikalischen Hintergrund eingegangen.

2.1 Hardware/Codierungsverfahren

Disketten sind magnetische Datenträger in Scheibenform, auf welchen die Daten sequenziell in konzentrischen Spuren abgespeichert werden. Diese wurden Anfang der 70er Jahre entwickelt. Die erste Diskette war noch 8” groß. Da diese Größe sehr unhandlich war, wurden bald danach die 5¼” und 3½” Disketten entwickelt. Die Tabelle “Diskettengrößen” gibt eine Übersicht über bisher verwendete Diskettentypen. Die Disketten werden auch folgendermaßen bezeichnet:

- Maxi-Diskette: 8”
- Mini-Diskette: 5¼”
- Micro-Diskette: 3½”

Darüberhinaus existieren noch weitere Typen: die 3”-Diskette von Schneider oder die 2,5”-Diskette von Sony, die sich jedoch nie richtig durchgesetzt haben. Der häufig verwendete Begriff „Floppy-Disk“ („schlappe Scheibe“) kommt von den ersten beiden Typen, da diese in keiner starren Hülle waren und sich leicht “verbiegen” ließen. Der letzte Typ ist in einer festen Hülle untergebracht und hat zusätzlich noch einen Schutz des Schreib-Lesefensters. Die Sektorerkennung der ersten beiden Typen fand noch über ein Indexloch in der magnetischen Scheibe statt. Bei der Micro-Diskette wird dies über den eigentlichen Antrieb gelöst. Hier gibt es ein zweites Loch, in welches dann ein Stift passt. Dieses Kennzeichen wird auch Indexmarkierung genannt und ist nur ein einziges Mal vorhanden (siehe Bilder “Indexmarkierung”). Es gab früher auch Disketten, welche über mehrere Markierungen verfügten; diese setzten sich ebenfalls nicht durch. Alle Typen haben einen Schreibschutz, jedoch hat derjenige der Micro-Diskette den Vorteil, dass er nicht zwangsläufig unwiderruflich gesetzt werden kann.

Disk-größe	Kapazität unformatiert	Kapazität formatiert	Seiten pro Disk	Spuren pro Seite	Bez. der Dichte	Horiz. Dichte (TPI)	Sektoren pro Spur (DOS)	Drehzahl U/min	Spurbreite (mm)
8"	???	???	2(DS)	???	???	???	???	???	???
5¼"	500 kB	360 kB	2(DS)	40	DD	48	9	300	0,35
5¼"	1,5 MB	1,2 MB	2(DS)	80	HD	98	15	360	0,16
3½"	1,0 MB	720 kB	2(DS)	80	DD	135	9	360	0,115
3½"	2,0 MB	1,44 MB	2(DS)	80	HD	135	18	360	0,115
3½"	4,0 MB	2,88 MB	2(DS)	80	ED	135	36	???	0,115

Tabelle 2.1: Diskettengrößen

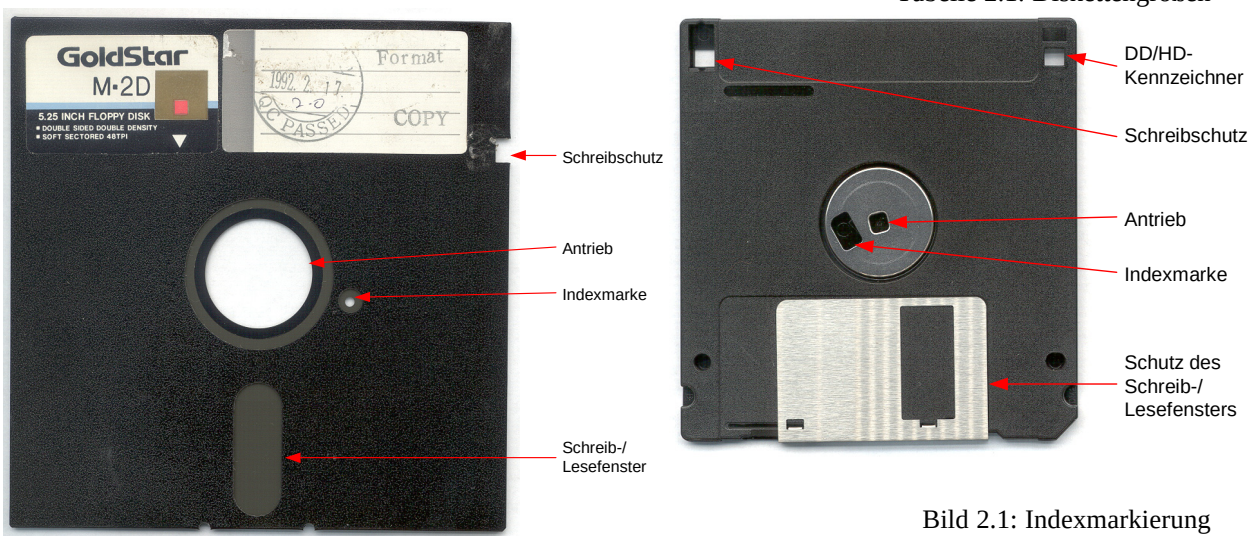


Bild 2.1: Indexmarkierung

Disketten sollten niemals Temperaturen über 50 °C ausgesetzt werden, da sonst die gespeicherten Daten verloren gehen können. Die meisten Hersteller garantieren eine Haltbarkeit der Daten von circa 70 Jahren. Da die hier behandelte Emulation nur die 3½ ” Laufwerke mit 1.44 MB Speicherkapazität behandelt, beziehen sich alle weiteren Angaben auf diese Laufwerke beziehungsweise Medien.
Die Daten werden mit zwei Köpfen auf den Datenträger geschrieben, das Beschreiben erfolgt beidseitig. Der Datenträger wird in 80 sogenannte Tracks aufgeteilt, die jeweils einer konzentrischen Spur entsprechen. Eine weitere Unterteilung der Tracks sind die Sektoren (siehe Bild “Sektoraufbau”).

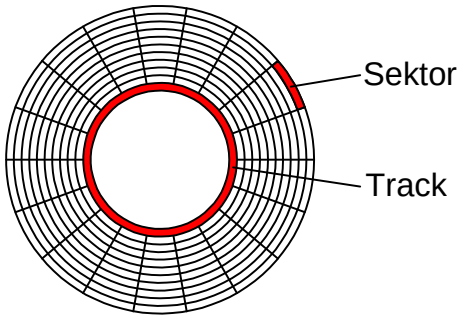


Abbildung 2.1: Sektoraufbau

Wenn nur einzelne Bereiche der Informationen geändert werden sollen, muss somit nicht immer ein kompletter Track geschrieben werden, sondern nur die entsprechende Anzahl Sektoren. Durch diese Maßnahme wird erreicht, dass der Datenträger sich magnetisch nicht so schnell abnutzt und auch bei einem Abbruch des Schreibvorgangs (zum Beispiel bei einem Systemabsturz) nur ein Teil der Daten nicht mehr benutzbar ist. Ein Track enthält immer 18 Sektoren, die softwareseitig eingeteilt sind. Da Disketten mit einer konstanten Geschwindigkeit arbeiten, sind die Daten in den inneren Spuren dichter gepackt als in den äußeren, was nicht zu einer optimalen Ausnutzung des Platzes führt, wie zum Beispiel bei einer CD. Normale Disketten arbeiten standardmäßig mit einer Umdrehungsgeschwindigkeit von 360 U/min. Hieraus ergibt sich eine Bittransferrate von 500 kHz. Andere Rechnersysteme oder Laufwerke arbeiten teilweise mit höheren Umdrehungszahlen, um eine höhere Datentransferrate zu erreichen. Die Datenrate wird aber leider nicht optimal genutzt, wie man leicht selbst feststellen kann, da sonst eine Diskette in wenigen Sekunden ausgelesen wäre. Zwei Gründe sind hier zu nennen: zum einen gibt es nach einem Trackwechsel eine kurze Pause, da sich die Mechanik wieder beruhigen muss, zum anderen liegt es daran, dass der Controller die Daten nicht so schnell verarbeiten kann. Früher konnte in den Controller nur ein einzelner Sektor eingelesen werden, da das hierfür benötigte SRAM noch sehr teuer war. Dies ist zusätzlich ein Grund, warum eine Unterteilung in Sektoren erfolgte. Viele der heutigen Controller sind nur eine Weiterentwicklung und durch den immer noch von der damaligen Zeit bestimmten Befehlssatz kompatibel zu diesen. Sie können nur einen einzelnen Sektor einlesen. Um einen kompletten Track zu lesen, muss der Controller dazu veranlasst werden, Sektor 0 des aktuellen Tracks einzulesen. Dieser wird dann in den Hauptspeicher kopiert. Anschließend werden die restlichen Sektoren gelesen.

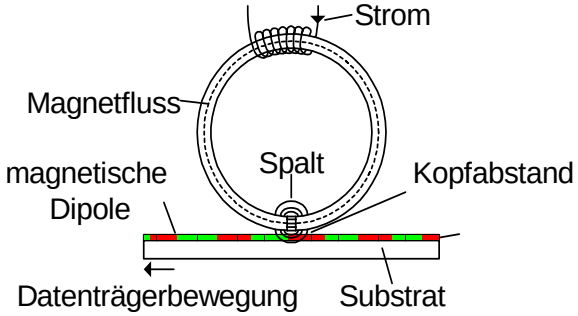


Abbildung 2.2: Ringmagnet

Der Datenträger besteht aus einer Kunststoffscheibe, welche mit einem ferromagnetischen Stoff beschichtet ist. Über einen kleinen Ringmagneten mit einem Spalt werden die Daten nun in diesem Material gelesen/ gespeichert. Aufgrund des Spaltes treten die Magnetfeldlinien aus dem Kern des Ringmagneten aus und gehen durch den Datenträger. Hierbei werden die Elementarmagneten der ferromagnetischen Schicht ausgerichtet. Durch einen Wechsel des Stromes im Magneten können jetzt Informationen abgespeichert werden. Es muss jedoch dafür gesorgt werden, dass keine größeren Stellen entstehen, bei denen kein Flusswechsel stattfindet. Ansonsten könnte ein Problem mit der Synchronisation auftreten, weil sich Disketten nicht mit konstanter Geschwindigkeit drehen. Die Gründe sind hauptsächlich mechanischer Natur. Das weiter unten erwähnte Kodierungsverfahren sorgt dafür, dass keine größeren „leeren“ Stellen entstehen. Beim Lesen der Daten werden die einzelnen magnetisierten Bereiche nicht erkannt, da ein konstantes Magnetfeld keinen Strom in einer Spule induziert, und die Richtung des Magnetfeldes nicht festgestellt werden kann (Anmerkung: es ist zwar möglich, die Richtung des Magnetfeldes herauszufinden, jedoch mit nicht vertretbarem Aufwand). Bei den Polwechseln auf dem Datenträger hingegen wird ein Flusswechsel im Ringmagneten erzeugt, der eine Spannung induziert. Diese Spannung ist sehr gering und muss erst verstärkt werden. Die Flusswechsel sind die eigentlichen Informationsträger.

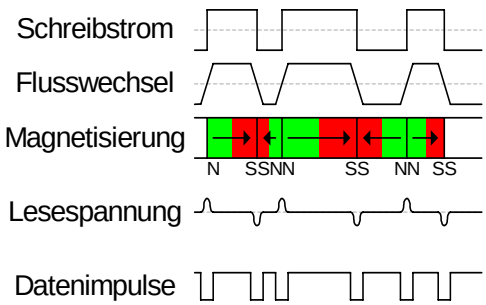


Abbildung 2.3: Ringmagnet

Es wird auch noch darauf hingewiesen, dass die Datensicherheit wesentlich besser geworden ist, was hauptsächlich an der verbesserten Logik der heutigen Controller liegt. Um die Datensicherheit zu erhöhen, werden die Daten nicht genau in dem Timing abgespeichert, wie sie eigentlich das Kodierungsverfahren vorgibt, sondern leicht versetzt abgespeichert (bei dicht aufeinander liegenden Pegelwechseln wird das Timing noch weiter verkürzt). Dieses Verfahren wird Präkompensation genannt. Da die Stellen der Flusswechsel später nicht dort erscheinen, an denen sie geschrieben wurden, sondern durch Überlagerung der Feldlinien an einer leicht versetzten Position, werden die Daten bereits beim Speichern leicht verschoben, und zwar so, dass beim Lesen alles an der richtigen Stelle erscheint. Die folgende Abbildung zeigt zwei einfache Bits, einmal einen Flusswechsel mit positiver und einmal einen mit negativer Induktionsspannung. Bei der Überlagerung von beiden ergeben sich leicht verschobene und von der Amplitude her verringerte Maxima.

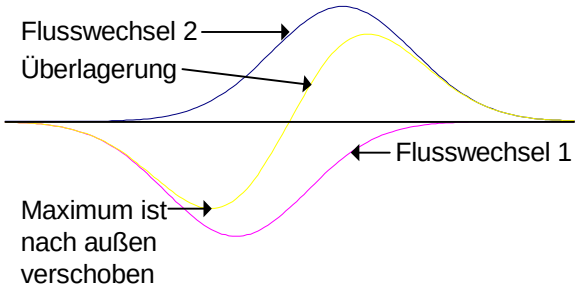


Abbildung 2.4: Bitverschiebung

Bei der Erkennung der gelesenen Daten wird in der Regel folgendes Verfahren benutzt: es wird ein Bereich definiert, in dem gültige Daten liegen dürfen und ein anderer Bereich, bei dem ein Fehler ausgegeben wird. Abbildung „Biterkennung“ zeigt diese Vorgehensweise. Die Erkennung ist im Moment nicht erforderlich; falls später das Schreiben auf Disketten noch hinzugefügt werden soll, ist diese Information aber hilfreich.
Der Ansatz, die Daten einfach auf den Datenträger zu schreiben – das heisst Bit = Strom – ist zwar keine schlechte Idee, jedoch besteht dann das oben bereits erwähnte Problem der Synchronisation. Der Controller muss beim Einlesen der Daten wissen, wann welches Bit kommt. Falls eine längere Sequenz mit gleichen Werten

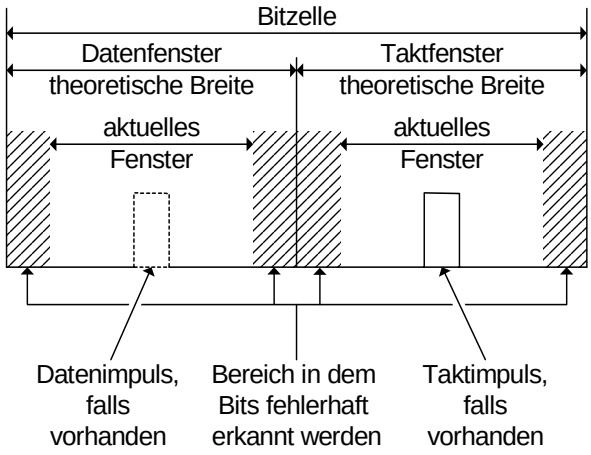


Abbildung 2.5: Biterkennung

folgt, geht die Synchronisation verloren. Um dieses Problem zu umgehen, wurden im Laufe der Zeit verschiedene Kodierungsverfahren entwickelt: RZ, NRZ, NRZI, PM, GCR, RLL, FM, MFM, M²FM, um nur die bekanntesten zu nennen. Da bei Disketten die Verfahren FM, MFM und M²FM die gebräuchlichsten sind, wird hier nur auf diese drei eingegangen. Informationen über andere Verfahren können in der Standardliteratur wie [T1] oder im Internet gefunden werden.
Eines der ersten Kodierungsverfahren war das sogenannte FM (**F**requenz **M**odulation). Es basiert auf folgendem Prinzip: zu Beginn jedes Bits wird ein Pegelwechsel durchgeführt, um eine Synchronisation zu erreichen. In der Mitte des Bits wird nur ein Pegelwechsel durchgeführt, wenn es sich um eine 1 handelt. In Abbildung “Kodierungsverfahren” ist dies zu erkennen. Da bei FM die Pegelwechsel sehr eng beieinander liegen können, wurde dieses Verfahren leicht modifiziert und das MFM (**M**odifizierte **F**requenz **M**odulation) entstand. Dieses Verfahren hat den Vorteil, dass doppelt soviele Daten auf gleichem Raum gespeichert werden können. Die Funktionsweise ist sehr ähnlich: in der Mitte des Bits wird bei einer 1 ein Pegelwechsel durchgeführt, bei einer 0 nur dann am Anfang, falls eine 0 vorausging. Dies eliminiert das Problem, dass Pegelwechsel in einem Abstand von einem halben Byte stehen.
Es existiert noch ein weiteres Verfahren, das M²FM. Bei diesem wurde versucht, die Packungsdichte noch weiter zu erhöhen: bei einer 1 wird wieder in der Mitte ein Pegelwechsel durchgeführt, bei einer 0 jedoch nur dann am Anfang, falls im vorherigen Byte kein Pegelwechsel durchgeführt wurde. Dies hat zur Folge, dass bei Folgen

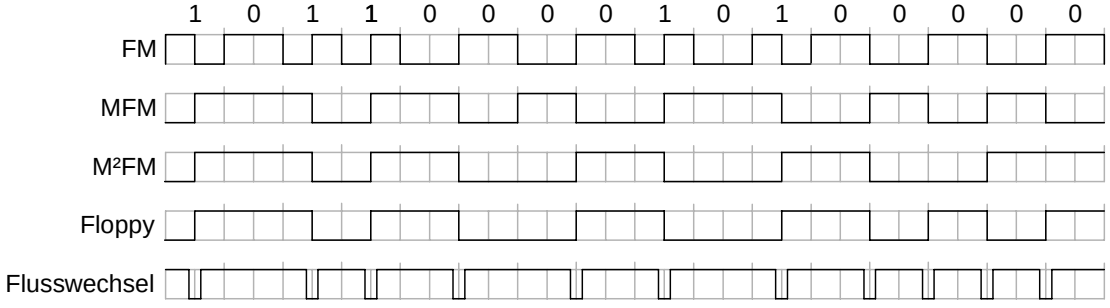


Abbildung 2.6: Kodierungsverfahren

von 0 nur bei jedem zweiten Byte ein Flusswechsel aufgezeichnet wird.
Es wurde lange versucht herauszufinden, wieso keines der oben genannten Protokolle bei der Floppyemulation

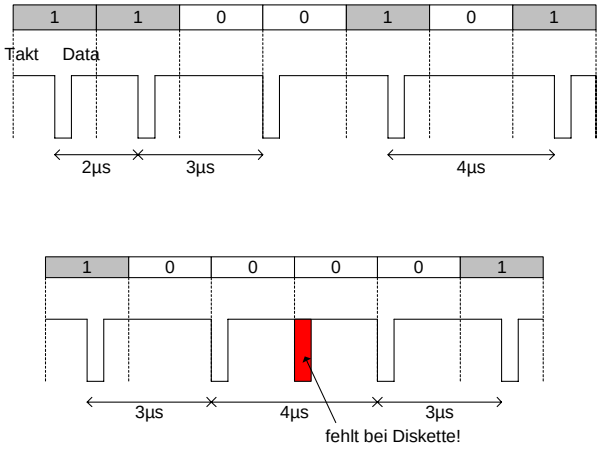


Abbildung 2.7: MFM-Timing

funktionierte. Die Lösung ging aus keiner Dokumentation direkt hervor. Das Protokoll ist eine Mischung aus M²FM und MFM. Implementiert wurde es als MFM Kodierung mit Ausnahme der Abfolge “1-0-0-0-0-1”, bei der kein zusätzlicher Flusswechsel in der Mitte stattfindet. In der Dokumentation zum Laufwerk “W1 Series” (ist auf beiliegender CD zu finden) wird dies zwar erwähnt, allerdings wird behauptet, dass es nur bei der Rahmeninformation der Fall sei, was jedoch nicht stimmt. Das Verfahren wird auch bei den Nutzdaten angewendet. Abbildung “Kodierungsverfahren” zeigt alle Verfahren nochmals im Vergleich, um die Unterschiede zu verdeutlichen.
Die eigentliche Übertragung beinhaltet ausschließlich Flusswechsel, die auf der Leseleitung als negative Impulse vom Diskettenlaufwerk gesendet werden.
Die untere Abbildung „MFM-Timing“ geht nochmals genauer auf die Unterschiede und auf das genaue Timing der Übertragung ein.
Eine Bitzelle ist bei einer Übertragung von 500kb/s genau 2µs lang. Ein Flusswechsel ist ein Low-Impuls von 400ns Länge, der aus einer Messung hervorging. Bei der Emulation wurde eine Dauer von 375ns verwendet, was auf einen günstigen Teiler der Taktfrequenz zurückgeführt werden konnte.
Um herauszufinden, was beim emulierten Kodierungsverfahren falsch war, mussten die Werte einer Original-Floppy mitgeschnitten werden. Da die Speichertiefe eines Oszilloskops und eines Logic-Analysators nicht ausreichten, um einen gesamten Sektor einzulesen, wurde eine kleine VHDL-Schaltung programmiert, welche die Daten des Floppy-Kabels ausliest und in einem angeschlossenen SRAM abspeichert. Es wurde darauf geachtet, dass das Design in dieselbe FPGA programmiert werden kann wie die Floppyemulation, so dass keinerlei bauliche Änderungen vorgenommen werden mussten. Die Daten wurden dann mit Hilfe eines weiteren kleinen C-Programms aufbereitet.
Dabei wurde der unten dargestellte Zustandsübergangsgraph verwendet. Hier wurden einzelne Besonderheiten festgestellt wie zum Beispiel fehlender Taktimpuls bei der Bitkombination 1-0-0-0-0-1. Ebenfalls konnte die Grundlage zur Berechnung der CRC und der Sektorgrößen in Erfahrung gebracht werden, da über die Software nicht auf die Rahmendaten, sondern nur auf die Nutzdaten zugegriffen werden kann.
Anhand der mitgeschnittenen Werte konnte auch festgestellt werden, dass sich die Daten bei

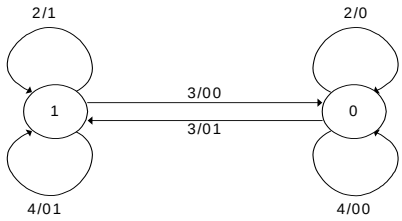


Abbildung 2.8: M²FM-Zustandsautomat

Sektorübergängen manchmal ungewöhnlich verhielten. Ursache hierfür ist, dass auch einzelne Sektoren geschrieben werden können und somit die zuvor gespeicherten Daten überschrieben werden. Da das vorherige Timing nicht genau eingehalten werden kann, überlappen sich somit am Ende die Daten, was zu diesem Effekt führt. Das ist auch der Grund, weshalb sogenannte GAP’s eingeführt werden. Diese sind Bereiche am Anfang und Ende eines Sektors, die mit einem vordefinierten Wert aufgefüllt werden und somit als Puffer dienen.
Die physikalische Speicherung auf dem Datenträger wurde bereits im vorhergehenden Abschnitt erklärt. Nun wird der Aufbau des Tracks eine Abstraktionsstufe höher erläutert. Er ist - wie in Abbildung “Trackaufbau” dargestellt - strukturiert.

Spur Begin		GAP 4A	80x	4E	
		Sync		00	
		IAM	12x	C2 C2 C2 FC	Indexadressmarke
		GAP 1	50x	4E	
Sektor 1		Sync	12x	00	
		IDAM		A1 A1 A1 FE	Indexadressmarke
		ID		sp. kf. sk. gr.	Spur, Kopf, Sektor, Größe
		CRC		xx xx	
		GAP 2	22x	4E	
		Sync	12x	00	
		DAM		A1 A1 A1 FB	Datenadressmarke
		Daten	512x	Daten	Eigentliche Nutzdaten
		CRC		xx xx	
Restliche Sektoren		GAP 3	80x	4E	
		...			Restliche Sektoren
Ende		GAP 4B	872x	4E	Überschreibsicherheitsbereich

Tabelle 2.2: Trackaufbau

Die Emulation dieser Daten stellte kein Problem dar. Einige Punkte sind jedoch zu beachten:

- Track/ Head fangen bei 0 an, Sektor hingegen bei 1
- Size berechnet sich zu $\log_2$(Länge)-7 was in unserem Fall, 512 Byte, genau 2 ergibt

Mit Hilfe der Gap´s kann sich der Floppycontroller auf die Daten synchronisieren. Anschließend kommen die sogenannten Syncs, die darauf hinweisen, dass jetzt die Daten folgen werden. Normalerweise könnte man auch die Werte der Gap´s und Sync´s vertauschen, doch weist genau jene Vorgehensweise einen Vorteil auf. Die Gap's sind gut erkennbar: Abfolge von 3/3/3/3/2/2µs zwischen den Pegelwechseln. Der Kontroller weiß durch die Datenänderung 4Eh->00h, dass Daten folgen werden. Die Sync Impulse sind eine Abfolge von 8x2µs. Dies entspricht genau den 500 kHz Taktfrequenz, die zur Taktgenerierung benötigt werden. Daher sind die Impulse zur Kalibrierung der Lesetaktquelle hervorragend geeignet. Die Gap´s haben des weiteren die Aufgabe, den Unterschied zwischen langsamen und schnellen Kontrollern wegzupuffern. Da die Sektoren auch einzeln geschrieben werden können, besteht sonst die Möglichkeit, dass ein Sektor in den nächsten „hineingehen" könnte.
Die Berechnung der CRC stellte noch eine kleine Herausforderung dar. Zur ihrer Berechnung wird das Verfahren von CRC-16 benutzt, welches durch die X-OR Division den Divisionsrest berechnet. Als Dividend wird der Wert 11021h benutzt; es werden jeweils 16 Bit berücksichtigt. An welcher Stelle die Division angefangen oder welcher Initialwert benutzt wird, konnte leider nicht herausgefunden werden. Falls der Wert 84CFh vor die entsprechende Adressmarke gestellt und anstelle der CRC ein 0000h eingefügt wird, kommt als Ergebnis genau die zu berechnende CRC heraus. Dies ist der große Vorteil der CRC.
Das Abbildung "CRC-Beispiel" veranschaulicht die Arbeitsweise von CRC. Es wird A209h/9Bh=17Bh Rest 6Ch ausgerechnet.
Ein weiterer Vorteil von CRC ist die einfache Berechenbarkeit bei einer seriellen Übertragung. Das unten dargestellte Bild zeigt die Arbeitsweise eines

```
1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 / 1 0 0 1 1 0 1 1 = 1 0 1 1 1 1 0 1 1 , 1 1 0 1 1 0 0
1 0 0 1 1 0 1 1
  0 1 1 1 0 0 1 0
  0 0 0 0 0 0 0 0
    1 1 1 0 0 1 0 0
    1 0 0 1 1 0 1 1
      1 1 1 1 1 1 1 0
      1 0 0 1 1 0 1 1
        1 1 0 0 1 0 1 0
        1 0 0 1 1 0 1 1
          1 0 1 0 0 0 1 1
          1 0 0 1 1 0 1 1
            0 1 1 1 0 0 0 0
            0 0 0 0 0 0 0 0
              1 1 1 0 0 0 0 0
              1 0 0 1 1 0 1 1
                1 1 1 1 0 1 1 1
                1 0 0 1 1 0 1 1
                  1 1 0 1 1 0 0
```

Abbildung 2.9: CRC-Beispiel

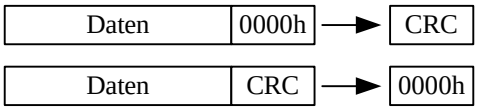


Abbildung 2.10: CRC-Eigenschaft

Berechnungsautomaten. Die CRC ist in diesem Falle nicht 11021h! Eine genaue Beschreibung der mathematischen Funktionsweise des CRC-Verfahrens ist in [T1] zu finden.

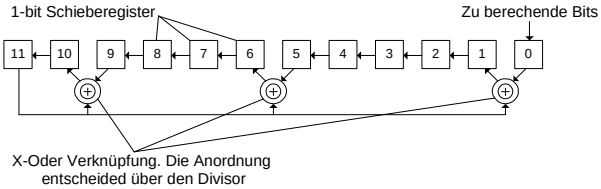


Abbildung 2.11: CRC-Automat

Die physikalische Anbindung des Diskettenlaufwerkes an den Computer geschieht über ein 34-poliges Flachbandkabel. Dieses hat folgende Ader-/Pinbelegung:

Nummer	PC FD	Bezeichnung	Beschreibung
2	⇒	/REDWC	Signal zum Wählen der Schreibdichte
4	-	n.c.	nicht verbunden
6	-	n.c.	nicht verbunden
8	⇐	/INDEX	Indexmarkierungssignal
10	⇒	/MOTEA	Motor von Laufwerk A einschalten
12	⇒	/DRVSB	Laufwerk B soll antworten
14	⇐	/DRVSA	Laufwerk A soll antworten
16	⇒	/MOTEB	Motor von Laufwerk B einschalten
18	⇒	/DIR	Richtung für den Schritt
20	⇐	/STEP	einen anderen Track anwählen (Schritt)
22	⇒	/WDATA	die zu schreibenden Daten
24	⇒	/WGATE	Aktivierungssignal zum Schreiben der Daten
26	⇐	/TRK00	Kopf befindet sich auf Spur 0
28	⇐	/WPT	das Medium ist Schreibgeschützt
30	⇐	/RDATA	die eigentlichen Lesedaten
32	⇒	/SIDE1	Seite 1 wurde ausgewählt
34	⇐	/DSKCHG	ein Diskettenwechsel erfolgte

Tabelle 2.3: Floppy-Kabelbelegung

Alle ungeraden Pins liegen auf Masse und dienen der Abschirmung der Daten voneinander, außerdem minimieren sie Störeinflüsse von außen. Die einzelnen Signale sind Low aktiv. Sie werden über einen Pull-Up-Widerstand nach +5V gezogen und können über einen offenen Kollektor angesteuert werden. Eine versehentliche Zerstörung von Komponenten wird somit weitestgehend ausgeschlossen. Die Abbildung Floppy-Timing zeigt den eigentlichen Aufbau des Interfaces. Für die Spannungen und Ströme gelten die in Tabelle „Spannungs-Charakteristika" angegebenen Werte.
Die Angaben der Floppycharakteristika gelten für /DEVSELx auf "Low"-Pegel. Bei "High"-Pegel sind alle Leitungen offen (Tri-State). /DEVSELx steuert das Laufwerk an. Ist es "High", reagiert das Laufwerk nicht, bei "Low" werden alle Signale generiert. Das Signal /MOTEx steuert den Motor und ist unabhängig vom /DEVSELx-Signal.
Über die Signale /STEP und /DIR wird der Kopf auf die einzelnen Spuren (Tracks) gefahren. Bei einem "Low"-Impuls wird ein Schritt ausgeführt. Die Richtung wird

Eingangssignal	
High	2.4V bis V_{CC}
Low	0 bis 0.6V
Impedanz	1.5 KΩ nach V_{CC}
Ausganssignal	
High	V_{OH}=4.6V ~ V_{CC}(I_{OH}=-2mA)
Low	V_{OL}=0.4V max. (I_{OL}=6mA)

Tabelle 2.4: Spannungs-Charakteristika

über das Signal /DIR angegeben, wobei "Low"-Pegel Schritte in Richtung Spur 79 bedeuten, der innersten Spur. Das genaue Timing ist in der Abbildung „Floppy-Timing" zu sehen. Die Leitungen /WDATA und /WGATE seien nur der Vollständigkeit halber erwähnt. Die Timings der /WDATA Leitung sind die gleichen wie beim Lesen. Sobald /WGATE auf "Low" ist, können Daten geschrieben werden, welche spätestens 4µs danach beginnen müssen. Dasselbe gilt für das Ende des Schreibens, spätestens 4µs nach den letzten Daten muss /WGATE wieder auf "High" zurückgegangen sein. Das /SIDE1 Signal gibt an, auf welcher Seite gelesen oder geschrieben wird. Ein Seitenwechsel während des Schreibens ist nicht gestattet, beim Lesevorgang kann jedoch ein ständiger Wechsel stattfinden, was bei der Emulation zu Geschwindigkeitsproblemen führte. "Low" Pegel bedeutet Seite 1, "High" hingegen Seite 0. Das Signal /TRK00 dient der Synchronisation mit den Spuren. Sobald Spur 0 erreicht wurde, geht dieses Signal auf "Low". Falls die Step-Impulse zu schnell kamen, ist es möglich, dass vielleicht ein Schritt nicht ordnungsgemäß ausgeführt wurde. Auch dies ist dann an diesem Signal erkennbar. Um die Anzahl der Spuren herauszufinden, hat IBM einen Standard eingeführt, der dies ermöglicht: der Kopf wird einfach mehr als 40 Schritte nach innen gefahren. Wenn das Laufwerk nur 40 Spuren hat, wird der Kopf anschlagen, und die Position bleibt gleich. Anschließend wird der Kopf wieder auf Position 0 gefahren. Falls nun nach 40 Schritten bereits Position 0 erreicht wird, handelt es sich um ein Laufwerk mit 40 Spuren, ansonsten um eines mit 80. Bei schreibgeschützter Diskette befindet sich /WPRO auf "Low" Pegel. Das /DSKCHG Signal ist "Low", sobald entweder das Laufwerk Strom bekommen hat – Einschalten des Rechners – oder eine Diskette ausgeworfen wurde. Das Signal wird wieder auf "High" gesetzt, sobald ein Spurwechsel mit dem /STEP-Signal vorgenommen wurde. Hierüber scheinen auch einige Rechner festzustellen, ob eine Diskette eingelegt ist. In Tabelle „Spannungs-Charakteristika" sind noch einige weitere Daten aufgeführt.

2.2 VHDL Design

Zuerst wurde versucht die Daten direkt über den SPI des µCsimm's auszugeben, die Codierung sollte ein GAL erledigen. Leider scheiterte dieses Verfahren: den Datenstrom aufrechtzuerhalten ist unmöglich, da der SPI des Prozessors erst nach vollständiger Übertragung signalisiert, dass er neue Daten haben möchte. Die einzige Möglichkeit, dies zu erreichen, wäre die, einen FIFO extern dazwischenzuschalten. Selbst über Interrupts wäre dies nicht machbar, da die Latenzzeit des Prozessors zu groß ist. Von einer Übertragung durch das µCsimm ist aus zwei Gründen abzuraten: erstens spricht die Kompatibilität dagegen, da das ganze Projekt später einmal auf einen anderen Prozessor portiert werden soll, höchstwahrscheinlich NIOS®. Zweitens wäre das System ständig damit beschäftigt, immer wieder die gleichen Daten zu übertragen und dies mit einer nicht geringen Bandbreite von 500kbps.
Der zweite Ansatz war, die Daten in einem externen Ram zwischenzuspeichern. Da 2 MB sehr viel sind und das SRAM zudem noch sehr teuer, wurde der Ansatz gewählt, immer nur einen Track im externen RAM zu halten und den Rest im DRAM des µCsimm's. Eine Spur ist 2*12790 Byte groß, einmal für Kopf 0 und einmal für Kopf 1, was in einem 32 kB Ram einfach unterzubringen ist, zum Beispiel eine Kombination mit dem Video-RAM aus der Grafikkartenemulation, in welchem noch genügend ungenutzter Platz zur Verfügung steht. Wenn ein Trackwechsel erkannt wurde, müssen nur die Daten des

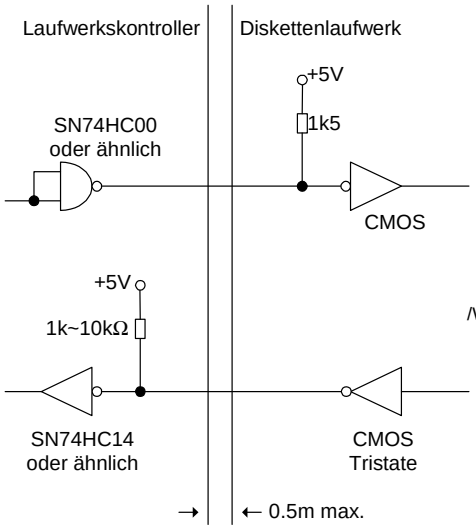


Abbildung 2.12: Physikalischer Floppyaufbau

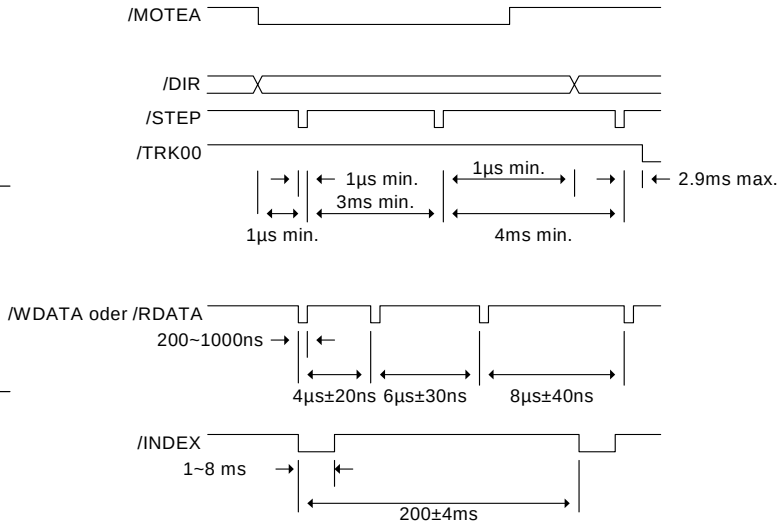


Abbildung 2.13: Floppy-Timing

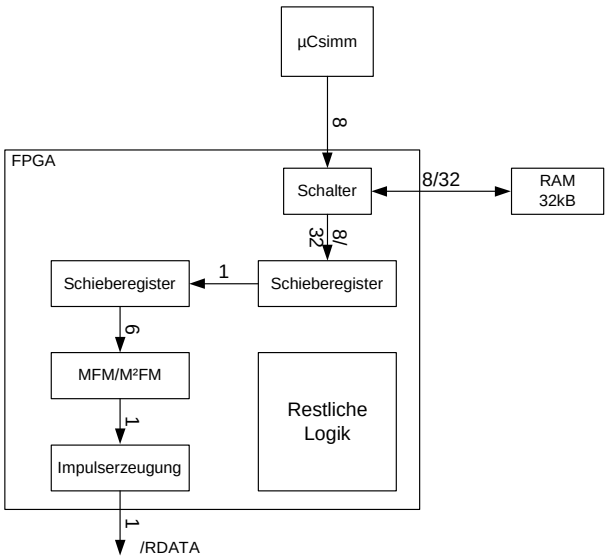


Abbildung 2.12: Floppy-FPGA-Aufbau

neuen Tracks in das SRAM geschrieben werden. Die im SRAM liegenden Daten können so asynchron und unabhängig vom Prozessor an den Host übertragen werden. Das größte Problem hierbei ist die Geschwindigkeit, mit der die Daten in das SRAM geschrieben werden.

Innerhalb dieser Diplomarbeit wurde nur ein Test durchgeführt, über Polling festzustellen, ob ein Spurwechsel stattgefunden hat. Ein kompletter Zusammenbau der einzelnen Teile konnte leider aus Zeitgründen nicht stattfinden. Stefan Philipp hat mehrfach gezeigt, dass von einer solchen emulierten Diskette gebootet werden kann. Die Integration ist größtenteils angedacht und teilweise auch schon umgesetzt. Lediglich kleinere Teile müssen noch realisiert werden, insbesondere die Anbindung mit dem Interrupthandler.

Das VHDL Design besteht aus mehreren Teilen: "floppy_mfm.vhd" wickelt die eigentliche Kodierung der Daten ab. Dies wird, wie im vorangegangenen Kapitel beschrieben, erledigt. Das komplette Modul beinhaltet zwei Schieberegister. Das Erste besitzt einen parallelen Eingang, über den die 32 Bit breiten Daten aus dem RAM in einen seriellen Strom konvertiert werden. Die nun serialisierten Daten werden in ein weiteres Schieberegister mit 6 Stufen geschoben. Über einen parallelen Ausgang des Schieberegisters werden die Daten zur weiteren Kodierung ausgegeben; das für die Kodierung aktuelle Bit ist Bit 2. Das Signal "except" gibt die Ausnahme zwischen dem MFM und dem bei Disketten verwendeten Kodierungsverfahren an und initiiert die entsprechende Aktion, dass kein Takt-Impuls gesendet werden muss. Der MFM-Mechanismus besteht aus einem 4 Bit Zähler, der das genaue Timing der Daten vorgibt. Ein Low-Impuls auf der Leitung "next_data" fordert neue Daten vom SRAM an. Die Abbildung "Floppy-Encoding" zeigt die genaue Vorgehensweise.

Die Behandlung der aktuellen Position des Kopfes übernimmt der Teil "floppy_track.vhd". Ein Step Impuls löst einen Spurwechsel aus. Ein Anschlag bei Position 0 oder 79 wird ebenfalls realisiert. Desweiteren wird hier das /TRK00 Signal generiert.

Die Steuerung der kompletten Einheit übernimmt "floppy_ctrl.vhd". Das µCsimm wird durch die IRQ-Leitung über jegliche Änderung informiert. Darüberhinaus wird die Anforderung der Daten vom SRAM übernommen.

Diese Teile wurden ursprünglich im Laufe dieser Arbeit entwickelt; Stefan Philipp führte eine Implementierung bei seinem Design durch, welches später die endgültige Version geben sollte. Daher war es sinnvoll, die Anbindung an das SRAM durch ihn realisieren zu lassen, zumal auf dieses auch parallel durch den PCI-Bus zugegriffen werden kann. Die Floppyemulation war der erste Teil, der in der Diplomarbeit in Angriff genommen wurde, wobei die eigentliche Schnittstelle zum µCsimm (Thema von Stefan Philipp) zu diesem Zeitpunkt noch nicht existierte. Sonst wäre sicherlich eine komplett lauffähige Version zustande gekommen.

2.3 Software

Die Software besteht aus mehreren verschiedenen Teilen, die voneinander unabhängige Aufgaben erledigen:

Zuerst muss eine Diskette komplett ausgelesen werden. Es wäre schön gewesen, ein Tool zur Hand zu haben, welches eine Diskette auf Low-Level Ebene auslesen kann. Leider konnte auch im Internet kein entsprechendes Werkzeug gefunden werden. Daher wurde das Auslesen einer Diskette mit DriveImage 5.0 erledigt. Wichtig ist, dass das Abspeichern der Daten unkomprimiert erfolgt. Die Daten liegen dann so vor, wie sie auch auf der Diskette zu finden sind, abgesehen von den Rahmendaten. Ein so generiertes Abbild einer Diskette ist 1.44 MB groß, entsprechend der Kapazität einer formatierten Diskette. Um die fehlenden Daten zu erzeugen, wurde ein C-Programm geschrieben, das die Image-Datei einliest, die fehlenden Informationen wie CRC's, IDAM's, Sync's und GAP's erzeugt und anschließend die Daten abspeichert. Die so entstandenen Daten können direkt an den Host gesendet werden. Die Generierung der fehlenden Daten kann später einmal entweder auf µCsimm Seite

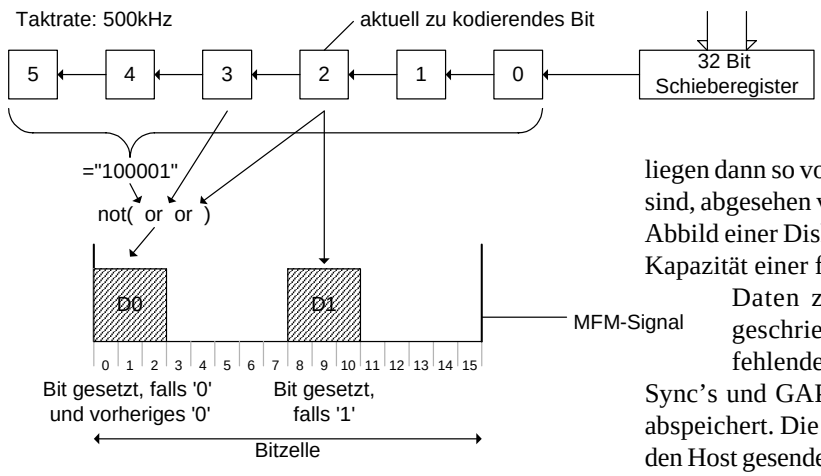


Abbildung 2.13: Floppy-Encoding

oder sogar direkt in der FPGA erledigt werden. Dies würde einiges an Platz und Zeit sparen.
Auch wäre es ratsam, ein Programm zu schreiben, welches die Daten in einem so erzeugten Image direkt verändern kann, das heisst das Filesystem einer Diskette nachzuempfinden. Hierfür existieren aber sicherlich schon Programme, besonders auf Unix-Seite, die an diese Problemstellung angepasst werden können. Damit würde dann ein einfacher Zugriff auf die serialisierten Diskettendaten ermöglicht werden.

Die im vorigen Abschnitt erzeugten Abbilder können einfach auf einem Server, im Falle dieser Diplomarbeit auf einem NFS-Server abgelegt werden. Das µCsimm muss dann dieses Laufwerk gemountet haben, um die Images lesen zu können. Bei der bisherigen Programmversion konnte das Testprogramm nur ein festes Image einlesen. Dieses wird dann im Hauptspeicher abgelegt und von dort aus, je nach Anforderung, über die FPGA an das angeschlossenen SRAM übertragen. Danach folgt die Übermittlung an den eigentlichen Host.

3 Tastatur-/Mausemulation

Zu Beginn der Diplomarbeit sollte ursprünglich nur eine Tastaturemulation programmiert werden. Nachdem die Emulation erfolgreich funktionierte und herausgefunden wurde, dass die Maus fast identisch mit der Tastatur ist, wurde die Mausemulation gleich miterledigt. Wegen der geringen Unterschiede werden beide Geräte in diesem Kapitel gemeinsam behandelt. Diskrepanzen werden an entsprechender Stelle herausgearbeitet.

3.1 Hardware/Übertragungsprotokoll

Die PS/2 Schnittstelle überträgt Daten über zwei Leitungen: eine Takt- und eine Datenleitung. Bei vielen älteren Mainboards existiert noch ein DIN-Stecker, der genauso funktioniert. Aus diesem Grund wird auf die ältere Norm nicht weiter eingegangen. Ein normales (PS/2 nach DIN) Adapterkabel, welches überall im Handel zu erwerben ist, funktioniert. Bei Mäusen gibt es noch eine serielle Übertragung. Da diese Art der Anbindung über die serielle Schnittstelle so gut wie nicht mehr benutzt wird, im Gegenteil der neue PC-Standard sieht keine serielle Schnittstelle mehr vor, und jeder neue Rechner zwei PS/2-Schnittstellen (für Maus und Tastatur) besitzt, wurde auf die serielle Norm nicht weiter eingegangen. Vielmehr ist es mittlerweile interessanter, beide Geräte über USB anzubinden, da sich dieser Standard immer mehr durchsetzt.

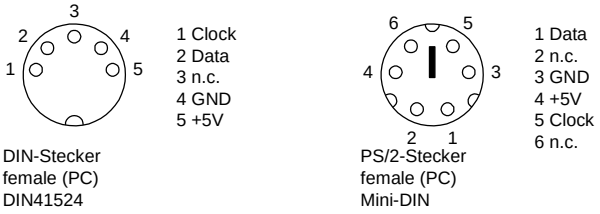


Abbildung 3.1: PS/2-Stecker

Das eigentliche PS/2 Protokoll basiert auf einer getakteten seriellen Datenübertragung. Zusätzlich werden Synchronisationsdaten übertragen, die die Richtung der auszuführenden Übertragung angeben oder auch dem Host die Möglichkeit eröffnen, das Senden von Daten durch das Gerät zu unterbinden.
Die eigentliche Hardware basiert auf zwei Leitungen, die an beiden Enden über Pull-Up Widerstände nach +5V gezogen werden; an beiden Enden kann somit der Pegel gelesen werden. Um Daten auf die Leitung zu legen, wird über einen offenen Kollektor die entsprechende Leitung nach GND gezogen. Hierdurch werden Kurzschlüsse und Kollisionen umgangen. Der ganze Aufbau ähnelt stark dem des I²C-Busses. Nachfolgende Abbildung gibt den Aufbau einer Leitung wieder.
Die Takterzeugung liegt im Aufgabenbereich des angeschlossenen Gerätes (Tastatur/Maus). Bei fallender Taktflanke werden die vom Gerät kommenden Daten durch den Host übernommen. Das Gerät hingegen übernimmt die Daten immer bei steigender Flanke. Hieraus ergibt sich eine mit steigender Flanke arbeitende Statemachine auf Geräteseite und eine mit fallender auf

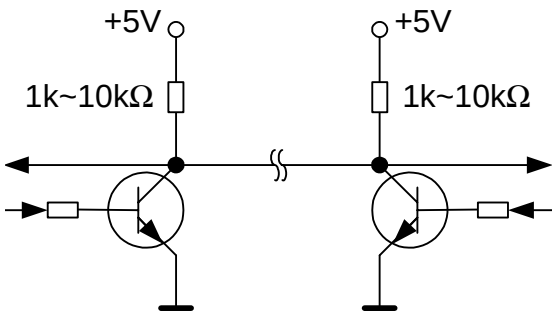


Abbildung 3.2: Aufbau PS/2-Schnittstelle

Hostseite. Dadurch sind die Daten auf der Seite des Geräts bis zur fallenden Flanke stabil und können somit bei steigender Flanke sicher übernommen werden.
Die Taktfrequenz der Übertragung liegt in einem weiten Bereich von 10 bis 33kHz; die meisten Geräte hingegen arbeiten zwischen 10 und 20 kHz. Es wird empfohlen, eine Taktrate von 15kHz zu benutzen, falls andere Anforderungen dem nicht entgegenstehen.
Als Protokoll wird eine 8-O-1 Übertragung verwendet, das heißt es werden folgende Bits nacheinander übertragen:

- 1 Startbit (immer 0)
- 8 Datenbits (LSB wird als erste übertragen)
- 1 Paritybit (ungerade Parität, ungerade Anzahl von Einsern)
- 1 Stopbit (immer 1)
- (1) Acknownledgebit (dieses wird nur bei der Datenübertragung vom Rechner zum Gerät benutzt)

Die allgemeine Warteschleife beinhaltet folgenden Zyklus:

1. Ist die Datenleitung Low, gehe zu emfangen
2. Falls Daten gesendet werden sollen, müssen beide Leitungen für mindestens 50 µs auf High sein

Falls nun das Gerät etwas an den Host senden möchte, hat es folgende Funktionen zu erfüllen:

1. Grundbedingung siehe oben
2. Daten auf 0 setzen
3. Takt auf 0 und wieder auf 1 setzen, mit einem der Übertragungsfrequenz entsprechenden Timing
4. Datenbit ausgeben
5. Takt wieder auf 0 und dann auf 1 setzen, wie bei 3
6. Schritte 4 und 5 werden solange wiederholt, bis alle Daten ausgegeben sind
7. Paritybit und Stopbit auf die gleiche Weise senden

Falls nun der Host Daten senden möchte, ist folgendes zu tun:

1. Grundbedingung siehe oben
2. Taktimpulse ausgeben und bei steigender Flanke die Daten übernehmen (Daten+Parity)

Abbildung 3.3: PS/2-Protokoll

3. Zusätzlich noch einmal eine 0 ausgeben mit Taktimpuls

Die Kommunikation der Daten zwischen Host und Gerät läuft nach folgendem Schema ab: sollte eine Taste gedrückt werden, wird der zugehörige Scancode an den Host geandt. Jeder Taste ist genau ein Scancode zugeordnet, der aus mindestens einem Byte besteht. Genauso kann der Host auch Daten an das Gerät senden; das Gerät antwortet in der Regel mit entsprechenden Bytes. Wenn der Host keine Daten empfangen will, kann er durch Setzen der Taktleitung auf Low den Empfang sperren. Das Gerät muss diesen Zustand erkennen und gegebenenfalls warten.
Im Folgenden wird auf die Besonderheiten der einzelnen Geräte eingegangen.

3.1.1 Tastatur

Bei der Tastatur gibt es drei verschiedene Versionen: Set 1, 2 und 3, die der Art der Tastenübertragung entsprechen. Jedes Set hat eine andere Scancodeerzeugung der Tasten und kann im Anhang 11.4 gefunden werden. Set 1 ist ein altes Format, welches bei alten XT-Rechnern zu finden ist. Da diese Rechner heute so gut wie nicht mehr existieren (<i286), wurde hierauf nicht weiter eingegangen. Zu erwähnen sei jedoch, dass die normalen Tastaturen dieses Protokoll in der Regel noch unterstützen. Vom Aufbau her ist es den beiden neueren ähnlich, lediglich beim Loslassen einer Taste wird zum Keycode eine 80h addiert, anstatt ein Byte voranzustellen. Das Tastaturlayout ist ebenfalls leicht unterschiedlich.

Die Sets 2 und 3 haben nur geringe Unterschiede, die den normalen Betrieb nicht betreffen. Da im Rahmen dieser Arbeit nur der normale Betrieb von Interesse ist, sei der interessierte Leser auf ein eigenes Studium der Unterschiede im Anhang [11.4] verwiesen. Jede Tastatur unterstützt normalerweise alle drei Scancode Set's. Die Umschaltung erfolgt durch einen entsprechenden Befehl (F0h), der weiter unten aufgeführt wird. Im folgenden wird daher nur Set 2 behandelt. Siehe Abbildung „MF-II Tastaturlayout".
Sobald eine Taste gedrückt wird, muss der zu dieser Taste gehörende Keycode gesendet werden. Nach einer Wartezeit (Delay), die eingestellt werden kann, sendet die Tastatur den Keycode mit einer bestimmbaren Wiederholrate (Repeat Rate) immer wieder, die Frequenz hierfür ist ebenfalls wählbar. Sobald die Taste losgelassen wird, wird das Byte F0h, gefolgt von dem gleichen Keycode, gesendet. Dies teilt dem Host mit, dass die Taste losgelassen wurde. Einige Sondertasten senden nicht einen einzigen Code, sondern eine Abfolge von Codes (siehe Anhang). Für diese Tasten gibt es auch eine spezielle Sequenz, welche beim Loslassen übertragen wird; sie ist ebenfalls in der Tabelle angegeben. Die komplette Berechnung und Verwaltung der Tasten übernimmt das Ja[illegible] erfolgt durch den "tcpserv" des µCsimm's, welcher auch die Verwaltung von Bytes, die vom Host an das Gerät gesendet werden, übernimmt. Eine spätere Portierung der Scancodeverwaltung in den TCP-Server ist auf jeden Fall sinnvoll; jedoch ist zum Experimentieren eine Behandlung im Java-Client einfacher zu bewerkstelligen.

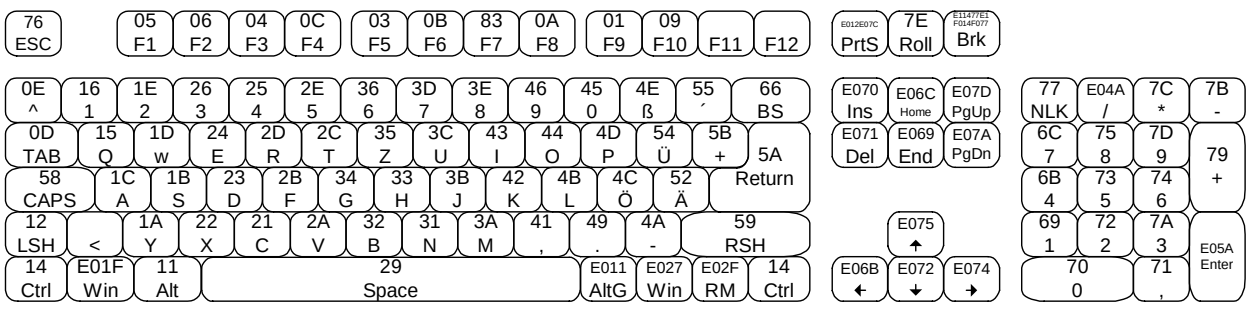


Abbildung 3.4: MF-II Tastaturlayout

Viel wichtiger sind die Kommandos, welche zur Steuerung der Tastatur verwendet werden. Auf sie wird nun näher eingegangen.
Die Befehle, die der Rechner an die Tastatur senden kann, sind in Tabelle „PC-Tastatur Befehle“ angegeben.
Die Wartezeit hat folgende Bedeutung:

Bits 5+6	Watezeit
00h	0,25s
20h	0,50s
40h	0,75s
60h	1,00s

Die Wiederholrate ist in Tabelle „Wiederholrate“ angegeben.
Die Befehle, welche die Tastatur an den Host senden kann, sind in Tabelle „Tastatur-Host Befehle“ aufgelistet.
Die Bezeichnungen “make/break/typematic” bedeuten folgendes: beim Modus “make” sendet die Tastatur nur die Tasten, wenn sie gerade gedrückt werden. Es gibt keine Wiederholungen, und auch gleichzeitig gedrückte Tasten werden nicht erkannt. Im Modus “make/break” wird jeweils das Drücken einer Taste gemeldet. Sobald die Taste losgelassen wird, erfolgt eine Übermittlung an den Host. Hierbei ist ein Erkennen mehrerer gleichzeitig gedrückter Tasten möglich. Wiederholraten müssen durch das Betriebssystem selbst erzeugt werden. Der Modus typematic ist der wohl geläufigste: Tasten werden gemeldet, sobald sie gedrückt werden. Nach der oben spezifizierten Wartezeit wird dieser Wert mit der eingestellten Wiederholrate erneut gesendet. Beim Loslassen wird der diesem Ereignis entsprechende Scancode gesendet. Die Tasten müssen nicht zwangsläufig alle im gleichen Modus arbeiten.
Die Tastatur kann noch einige weitere Befehle senden: AAh bedeutet, dass der Selbsttest der Tastatur erfolgreich durchgeführt wurde. FCh hingegen bedeutet, dass beim

Befehl	Zu beantworten mit	Funktion
EDh	FAh, (a), FAh	Leuchtdioden setzen a.0 Scroll, a.1 Num, a.2 Caps
EEh	EEh	Echo, nur mit EEh zu beantworten
F0h	FAh, (01-03), FAh	Scancode setzen, Scancode übernehmen und speichern
	FAh, (00), (a)	Scancode abfragen, aktiven zurückgeben
F2h	FAh, ABh, 83h	ID der Tastatur abfragen
F3h	FAh, (a), FAh	Wartezeit und Wiederholrate setzen, Bedeutung siehe unten
F4h	FAh	Tastatur aktivieren (beginnen, Zeichen zu senden)
F5h	FAh	Tastatur deaktivieren (Zeichen dürfen nicht mehr gesendet werden)
F6h	FAh	Standardeinstellungen laden, LED Zustände bleiben erhalten
F7h	FAh	Alle Tasten auf typematic stellen
F8h	FAh	Alle Tasten auf make/break stellen
F9h	FAh	Alle Tasten auf make stellen
FAh	FAh	Alle Tasten auf typematic/make/break stellen
FBh	FAh, (a), FAh	Taste a auf Modus typematic setzen
FCh	FAh, (a), FAh	Taste a auf Modus make/break setzen
FDh	FAh, (a), FAh	Taste a auf Modus make setzen
FEh	(a)	Letztes Byte nochmals senden
FFh	FAh, AAh	Tastatur neu starten (Reset)

Tabelle 3.1: Host-Tastatur Befehle

Befehl	Funktion
00h	Tastatur hat einen Fehler gefunden/Tastenpuffer Überlauf (bei Scancode Set 2 oder 3)
83ABh	Tastatur ID
AAh	Selbsttest war erfolgreich
EEh	"Echo" - EEh wird zurückgesendet
FAh	Bestätigung (Acknownledge)
FCh	Selbsttest ist fehlgeschlagen
FEh	Letzes Byte nochmals senden
FFh	Tastatur hat einen Fehler gefunden/Tastaturpuffer Überlauf (bei Scancode Set 1)

Tabelle 3.2: Tastatur-Host Befehle

Bits 0-4	Zeichen/s	Bits 0-4	Zeichen/s	Bits 0-4	Zeichen/s	Bits 0-4	Zeichen/s
00h	2,0	08h	4,0	10h	8,0	18h	16,0
01h	2,1	09h	4,3	11h	8,6	19h	17,1
02h	2,3	0Ah	4,6	12h	9,2	1Ah	18,5
03h	2,5	0Bh	5,0	13h	10,0	1Bh	20,0
04h	2,7	0Ch	5,5	14h	10,9	1Ch	21,8
05h	3,0	0Dh	6,0	15h	12,0	1Dh	24,0
06h	3,3	0Eh	6,7	16h	13,3	1Eh	26,7
07h	3,7	0Fh	7,5	17h	15,0	1Fh	30,0

Tabelle 3.3: Wiederholrate

Selbsttest ein Fehler auftrat. Wenn im normalen Betriebsmodus ein Überlauf des Tastenpuffers stattgefunden hat, sendet die Tastatur, sofern Set 1 aktiv ist, ein FFh, bei Set 2 und 3 ein 00h an den Host.
FAh ist die Bestätigung, dass ein Befehl empfangen wurde und dass entsprechende Reaktionen eingeleitet werden. Diese Bestätigung wird in der Regel spätestens 20 ms nach dem entsprechenden Befehl gesendet.
Nach einer Reset-Aufforderung (FFh) und der Bestätigung durch das Gerät (FAh) führt die Tastatur einen Neustart durch. Spätestens 500ms danach muss die Tastatur das PowerOnReset-Byte (AAh) senden, um zu signalisieren, dass sie nun wieder betriebsbereit ist. Geschieht dies nicht, bringt das BIOS in der Regel die Meldung "Keyboard Error". Es ist wichtig zu wissen, dass dieser Datenaustausch zu einem sehr frühen Zeitpunkt geschieht. Erst ein paar Sekunden später fragt das BIOS die Tastatur-ID ab und initialisiert alle Werte. Die Erklärung der restlichen Werte ist in der Tabelle zu finden.

3.1.2 Maus

Eine Übersicht über alle Befehle, die der Host an die Maus senden kann, sind in Tabelle „Host-MausBefehle" dargestellt.
Der Unterschied zwischen dem Streammodus und dem Remotemodus ist folgender: im Streammodus, welcher in der Regel benutzt wird, sendet die Maus Bewegungsdaten, sobald sie bewegt wird (Tastendrücke ebenfalls). Im Remotemodus werden nur Bewegungsdaten an den Host gesendet, wenn dieser es mit EBh anfordert. Die Bewegungsdaten bestehen aus drei Bytes, die folgendermaßen aufgebaut sind:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	YV	XV	YS	XS	0	M	R	L
2	X-Delta							
3	Y-Delta							

"M", "R", "L" stehen für die Maustasten. Eine 1 bedeutet, dass diese Taste gerade gedrückt ist. Die Bits "XS" und "YS" sind Vorzeichenbits der Bewegung. Bei 0 wurde eine Bewegung nach rechts beziehungsweise unten ausgeführt, bei 1 nach links beziehungsweise oben. Die Delta Werte enthalten den Wert der Bewegung. Zusammen mit dem Vorzeichen-Bit ergibt sich eine 9 Bit vorzeichenbehaftete Zahl in Zweierkomplementdarstellung. Eine Bewegung um eine einzige Position in negativer Richtung hat dann folgendes Aussehen: Vorzeichen (1), Delta-Wert (FFh). "XV" und "YV" sind Bits, die einen Überlauf der Bewegung angeben. Wie das Betriebssystem darauf reagiert, ist ihm dann selbst überlassen.
Die Auflösung, welche durch den Befehl E8h verändert werden kann, hat nachstehende Auswirkung:

Wert	Impulse/mm
00h	1
01h	2
02h	4
03h	8

Die Abtastfrequenz kann über den Befehl F3h gewählt werden und wird in Abtastungen pro Sekunde angegeben. Erlaubt sind folgende Werte: 10, 20, 40, 60, 80, 100 und 200 (dezimal). Andere Werte werden sicherlich auch von jedem Gerät angenommen, jedoch werden sie dazu benutzt, verschiedene Maustypen zu kennzeichnen.
Der Befehl Status-Request (E9h) gibt in drei Bytes den aktuellen Status der Maus zurück; diese sind wie folgt aufgebaut:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Remote	Enable	Scaling	0	L	M	R
2	0	0	0	0	0	0	Resolution	
3	Samplerate							

Remote: 1= Remote-Modus, 0= Stream-Modus
Enable: 1= Daten Senden, 0= keine Daten senden, dies ist nur im Stream-Modus aktiv
Scaling: 1= 2:1, 0=1:1
Left, ...: 1= entsprechende Taste gedrückt.
Resolution: siehe E8h
Samplerate: siehe F3h

Befehl	Zu beantworten mit	Funktion
E6h	FAh	Skalierung auf 1:1 setzen
E7h	FAh	Skalierung auf 2:1 setzen
E8h	FAh, (a)	Auflösung setzen
E9h	FAh, (A), (B), (C)	Status abfragen
EAh	FAh	Streammodus wählen
EBh	FAh, (A), (B), (C)	Ein Bewegungsdatenbyte lesen
ECh	FAh	Echomodus ausschalten
EEh	FAh, n*[(a), (A)]	Echomodus einschalten
F0h	FAh	Remote-Modus aktivieren
F2h	FAh, 00h	ID der Maus abfragen
F3h	FAh, (a), FAh	Sample Rate setzen
F4h	FAh	Streammodus einschalten
F5h	FAh	Streammodus ausschalten
F6h	FAh	Standardeinstellungen laden
FEh	(a)	Letztes Byte nochmals senden
FFh	FAh, AAh, 00h	Maus resetten
Andere	FEh oder FCh	Daten nochmals senden lassen (FEh), falls wieder Fehler -> FCh (Error)

Tabelle 3.4: Host-Maus Befehle

Die ID der Maus hat noch eine Besonderheit. Normalerweise ist sie 00h, falls aber der Rechner in einer bestimmten Reihenfolge die Sample Rate der Maus verändert (F3h, C8h, F3h, 64h, F3h, 50h) und anschließend die ID nochmals abfragt, meldet sich die Maus gegebenenfalls mit 03h anstatt mit 00h. Dies ist dann der Fall, wenn es eine Microsoft IntelliMouse ist. Daraufhin schalten sowohl der Host als auch die Maus in einen Modus, bei dem das Bewegungsdatenpaket aus vier anstatt drei Bytes aufgebaut ist. Das vierte zusätzliche Byte enthält dann Informationen über den Zustand des Rades.
Im Echomodus sendet die Maus alle empfangenen Daten unverändert wieder zurück. Dieser Modus kann nur durch das Senden von FFh oder ECh aufgehoben werden.
Die Standardeinstellungen einer Maus, welche mit F6h wieder hergestellt werden können, sind bei einer Gensys GL310MC2D Maus folgende:

samplerate	100 samples/s
resolution	4 counts/mm
scaling	1:1
streammode	selected
data reporting	disabled

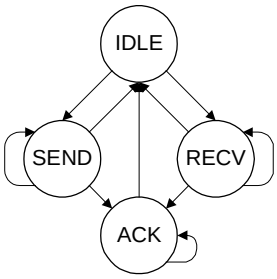
Zu beachten ist noch folgendes: wenn die Maus mit dem Befehl E7h (Set Scaling 2:1) umgestellt wird, werden die Bewegungen in der Maus anders umgerechnet. Folgende Tabelle gibt den Algorithmus wieder:

Bewegungszähler	Übermittelter Wert
0	0
1	1
2	1
3	3
4	6
5	9
>5	2*n
0	0

Der Algorithmus wurde nicht implementiert, da es sich bei den meisten Betriebssystemen um solche handelt, welche die Beschleunigung selbst verwalten. Es ist sinnvoller, die interne Beschleunigung der Betriebssysteme zu entschlüsseln und dieser dann beim Java-Client entgegenzuwirken, um eine möglichst gute Übereinstimmung zwischen lokalem und entfernten Mauszeiger zu gewährleisten. Beim Java Client wurde darauf geachtet, dass der maximale Wert 50 nicht überschreitet. Damit werden die Beschleuniger vermutlich weitestgehend ausgeschaltet. Im Zweifelsfalle kann dieser Wert jedoch noch weiter heruntergesetzt werden.

3.2 VHDL Design

Das VHDL-Design der PS/2-Statemachine besteht aus mehreren Teilen. Eingangsseitig werden Filter eingeschleift, um die PS/2 Signale zu filtern. Die Filterung wird jeweils über ein Schieberegister realisiert, welches eine Tiefe von 8-Bit hat. Erst wenn alle 8 Bits den gleichen Wert haben, wird der Wert durchgeschaltet. Hierdurch wird eine recht gute Filterung der Signale gewährleistet und Prellen oder Übersprechen der Leitungen eliminiert. Sollte der Filter zu schnell arbeiten, kann die Taktrate erniedrigt werden, so dass das Schieberegister langsamer läuft und die Filterung besser wird. Ein weiterer Teil dient der Generierung eines Timeout-Signals. Durch die Invertierung des Eingangssignals “restart” kann der Timeout-Zähler erneut gestartet werden. Eine solche Vorgehensweise ist zwar störanfälliger, jedoch in diesem Zusammenhang die geeignetere, da die Statemachine vereinfacht wird und dadurch schneller läuft. Sobald der Zähler einen bestimmten Wert überschritten hat, bleibt er stehen und zeigt über das Signal “timeout”, dass eine Zeitüberschreitung aufgetreten ist.
Das eigentliche Hauptmodul besteht aus zwei Statemachinen. Die eine – Main – bearbeitet den Hauptzustand und besteht aus vier Zuständen: der “Idle”-Zustand wartet solange, bis Daten gesendet oder empfangen werden. Falls Daten empfangen werden, wird in den Empfangsmodus “Receive” gewechselt, falls Daten gesendet werden sollen, in den Modus “Send”. Diese Übergänge haben die oben bei der Beschreibung des PS/2-Protokolls erwähnten Zwangsbedingungen. Sobald die Übertragung komplett ist oder ein Fehler (Timeout oder Host bricht die Übertragung ab) aufgetreten ist, wird in den normalen “Acknownledge”-Zustand gewechselt, der einen IRQ auslöst und somit das µCsimm über das Ende der Übertragung verständigt.



Eine weitere Statemachine – Sub – bearbeitet den entsprechenden Zustand bei der Übertragung (Senden/Empfangen). Bei dieser sind Zustände wie “Start” (Start-Bit), “Bits”, “Parity”, “Stop”, “ADD” (Bestätigungsmeldung des Geräts) und “ACK” (Übertragung ist abgeschlossen) zu finden. Diese repräsentieren die einzelnen Bits der Übertragung und steuern die Zustände von Daten- und Taktleitung. Der Takt kommt aus einem weiter unten erwähnten Taktgenerator und wird je nach Zustand durchgereicht oder nicht. Der Zustand “Bits” bleibt solange aktiv, bis alle 8 Bits komplett übertragen wurden. Je nach Zustandskombination zwischen beiden Statemaschinen werden dann die Signale für „gesendet“ oder „empfangen“ gesetzt. Diese wiederum bringen dann die Main-Statemaschine in den ACK-Zustand.
Zusätzlich zu diesem Design gibt es noch einen Taktgenerator, der den zur Übertragung nötigen Takt erzeugt. Da dieser sowohl für Maus als auch für die Tastatur unverändert benötigt wird, wurde er nicht als Bestandteil des PS/2-Moduls realisiert, sondern außerhalb dessen als eigenständiges Modul.

Die weitere Bearbeitung der übertragenen Bytes liegt in der Hand des µCsimm's. Die Schnittstelle des Moduls ist wie in Tabelle „PS/2-VHDL-Schnittstelle" aufgebaut. Die einzelnen Leitungen zum Steuern des Moduls wurden absichtlich zu einem 8-Bit-Bus zusammengefasst, da es so einfacher ist, alle Module zusammenzubinden. Es gibt dann einen Standard, bei dem alle Module ein komplettes Byte mit sich bringen und nur in den Registerraum an der entsprechenden Stelle eingeblendet werden müssen; andernfalls wären viel mehr Port-Zuweisungen notwendig und es gäbe keine so übersichtliche Struktur der einzelnen Register. Des weiteren ist es für das Schnittstellenmodul zum µCsimm nicht von Bedeutung, welche Signale weitergereicht werden sollen, sondern nur für das im µCsimm laufende Programm. Wichtig hingegen ist, dass die Bits, welche ausgegeben werden sollen, zwischengespeichert werden müssen. Eine 8-Bit breite Standardschnittstelle hat keinen Nachteil auf den Platzbedarf in der FPGA, da die Synthesetools die nicht benutzten Leitungen wegoptimieren.

3.3 Software

Der Softwareteil besteht aus dem Maus- und Tastaturmodul im µCsimm und arbeitet eng mit dem TCP-Modul zusammen. Innerhalb dieser Module findet auch die Verwaltung des VHDL-Moduls statt. Es wird für die korrekte Behandlung des Protokolls auf Byteebene gesorgt. Diese Teile sind ebenfalls für die Erkennung der beiden Geräte bei einem Start des Hosts verantwortlich.

Die Module laufen in einem Interruptmodus ähnlichen Zustand, das heißt sie werden immer nur dann aktiv, wenn die externe IRQ-Leitung auf 0 gezogen wird. Eine 0 löst einen Interrupt aus, der dann als Signal an das Hauptprogramm weitergeleitet wird. Dasselbe kann den Rest der Zeit ruhen und beeinträchtigt somit die Leistungsfähigkeit des µCsimm's nicht. Das Problem, wie nun eine Übertragung gestartet werden kann, wurde durch folgenden Kniff gelöst: das zu übertragende Byte wird an eine bestimmte Stelle geschrieben, ein Bit wird gesetzt und ein Signal wird an den eigenen Prozess gesendet, was die gleiche Auswirkung hat wie ein externer Interrupt.

In der Routine wird nun zuerst geprüft, ob ein neues Zeichen in die Warteschlange für die Ausgabe gestellt werden soll. Ist dies geschehen, wird getestet, ob auf eine zuvor vom Host gestellte Anforderung reagiert werden soll. Dies ist zum Beispiel dann der Fall, wenn der Host die LED's der Tastatur setzen möchte. Hierbei sendet der Host als erstes das Byte EDh, das gleich mit einem FAh beantwortet wird, unabhängig vom nächsten Zeichen im Tastenpuffer. Das nächste vom Host empfange Zeichen ist nun der Status, er darf nicht als Kommando interpretiert, sondern muss entsprechend bearbeitet werden.

Falls zuvor keine Anforderung kam, wird – falls ein Byte empfangen wurde – in einer switch-case Anweisung überprüft, welche Anforderung gestellt wurde und demgemäß reagiert. Eine Ausnahme gibt es noch: falls ein FEh empfangen wurde (Resend), muss das letzte Byte ein weiteres mal übertragen werden.

Die Übertragung, speziell die Reihenfolge der zu sendenden Bytes, wird folgendermaßen realisiert: es gibt einen Puffer – 8 Byte groß – welcher die auszugebenden Daten enthält. Soll nun ein weiteres Zeichen ausgegeben werden, so wird das Zeichen am Ende angehängt. Sollte hingegen auf eine Anfrage geantwortet werden, so wird das Zeichen am Anfang eingefügt. Falls mehrere Zeichen zurückgegeben werden sollen, so sind diese dann in der richtigen Reihenfolge am Anfang einzufügen.

Die eigentliche Steuerung des VHDL-Moduls findet am Anfang und am Ende statt. Am Anfang werden empfangene Bytes gelesen und an die obige Auswerteroutine weitergegeben. Sobald die Daten gelesen wurden, wird die Statemachine über die Floppy-Reset-

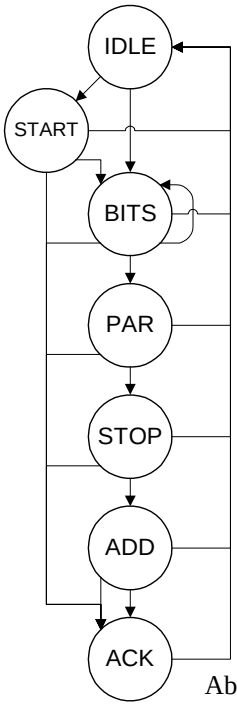


Abbildung 3.5: Sub-Statemachine

ps2_data, ps2_clock	Die Leitungen zum PS/2-Bus
to_host	Daten, die an den Host gesendet werden sollen
from_host	Daten, die vom Host empfangen wurden
config_in.0	1 = Daten sollen gesendet werden
config_in.2	1 = Main-Statemachine soll in den IDLE Zustand gehen
config_out.0	1 = Daten wurden erfolgreich gesendet
config_out.1	1 = Daten wurden erfolgreich emfangen und stehen zum Abholen bereit
config_out.2	1 = Main-Statemachine ist im IDLE Zustand
config_out.3	1 = Timeout ist aufgetreten
config_out.4	Paritybit beim Empfang
config_out.5	1 = Abbruch durch Host, Clock-Leitung ist nicht wieder auf 1 gegangen
config_out.6	PS/2-Clock-Leitung
config_out.7	PS/2-Data-Leitung
irq	Interrupt aufgetreten
I33MHZCLK	Takt vom PCI-Bus, Haupttakt
RESET	Alles zurücksetzen
ITimeoutCLK	Taktquelle für den Timeoutprozess
IPS2CLK	Takt für die PS/2-Clock Leitung

Tabelle 3.5: PS/2-VHDL-Schnittstelle

Leitung wieder in den Ausgangszustand gebracht. Dadurch ist sie wieder bereit, neue Daten zu senden/empfangen. Am Ende findet eine Überprüfung statt, ob noch ein Zeichen gesendet werden soll: falls ja, wird dies über die Send-Leitung initiiert.
Ein Setzen der Send-Leitung bewirkt, dass das Byte sobald als möglich an den Host übertragen wird.

Die so gestartete Übertragung kann natürlich auch vom Host unterbrochen werden, so dass später nochmals versucht werden muss, die Daten zu senden. Der Empfang der Daten wird durch ein gesetztes req-Bit signalisiert.
Ein weiterer Teil ist der Janus-Client, welcher die Zuordnung der Tasten zu den Scancodes vornimmt. Er wird im Abschnitt Janus-Client beschrieben.

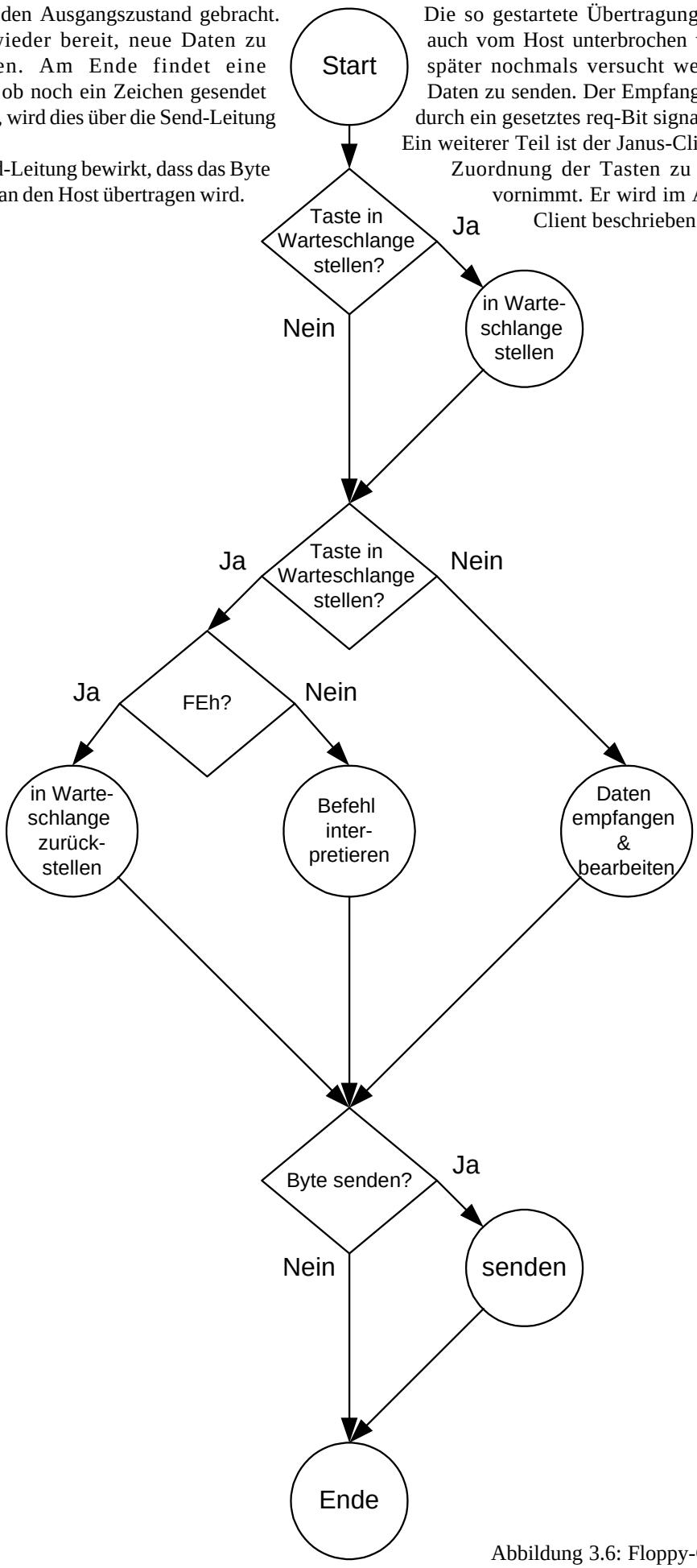


Abbildung 3.6: Floppy-C-Programm

4 Grafikinterface

Das Grafikinterface zu realisieren war eigentlich das Thema der Diplomarbeit von Stefan Philipp. Seine ersten Versuche führte er mit einer Orca-PCI-Karte durch; es konnte jedoch nicht funktionieren, da die Karte kein Expansions-ROM zur Verfügung stellt. Nachdem das Problem erkannt war, wurde eine Altera-PCI-Karte bestellt, welche die Option unterstützte. Bis die Karte jedoch geliefert wurde, verging ein halbes Jahr, weswegen sich das Grafikkarten-Thema so lange hinausgezögert hat. Da seine Praxiszeit fast abgelaufen war und immer noch kein VGA-BIOS existierte, wurde im Rahmen dieser Diplomarbeit versucht, das Problem in den Griff zu bekommen.
Nachdem ein BIOS gefunden war, wurde versucht, es beim PCI-Core einzubinden und einen Rechner mit der so entstandenen Grafikkarte zu testen. Auf Anhieb bootete der Rechner ohne Fehlermeldungen. Das ROM wurde auch in verschiedenen Grafikkarten getestet – alle liefen damit anstandslos. Es trat aber ein weiteres Problem auf: Grafikkarten tragen sich in den Speicher von A0000 bis BFFFF unter 1MB ein. Aus Zeitgründen wurde hier abgebrochen.

4.1 Hardware

Um den PCI-Bus von einer Karte aus anzusprechen, wird ein sogenannter PCI-Core, der von der Firma PLDApplications bezogen wurde und entsprechend konfiguriert werden musste, benötigt. Dieser Teil wurde von Stefan Philipp übernommen, der sich mit dem Thema PCI bereits längere Zeit beschäftigt hatte. Die folgenden Aufgaben galt es hierbei zu lösen: dem PCI-Core musste mitgeteilt werden, dass es sich bei der Karte um eine Grafikkarte handelt und dass ein Expansions-ROM vorhanden ist, welches in den Hauptspeicher einzubinden ist. Außerdem war das an die FPGA angeschlossene SRAM so einzubinden, dass ein Zugriff durch den PCI-Core, das µCsimm und das Floppy-Modul stattfinden konnte.
Die Konfiguration des Cores und des VHDL-Designs ist noch nicht komplett, aber soweit fortgeschritten, dass der Rechner bootet und ein Mitschnitt der BIOS-Funktionen, welche die Grafikausgabe standardmäßig realisieren, möglich ist. Da der Grafikspeicher noch nicht eingeblendet wird, ist die Aufgabe, eine Grafikkarte zu realisieren, noch nicht vollständig gelöst. Es scheint keine unlösbare Aufgabe zu sein, jedoch wird es nicht einfach sein, alle Register des Cores mit den richtigen Werten zu versehen. Problematisch wird sicherlich die Abhängigkeit der einzelnen Werte voneinander. Es existiert eine Option, die es ermöglicht, Speicher unter 1MB einblenden zu lassen, was eigentlich dem PCI-Standard widerspricht, aber bei Grafikkarten unabdingbar ist. Daher wurde die Option als Sonderfall in den offiziellen PCI-Standard mitaufgenommen. Wie dieser Speicher genau eingebunden wird, konnte nicht mehr herausgefunden werden. Dass er eingebunden wird, wurde noch nachgewiesen. Während der Tests mit Originalgrafik-karten, bei denen das BIOS durch das weiter unten beschriebene ROM ersetzt wurde, hat sich folgendes ergeben: das RAM der Grafikkarte wird in dieser Konstellation nicht mehr in den Bereich A0000h-BFFFFh eingebunden. Daraus lässt sich schließen, dass dies das Expansions-ROM übernimmt; das „wie“ konnte nicht mehr definitiv herausgefunden werden. Höchst-wahrscheinlich werden einige Register während des Boot-Vorgangs vom ROM initialisiert. Eigentlich muss sich eine Grafikkarte auch so verhalten, weil es möglich ist, mehr als eine Karte in einem Rechner zu betreiben, ohne dass Kollisionen auftreten.
Anscheinend schneiden die Grafikkarten die Speichersequenzen auf dem PCI-Bus mit und reagieren gegebenenfalls darauf. Darüberhinaus deutet auch die PCI-Bridge darauf hin, da dort kein Speichermapping für den Speicherbereich eingetragen war.
Nähere Details sind in der Arbeit von Stefan Philipp nachzulesen.

4.2 Boot-ROM

Ein entscheidender Teil der Grafikkarte ist das sogenannte VGA-BIOS (Boot-ROM der Grafikkarte). Dies wird als Expansions-ROM in den normalen Speicher beim PCI-Scan eingebunden und anschließend ausgeführt. In untenstehender Tabelle wird der Aufbau des Boot-ROM’s wiedergegeben:

Offset	Länge	Wert	Beschreibung
00h	1	55h	ROM-Signatur Byte 1
01h	1	AAh	ROM-Signatur Byte 2
02h	1		Größe des ROM's in 512 Byte-Blöcken
03h-05h	2		Einstiegspunkt für ROM-Programm-Code. Diese Adresse wird vom BIOS aufgerufen und ist normalerweise ein Sprung auf den eigentlichen Code.
06h-17h	12h		Reserviert. Zum Beispiel Copyright Text oder ähnliches. Kann frei verwendet werden. Auch Code darf hier noch stehen.
18h-19h	2		Zeiger auf PCI-Datenstruktur. Er ist relativ zum Beginn des ROM's.

Tabelle 4.1: Boot-ROM

Die PCI-Datenstruktur im Boot-ROM sieht folgendermaßen aus:

Offset	Größe	Wert	Beschreibung
00h	4	"PCIR"	Signatur des PCI-ROM's
04h	2		VendorID: eine ID, welche vom Hersteller der Karte vergeben wird. Es muss die gleiche wie im PCI-Funktionsregister sein. Für eine genauere Beschreibung in der PCI-Spezifikation nachlesen.
06h	2		DeviceID: sie ist eine ID, welche für das Gerät spezifisch ist. Siehe auch VendorID.
08h	2		Dieses Wort existiert erst ab PCI V2.2 und ist hier irrelevant.
0Ah	2		Länge der PCI-Struktur in Bytes. Little Endian!
0Ch	1		Revision der PCI-Datenstruktur. Bleibt auf 0.
0Dh	3		ClassCode, siehe Text
10h	2		Länge des Expansions-ROM's in 512 Byte-Schritten
12h	2		Revision der Daten
14h	1		Art des ausführbaren Codes; siehe unten
15h	1		Falls Bit 7 gesetzt ist, ist dies das letzte ROM
16h	2		Reserviert

Tabelle 4.2: PCI-Datenstruktur

Das verwendete BIOS hat folgenden Aufbau:

Abbildung 4.1: BIOS-Aufbau

Während des PCI-Bus Scans werden die einzelnen Expansions-ROM's in den Hauptspeicher eingebunden. Hierbei überprüft der Rechner das ROM auf korrekten Inhalt. Die Überprüfung beinhaltet die folgenden Schritte: Zuerst wird der Rahmen des ROM's geprüft. Dieser stammt noch aus alten ISA-Zeiten und wird in der Tabelle „Boot-ROM" in den ersten drei Zeilen verdeutlicht. Hinzu kommt noch ein weiteres Byte, welches in der Regel ganz am Ende des ROM's zu finden ist. Dieses Byte ist die Checksumme des Inhalts der Daten. Der Algorithmus, nach dem die Korrektheit des Inhalts abgeprüft wird, berechnet lediglich die Summe aller Bytes. Diese Summe muss am Ende 00h ergeben. So wird auch klar, warum das Byte nicht am Ende stehen muss.
Anschließend wird noch der PCI-Teil ausgewertet. Es ist eine Struktur (siehe Tabelle „PCI-Datenstruktur"), welche genauere Informationen für den PCI-Core beinhaltet. Da die Struktur so eingebettet wurde, ist es möglich, ein ROM, welches für PCI geschrieben wurde, auch in einer ISA-Erweiterung zu betreiben, ohne es anpassen zu müssen.
Wenn die Überprüfungen erfolgreich waren, wird das ROM an Offset 0003h angesprungen – siehe auch Tabelle "Boot-ROM". Da der PCI-Standard nicht für ein bestimmtes Rechnersystem geschrieben wurde, sondern ein offener Standard ist, muss noch festgelegt werden, um welchen Typ Code es sich handelt. Bisher wurden nur drei verschieden Werte vergeben:

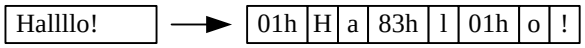
- 00h Intel x86 Code. Dies gilt normalerweise für PC's und wurde daher auch benutzt. Die anderen sind vielleicht sinnvoll; ob jedoch ein normaler PC den Code 01h akzeptieren würde, wurde nicht überprüft, da keine derartige Karte vorlag.
- 01h OpenBoot Code. Dieser Code ist ein FCode. Er steht für "tokenized Forth Code" und beinhaltet einen prozessorunabhängigen Code. Genaueres ist im IEEE Standard 1275-1994 (Standard for Boot (Initialization, Configuration) Firmware Core Requirements) nachzulesen.
- 02h HP PA/RISC. Dies wurde erst ab PCI 2.2 eingeführt und gilt für HP-Prozessoren.

Falls es möglich ist; Grafikkarten mit Code 01h in PC's zu betreiben, wäre dies wohl die beste Art, eine system-übergreifend kompatible Grafikkarte (CIA-Karte) zu realisieren. Im folgenden wird speziell auf den PC eingegangen.
Das VGA-BIOS hat die Aufgabe, zum einen die komplette Grafikkarte zu initialisieren und zum anderen sich in die Int10h Interruptroutine des BIOS zu hängen. Zu Zeiten von EGA/MGA war noch das PC-BIOS selbst dafür verantwortlich, wie die einzelnen Zeichen auf dem Monitor dargestellt werden. Damals war es auch noch sehr teuer, ein ROM auf einer Karte zu betreiben. Seit VGA werden sämtliche Grafikroutinen auf die Grafikkarte umgelenkt. Daher ist es auch für die Hersteller einfacher geworden, ihre Karte anzupassen. Das ist wohl auch der Grund, warum VGA als Standard für Grafikkarten definiert wurde. Außerdem werden seit VGA einige Register im unteren Speicherbereich nicht mehr benutzt. Um zu testen, ob auch wirklich eine Grafikkarte in dem Rechner vorhanden ist, ruft das BIOS nach dem PCI-Bus Scan einfach die Int10h Funktion mit dem Kommando für das Abfragen des Videomodus auf. Kehrt diese Funktion nicht mit den richtigen Parametern zurück, akzeptiert das Bios diese Karte nicht als Grafikkarte und erzeugt eine akustische Fehlermeldung („Beep").
Sämtliche Routinen werden von dem benutzten Bios gepatched und an entsprechende C-Funktionen weitergeleitet. An dieser Stelle wurde nun testweise eingegriffen. Die normalen Routinen, um etwas auf dem Bildschirm auszugeben, wurden so erweitert, dass die Bildschirmausgabe auch parallel dazu auf der seriellen Schnittstelle erfolgt. Die Ausgabe wurde mit einem anderen PC mitgeschnitten. Nun ist es bereits möglich, einem Rechner beim Booten zuzusehen, ohne dass ein Monitor vorhanden ist. Leider ist die Ausgabe oder zumindest der Boot-Prozess teilweise zeitkritisch, so dass das BIOS des PC's einen Timeout-Error meldet, sobald eine Parallelausgabe der Bildschirmdaten über die serielle Schnittstelle erfolgt. Leider sind die Debugmöglichkeiten sehr schlecht. Die einzigen Auswege sind Schreiben der Daten auf die serielle Schnittstelle oder einen vorher dazu definierten Port, der dann jedoch abgescannt werden muss.
Später kann diese Routine dazu verwendet werden, die Daten direkt zum µCsimm zu senden und dann weiter zum Java-Client, der daraufhin parallel zur normalen Bildschirmausgabe eine Remoteausgabe realisiert. Ebenfalls ist eine Erkennung von Fehlern etcetera möglich, da der Text nicht irgendwo im Bildschirm-speicher gesucht werden muss, sondern in der richtigen Reihenfolge ausgegeben wird und nur dieser Text durchsucht werden muss. So kann zum Beispiel auf das Wort "Error" sehr einfach getriggert werden.
Eine genaue Beschreibung der einzelnen Grafikkartenbefehle kann dem Quelltext, dem Anhang "VGA Konfigurationsregister und Interrupts" oder der Literatur [T2] und [P2] entnommen werden.
Um ein BIOS zu bekommen, wurde zuerst versucht, das einer Grafikkarte zu disassemblieren, was jedoch aufgrund der Komplexität nicht einfach war und deshalb auch aufgegeben wurde. Im Internet wurden zwei weitere

Versionen gefunden: die eine war für ein Seiko/Epson LCD-Modul und komplett in Assembler geschrieben, daher sehr lang und schwer zu pflegen und zu verstehen. Die andere stammt vom x86-Emulator Plex86, der ein PC-Emulator für Unix ist. Da der Emulator auch eine Grafikkarte braucht, ist bei dem Projekt ebenfalls ein Grafikkarten-BIOS vorhanden. Das BIOS ist fast komplett in C geschrieben, bis auf ein paar Stellen, an denen Inline-Assembler benutzt wurde. Das hat den großen Vorteil der besseren Verständlichkeit und Pflegbarkeit. Leider gibt es auch einen kleinen Nachteil: nicht alle Befehle sind vollkommen auscodiert. Die für die Textausgabe wichtigsten hingegen sind vorhanden. Mit Hilfe der Informationen aus den anderen Versionen wird es aber möglich sein, die fehlenden Teile zu rekonstruieren, zumal diese Teile nicht von großer Bedeutung sind (zum Beispiel Lightpen). Es ist sinnvoll, noch etwas zu warten, da davon ausgegangen werden kann, dass die fehlenden Routinen bei einer neuen Version sicherlich vorhanden sein werden. Für die reine Emulation wird die verwendete Version jedoch ausreichen.

4.3 Software

Die Software für das Grafikinterface ist als Modul für den TCP-Server ausgelegt. Bisher wurde es noch nicht komplett fertig gestellt, da die eigentliche Hardwareanbindung (PCI-Core und Expansions-ROM) noch nicht funktioniert. Die Fertigstellung war bis zur Hälfte der Diplomarbeit geplant. Daher wurde zum Testen der eigentlichen Übertragung ein Scheinmodul geschrieben, welches die Grafikdaten nicht aus dem SRAM, welches später einmal als Grafikkartenspeicher dienen wird, holt, sondern aus drei verschiedenen festen Arrays. Durch diese Maßnahmen konnte zumindest die Übertragung und Performance getestet werden.
Auf die Daten des Textbildschirms kann über zwei verschiedene Methoden zugegriffen werden. Das hat man dem Grafik-BIOS zu verdanken. Da die meisten Programme, welche auf dem Host laufen werden, nicht direkt in den Grafikspeicher, sondern über die normalen BIOS-Funktionen in denselben schreiben (INT10h), ergibt sich eine geniale Möglichkeit: sämtliche VGA-Routinen können in das VGA-BIOS umgeleitet werden und der Quelltext ist vorhanden und kann verändert werden; somit eröffnet sich die Möglichkeit, die Ausgabe dieser Routinen über das Netzwerk an den Java-Client weiterzuleiten. Durch dieses Verfahren werden nur noch Deltas übertragen und ein Bildschirmmoduswechsel ist ebenfalls leichter erkennbar. Eine weitere Möglichkeit überträgt lediglich den ganzen Bildschirminhalt komplett. Die Methode ist dann sinnvoll, wenn Programme oder Betriebssysteme direkt in den Grafikkartenspeicher schreiben und nicht über die BIOS-Funktionen darauf zugreifen. Es ist aber auf jeden Fall die sicherere Möglichkeit. Da hierbei viele Daten übertragen werden müssen, wird der Bildschirminhalt gepackt übertragen, um die Netzwerkperformance zu verbessern. Zum Testen wurde eine Übertragungsverkürzung durch folgendes Verfahren erreicht:



Falls gleiche Zeichen aufeinander folgen, so wird deren Anzahl plus das Zeichen selbst übertragen. Falls ungleiche Zeichen folgen, werden die Anzahl unterschiedlicher Zeichen und die Zeichen selbst übertragen. Die maximale Anzahl der Zeichen liegt in beiden Fällen bei 128, die Kennzeichnung, ob gleiche oder unterschiedliche Bytes folgen, wird in Bit 7 angezeigt. Eine Performance-steigerung kann in Zukunft dadurch erreicht werden, dass der Packalgorithmus Bildschirmänderungen zur vorangegangenen Übertragung feststellt und nur diese überträgt. Oder aber es wird ein besserer Packalgorithmus benutzt. Da sich der Bildschirminhalt eine sehr lange Zeit nicht ändert, wird hier eine enorme Steigerung erwartet. Bisher wurde, wie schon erwähnt, nur der zweite Modus implementiert, da er auch ohne Grafikkarte getestet werden kann. Der erste Modus wurde nur angedacht, einer Verwirklichung sollte jedoch nichts im Wege stehen, weil das komplette BIOS im C-Quelltext vorliegt und somit eine einfachen Anpassung gewährleistet ist.
Ein Bildschirminhalt kann zur Zeit über den Befehl “b” angefordert werden. Das Modul beantwortet diese Anforderung mit der Anzahl der folgenden Daten im Format Low-Byte, High-Byte, gefolgt von den gepackten Bildschirmdaten. Eine Übertragung der Farbinformation ist zur Zeit noch nicht vorgesehen, kann jedoch einfach nachimplementiert werden. Eine Dekodierung der Daten erfolgt im Java-Client JANUS.

5 TCP-Server

Der CIA-Server besteht aus mehreren Teilen: dem eigentlichen Hauptprogramm “tcpserv”, einem Interrupttreiber und den Modulen für Maus, Tastatur, Grafik, ... Dieser Aufbau wurde für eine leichtere Möglichkeit der Wartung und Pflege der Software gewählt. Die Module sind eigenständig und haben keine gegenseitigen Referenzen, daher ist es einfach, neue Module wie zum Beispiel USB hinzuzufügen. Im folgenden werden die einzelnen Teile beschrieben. Die Installation der Software beziehungsweise der Programmteile sind im Anhang zu finden. In diesem Kapitel wird mehr auf die Funktionsweise der Module eingegangen, die eigentliche Funktion der Module wird in den entsprechenden Kapiteln abgehandelt.

5.1 Interrupthandler

Um eine möglichst geringe Latenzzeit auf Ereignisse durch die FPGA zu erreichen, war es nötig, einen Interrupt zu verwenden. Über die Funktion “request_irq” ist es unter Linux möglich, einen Interrupttreiber einzubinden. Da sie eine Kernelfunktion ist, die nur aus dem sogenannten Kernelspace heraus aufgerufen werden kann, ist es normalen Programmen nicht gestattet, die Funktionen zu benutzen, nur Treiber und der Kernel selbst können auf die Funktionen zugreifen. Es gibt zwei Möglichkeiten: das komplette Programm kann als Treiber ausgelegt werden, jedoch sind dann die Debugmöglichkeiten nicht besonders komfortabel, außerdem entspricht es keinem guten Programmierstil. Als zweite Möglichkeit ergibt sich eine Auslagerung des eigentlichen Treibers aus dem Hauptprogramm. In diesem Falle wird ein Device benötigt, das den Interrupt-Handler einbindet. Die Idee ist es, einen externen Interrupt in ein Signal an das Hauptprogramm umzuwandeln. Hierdurch ergibt sich eine Entkopplung der beiden Teile, was eine spätere Portierung einfacher macht und die Abstraktionsebene des Programmierens stark anhebt. Aus diesem Grund fiel die Entscheidung schnell auf ein “Char-Device”. Es kann auf der einen Seite die Kernelfunktionen ausführen, auf der anderen Seite stellt es sich wie ein normales Device dar, welches über die Standardfunktionen bedient werden kann.

Über den Befehl “insmod” kann dem Betriebssystem ein Treiber zur Laufzeit hinzugefügt werden. µClinux besitzt diesen Befehl leider nicht, daher muss der Treiber direkt mit in den Kernel einkompiliert werden. Das hatte zur Folge, dass bei jeder kleinen Änderung am Treiber – dies war beim Testen sehr oft der Fall (circa 20 mal täglich) – der komplette Kernel neu kompiliert werden und in das Flash programmiert werden musste. Jeder Zyklus dauert circa 4 Minuten. Um den Treiber so kompatibel wie nur möglich zu gestalten, wurde er so geschrieben, dass er sowohl mit dem Befehl “insmod” während der Laufzeit geladen werden – für eine Linuxversion, die den Befehl “insmod” unterstützt – als auch direkt in den Kernel mit eingebunden werden kann, was hier nötig war. Falls µClinux später einmal den Befehl “insmod” besitzen sollte, kann der Treiber zu jedem Zeitpunkt geladen und wieder entladen werden, weswegen Änderungen an ihm wesentlich einfacher durchzuführen sind.

Nun zur Funktionsweise: Ein Device hat die Möglichkeit, einen Interruptvektor auf sich zeigen zu lassen. Das bedeutet: falls ein Interrupt auftritt, ruft Linux der Reihe

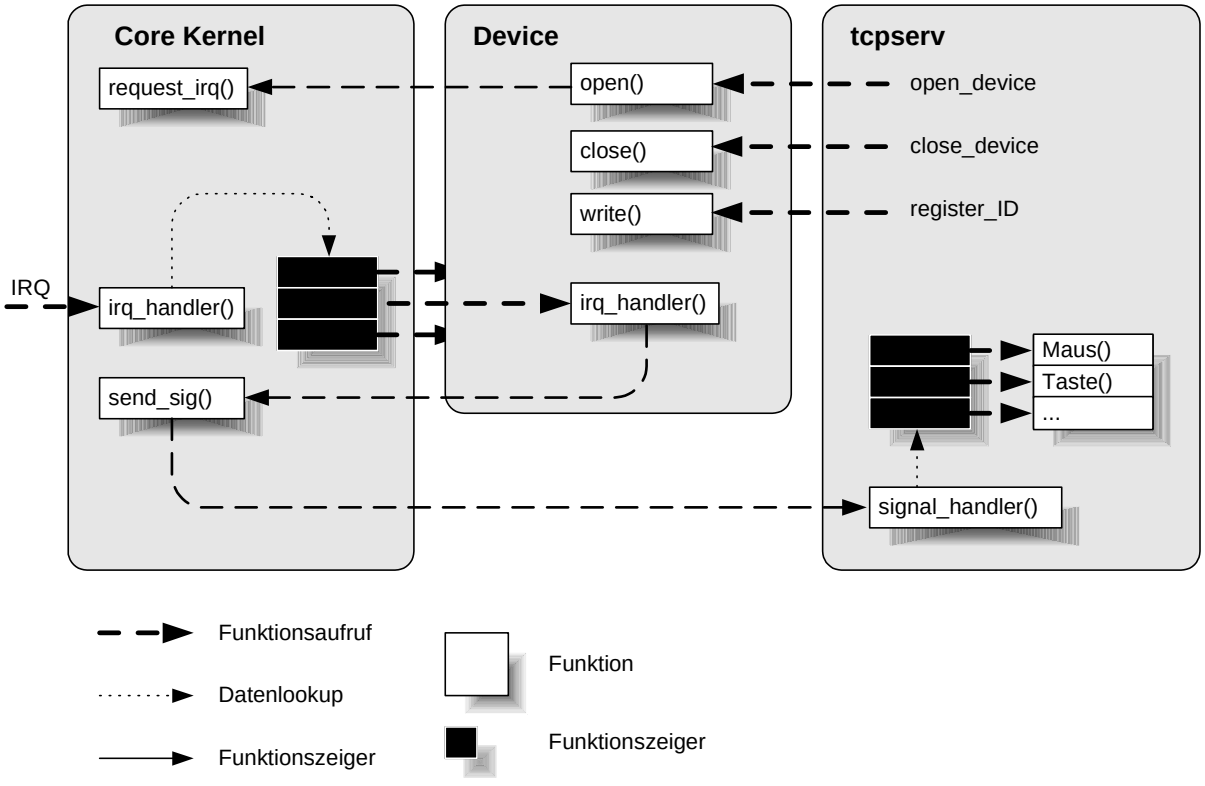


Abbildung 5.1: TCP-Signallauf

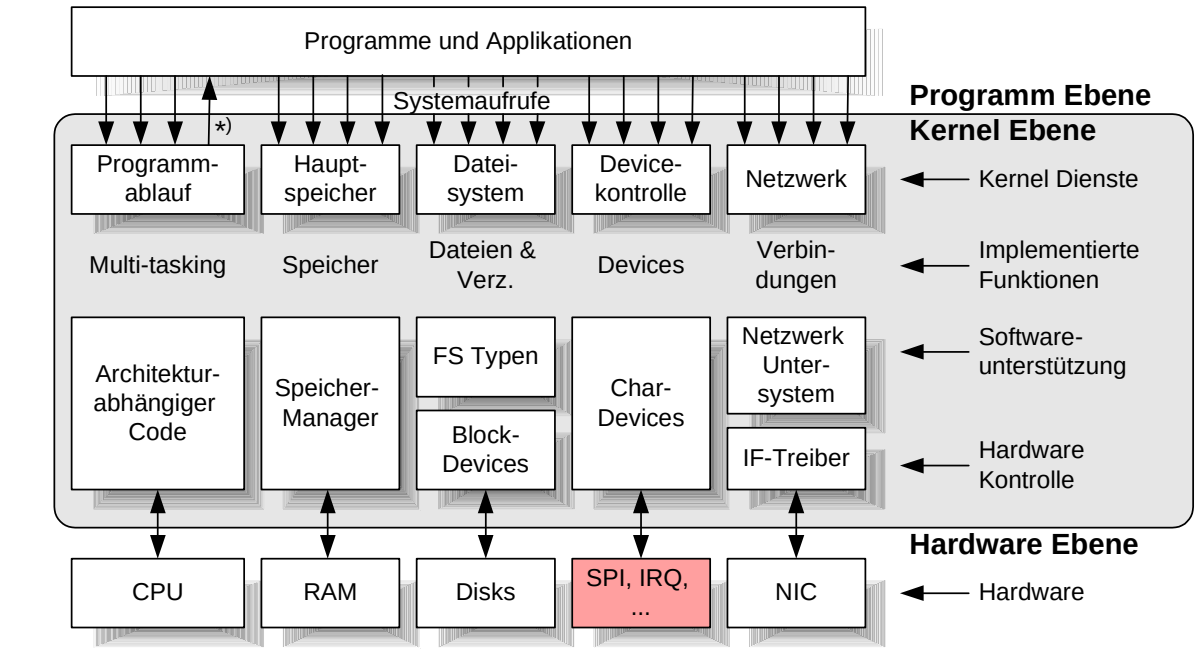


Abbildung 5.2: Unix-Kernel

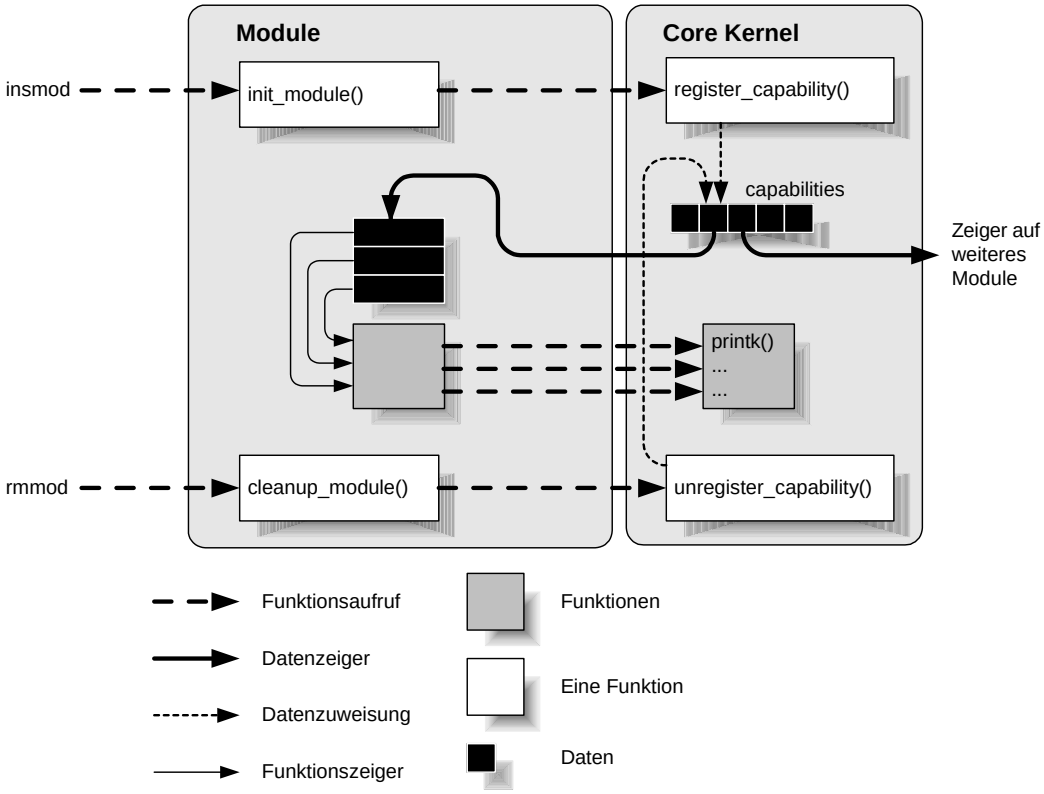


Abbildung 5.3: Unix-Signale

nach alle Programme auf, die sich bezüglich des aufgetretenen Interrupts registriert haben. Hierdurch ist es möglich, auf externe Ereignisse sehr schnell zu reagieren – es ist sogar die schnellste Möglichkeit. Als externe Quelle wurde im Falle des µCsimm's IRQ 3 gewählt, der zwar keine sehr hohe Priorität hat, aber in einem Test eine Rate von circa 1kHz erreichte, was einer Latenzzeit von 1ms entspricht und für die Anwendungen ausreichend ist, zumal die meiste Zeit durch die eigentliche Behandlung der Ereignisse beansprucht wird. Außerdem werden andere wichtige Aktionen von Linux hierdurch nicht beeinträchtigt. Der folgende Befehl fügt nun einen Interrupthandler in die Sprungtabelle von Linux ein:

```
rc=request_irq(IRQ_MACHSPEC|IRQ3_IRQ_NUM,
    fpga_interrupt, IRQ_FLG_STD, "FPGA-IRQ",
    NULL);
```

Als erstes Argument wird die Nummer des Interrupts übergeben, gefolgt von dem Zeiger auf die einzufügende Routine. Das vorletzte Argument gibt den Namen des Handlers an, um ihn später wieder identifizieren zu können. Die Möglichkeit, einen benutzerdefinierten Zeiger zu verwenden – das letzte Argument – wurde nicht genutzt, da µClinux keinen Speicherschutz besitzt und im gegebenen Falle auch keiner benötigt wird, weil nur ein Signal gesendet werden muss. In nachfolgender Abbildung ist der Ablauf des Programms zu sehen:

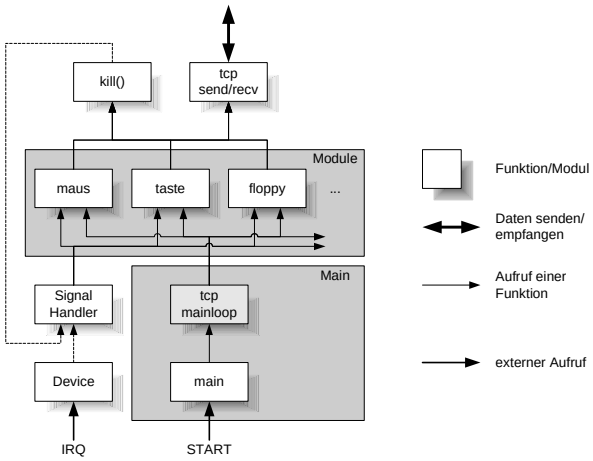


Abbildung 5.4: TCP-Server-Module

Beim Initialisieren des Device werden alle Funktionen beim System registriert. Um die Prozess-ID an den Devicetreiber zu übergeben, wurde die Funktion “write” verwendet. Beim Schreiben auf das Device kann ihm die Prozess-ID übergeben werden: dazu muss nacheinander ‚p‘, (ID-Low), (ID-High) geschrieben werden. Danach kennt der Devicetreiber die ID und kann dorthin ein Signal senden. Die ganze Vorgehensweise ist nicht die eleganteste, sie funktioniert jedoch stabil. Sobald ein Interrupt auftritt, wird die Interrupt-Funktion des Devices aufgerufen, die dann an die dem Device vorher übergebene Prozess-ID ein Signal sendet. Das Hauptprogramm hat nun die Möglichkeit, die Quelle des Interrupts zu lokalisieren und entsprechende Aktionen auszuführen. Die Ereignisbehandlung ist schließlich Sache der Module. Als Signal wurde “SIGUSR1” verwendet, das für solche Probleme gedacht ist.

Um die Aktionen des Treibers zu testen, gibt es nur eine Möglichkeit, Daten auszugeben – den Befehl “printk”: er hat die gleiche Syntax wie “printf”, arbeitet aber im “Kernelspace”. Er gibt die Daten auf der Standardkonsole aus, auf der auch alle anderen Ausgaben angezeigt werden, welche der Kernel oder die Module erzeugen. Debugdaten auszugeben hat den Vorteil, dass alles im Klartext zu lesen und nachzuverfolgen ist. Dadurch wird der eigentliche Treiber allerdings verlangsamt. Die Latenzzeit erhöht sich teilweise so stark, dass einige Funktionen nicht mehr korrekt abgearbeitet werden können, und es treten nicht vorhersehbare Fehler auf.

Um zu testen, ob Interrupts ausgelöst werden, ist es möglich, im Verzeichnis “etc/proc/” nach einer Datei namens “interrupt” zu schauen. In dieser sind alle Interrupts aufgelistet und auch deren bisherige Anzahl von Ereignissen. Hieran kann gesehen werden, wie viele und ob überhaupt Interrupts ausgelöst wurden.

Der Interrupttreiber hat allerdings ein kleines Problem mit der Eingabe von Zeichen. Wenn die “Return” Taste gedrückt wird, wird ein “SIGIO” an den Hauptprozess gesendet und dieser terminiert. Es wurde versucht, für dieses Signal auch Signalhandler einzubinden, um es abzufangen, es kamen jedoch noch weitere Signale an. Lediglich das Signal “SIGPROF” kann noch abgefangen werden, alle anderen nicht mehr. Im Moment ist dies kein Problem, da keine Eingaben zum Programm erforderlich sind.

5.2 Hauptprogramm

Die Abbildung “TCP-Server-Module” zeigt den Informationsfluss zwischen den einzelnen Modulen. “START” entspricht dem Aufruf des Servers (Programmstart). Zu Beginn werden die einzelnen Module initialisiert, was jedoch aus Übersichtlichkeitsgründen nicht eingezeichnet ist. “tcp mainloop” ist die eigentliche Hauptschleife, aus der heraus die einzelnen Module aufgerufen werden. Der ganze Server ist so ausgelegt, dass alle Aktionen, die das Netzwerk betreffen, über IP-Pakete angefordert werden müssen (Polling). Asynchron dazu kann der Signalhandler aufgerufen werden, der dann das Hauptprogramm wie ein Interrupt unterbricht. Die Unterbrechung kann von zwei Seiten kommen: einmal durch die Module selbst und einmal durch einen externen Interrupt, der durch das Device in ein Signal umgewandelt wird. Dies ist nötig, da sonst keine Synchronisation zwischen den beiden Teilen aufgebaut werden könnte. Um zum Beispiel ein Zeichen

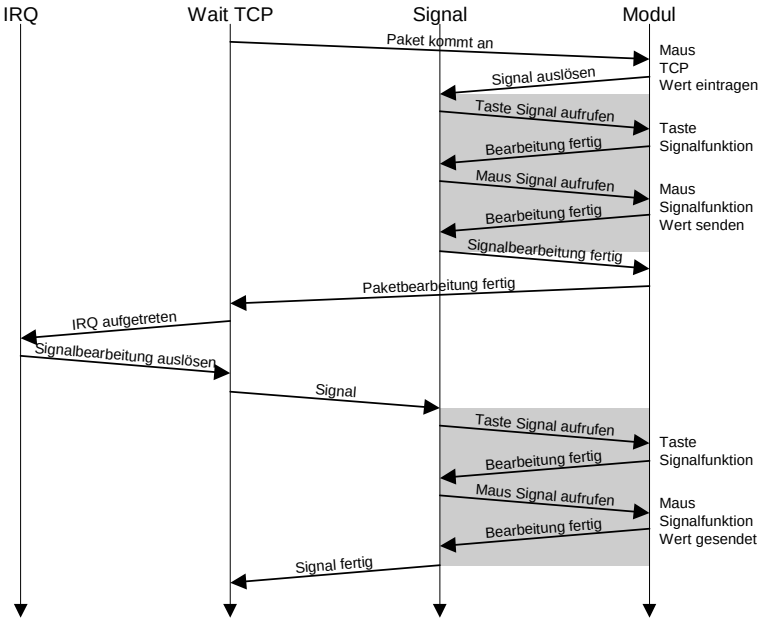


Abbildung 5.5: TCP-Server-Event-Trace

in die Ausgabewarteschlange der Tastatur zu setzen, wird das Zeichen an eine bestimmte Speicherstelle geschrieben, ein Bit gesetzt und dann ein Signal über den “kill”-Befehl an den eigenen Prozess gesendet. Der Signalhandler ruft alle Module immer wieder der Reihe nach auf, bis kein externer Interrupt mehr anliegt. Das Tastaturmodul erkennt, dass ein Zeichen ausgegeben werden muss, setzt es in die Warteschlange, startet gegebenenfalls eine PS/2-Übertragung und löscht das zuvor gesetzte Bit wieder.

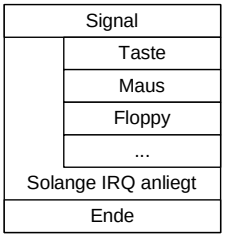


Abbildung 5.5: TCP-Server Signalablauf

5.3 Modulaufbau

In der Tabelle “TCP-Server Module“ sind alle bisher implementierten Module aufgelistet. Die ersten vier Module haben keine Funktionalität bezüglich der FPGA. Sie so zu realisieren, hat sich aber angeboten.

Init	Initialisiert sämtliche Funktionalitäten, wie Ports
Device	Öffnet das Device und übergibt diesem die eigenen Prozess-ID
TCP	Modul zum Verwalten der gesamten TCP-Funktionalität. Es ruft ggf. die anderen Module auf
UDP	Soll später den Broadcast übernehmen. Ist bisher nur angedacht.
Keyb.	Verwaltet die komplette Tastatur
Maus	Verwaltet die Maus
Floppy	Verwaltet das Diskettenlaufwerk
Video	Dient dem Packen und Auslesen eines Bildschirms. Das Auslesen ist noch nicht implementiert.

Tabelle 5.1: TCP-Server Module

Untereinander haben die Module nur wenige Referenzen, bei den ersten vier jedoch existieren viele. Beim Empfang von TCP-Daten werden sie dann gezielt an das entsprechende Modul weitergegeben.
Tabelle “TCP-Server Befehle“ zeigt alle bisher existierenden Befehle auf. Folgender Ausschnitt aus dem TCP-Modul zeigt die Funktionsweise:

```
switch(buffer[0])
{
    case ‘b’:  SendScreen(data);
               break;
    case ‘k’:  ProcessKey(data, 0);
               break;
    case ‘l’:  ProcessKey(data, 0x80);
               break;
    case ‘m’: ProcessMouse(data);
               break;
    case ‘q’:  printf(„Connection closed!\n“);
               return true;
    case ‘x’:  printf(„Closing TCP-Server for
restart!\n“);
               return false;
};
```

Die Funktionsweise der letzen vier Module wurde bereits in den entsprechenden Kapiteln erläutert. Weitere Module wie zum Beispiel ein USB-Modul können durch den modularen Aufbau sehr leicht nachträglich eingefügt werden. Wie dies zu realisieren ist, wird nun beschrieben.
Das Hauptprogramm des µCsimm’s stellt über eine Funktion die Möglichkeit zur Verfügung, Module für verschiedene Geräte einzubinden. Diese Module müssen lediglich hinzugelinkt werden, dafür sind sie im Makefile bei OBJS mitanzugeben. Ein Aufruf einer Modul-Init-Funktion muss im Hauptprogramm erfolgen.
Ein solcher Aufruf im Hauptprogramm wird in der Funktion “main_init_all” bewerkstelligt und sieht folgendermaßen aus :

```
if(!floppy_insert(gd)) return false;
```

Der Aufruf trägt zum Beispiel das Modul zur Floppyansteuerung ein. Das Modul sollte zu Beginn alle benötigten Daten reservieren und initialisieren. Anschließend muss die aufgerufene Funktion noch dafür

Befehl	Antwort	Funktion
'q'		Beendet den TCP-server, zu Debugzwecken sehr nützlich.
'b'	l, h, Bildschirmdaten	Fordert den aktuellen Grafikbildschirm an.
'k', a		Gibt einen Tastendruck über die PS/2-Schnittstelle aus. Der Wert a liegt zwischen 00h und 7Fh. bei ‚k' wird dieser Wert übertragen, bei , ‚l' wird noch 80h hinzuaddiert.
'l', a		
'm', a, b, c		Gibt eine Mausbewegung über die PS/2-Schnittstelle aus. Die Werte a, b und c liegen immer im Bereich 00h und 7Fh.
'x'		Beendet die Verbindung. Das µCsimm geht wieder in den Bereitschaftsmodus, um Verbindungen zu akzeptieren.

Tabelle 5.2: TCP-Server Befehle

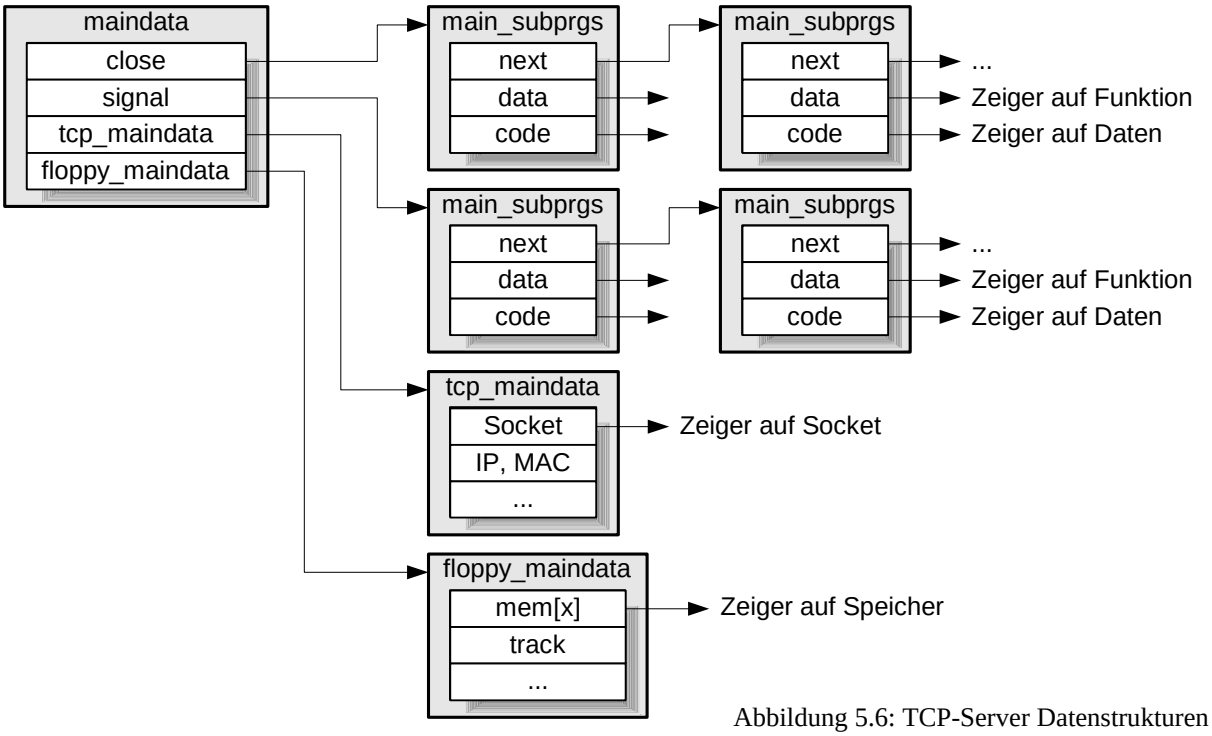


Abbildung 5.6: TCP-Server Datenstrukturen

sorgen, dass sie beim Hauptprogramm korrekt eingetragen wird:

```
bool main_register_subs(struct maindata *gd,
        bool (*code_close)(void *),
        (*code_signal)(void *), *data);
```

Die Parameter setzen sich wie folgt zusammen: der erste Zeiger ist eine Referenz auf die globale Struktur des TCP-Servers und wird zwecks Objektorientierung auf diese Art übergeben. Der zweite und dritte Parameter sind Zeiger auf Funktionen. Bleiben sie NULL, so wird die entsprechende Funktionalität nicht mit eingebunden. Die Close-Funktionsadresse verweist auf eine Funktion, die beim Beenden des TCP-Servers aufgerufen wird. Somit ist sichergestellt, dass der allozierte Speicher wieder korrekt freigeben wird und auch andere Einträge wieder rückgängig gemacht werden. Die Signal-Funktions-adresse wird aufgerufen, falls entweder ein externer Interrupt oder ein Signal auftreten. Sie muss dann prüfen, ob der Interrupt gegebenenfalls für das eigene Modul galt und in diesem Fall entsprechend handeln. Der letzte Parameter „Daten“ ist optional. Es ist ein void-Zeiger auf einen Speicherbereich. Dieser ist vonnöten, um globale Variablen soweit wie möglich zu vermeiden und um eine Abgrenzung der einzelnen Module voneinander gewährleisten zu können. Dadurch wird die Lesbarkeit des ganzen Programms wesentlich verbessert und Kollisionen mit Variablennamen werden umgangen.
Damit ein Modul in der Signal-Funktion auf den zuvor allozierten Speicher zugreifen kann, stellt das Haupt-programm eine Funktion dafür zur Verfügung:

```
void *main_getowndata(
        bool (*code_close)(void *));
```

Der ganze Mechanismus funktioniert jedoch nur, wenn das Modul eine Close-Funktion eingetragen hat.

5.4 Scriptsprache

Eine Scriptsprache für die später einmal geplanten Überwachungsmechanismen des µCsimm‘s wurde angedacht. Da zum damaligen Zeitpunkt wichtigere Untersuchungen im Vordergrund standen, wurde dieses Thema zurückgestellt. Damit jedoch die Überlegungen nicht einfach vergessen werden, soll hier kurz darauf eingegangen werden. Geplant ist die Scriptsprache, um gewisse Aktionen aufgrund von Konstellationen auszuführen. Ein Beispiel hierzu wäre: die Temperatur des Gehäuses übersteigt einen kritischen Wert ⇒ eine Nachricht wird versendet; sollte sie noch weiter ansteigen, wird der Rechner ausgeschaltet. Auf dem zu überwachenden Rechner, speziell im Cluster, wird eine Software installiert, die ebenfalls den eigenen Rechner überwacht; falls der Prozessor hängen bleiben sollte, existiert hierdurch eine weitere Sicherheit. Um eine Scriptsprache zu realisieren, gibt es mehrere Ansätze, wie zum Beispiel Perl, welches an sich schon viele Funktionalitäten mit sich bringt. Etwas in dieser Richtung zu schreiben, sollte nicht länger als einen Monat dauern. Vorgesehen waren eine Startdatei und mehrere Konfigurationsdateien:

Init.script:
: dieses Script wird beim Start automatisch ausgeführt und registriert alle Standardtrigger

<Programm>.script:
: ein Script, das durch einen Trigger oder von der Konsole aus gestartet werden kann

Geplant waren die Befehle in Tabelle „Scriptsprache“ auf der nächsten Seite.

Folgendes Beispielprogramm verdeutlicht die vorgesehene Funktionsweise bei einem Stromausfall:

Init.script:

```
On equals P0.1 1 Powerfail.script
        # Bei Stromausfall Powerfail.script ausführen
```

Powerfail.script:

```
    Clear P0.0                  # Strom des Rechners
                                  ausschalten
    Status “Powerfail: {IP}”  # Ereignis in globale
                                  Log-Datei schreiben
    End
```

Defin	Richtung eines Ports definieren
Defout	Richtung eines Ports definieren
Set <Port.Bit>	ein Bit setzen
Clear <Port.Bit>	ein Bit löschen
Cpl <Port.Bit>	ein Bit invertieren
Or <Port Byte>	ein Byte zu einem Port dazuODERn
And <Port Byte>	ein Byte zu einem Port dazuUNDen
Xor <Port Byte>	ein Byte zu einem Port dazuXODERn
Waitsec <zeit>	die Zeit <zeit> warten
Wait <until>	auf ein Ereignis warten
Goto	Sprung
If	Bedingung
Else	
Endif	
End	Programmende
<Label>:	Sprungmarke
Send <Nachricht>	eine Nachricht sofort an den Systemadministrator
	senden
Status <Nachricht>	eine Statusnachricht in eine globale Log-Datei
	eintragen
Execute <Programm>	ein Programm ausführen und auf dessen Ende warten
Fork <Programm>	ein Programm asynchron ausführen
Script <Programm>	ein anderes Sriptprogramm ausführen (Gosub/Call)
On (redge/fedge) <Port.Bit>	einen neuen Trigger in die dafür vorgesehene Datei eintragen, in diesem Falle wird auf ein bestimmtes Ereignis reagiert
On (greater/smaller) <Zähler>	
Set Counter <Variable> <Bedingung>	eine Variable wird automatisch bei der Bedingung hochgezählt. Als Bedingung kann „On (redge/fedge)...“ benutzt werden. Dies ist schneller, als es per Script zu erledigen.
{IP}	Konstante für IP-Adresse
{DATE}/{TIME}	Konstante für aktuelles Datum/Zeit
{MAC}	Konstante für MAC-Adresse
{MACTIME}	Konstante für eine Zeit, die durch die MAC-Adresse berechnet wird.
#	Kommentar

Tabelle 5.3: Scriptsprache

6 Janus-Client

Der Janus-Client ist der Softwareteil, welcher auf denjenigen Arbeitsstationen läuft, die einen Host steuern möchten. Es ist die zentrale Stelle, von der aus alles koordiniert werden kann.

6.1 Schnittstellen

Er besitzt eine TCP-Schnittstelle, welche zur CIA-Karte zwei Ports öffnen kann. Der eine Port (Port 77) dient der Kommunikation mit dem CIA-Server von Stefan Philipp, der andere (Port 5432) dient dem eigentlichen Steuern eines Rechners.
Sobald zum Beispiel der Button PowerOn geklickt wird, loggt sich der Janus-Client bei dem entsprechenden µCsimm auf Port 77 ein (sendet den Benutzernamen und das Passwort), übermittelt den gewünschten Befehl, hier "poweron", und beendet die Sitzung mit "quit" wieder. Die Methode hierfür ist "SendCmd" in "TCP_Client", welcher der zu sendende Befehl übergeben wird. Die bisher implementierten Befehle sind in Tabelle "Janus Befehle" aufgelistet. Eine detailliertere Beschreibung der Befehle ist in der Diplomarbeit von Stefan Philipp zu finden.

Befehl	Beschreibung
poweron	Schaltet die Stromversorgung des Hosts ein.
poweroff	Schaltet die Stromversorgung des Hosts aus.
reset	Löst beim Host einen Reset aus.
stat	Zeigt den aktuellen Status des Hosts an.
help	Zeigt alle verfügbaren Befehle an.
exit, quit, logoff, logout	Beendet die Verbindung.

Tabelle 6.1: CIA-Befehle

Die andere Möglichkeit, mit dem µCsimm zu kommunizieren, besteht in einer binären Verbindung. Das Programm "tcpserv" stellt hierfür auf CIA-Seite einen Port zur Verfügung. Ereignisse von Tastatur und Maus werden über diese Schnittstelle an den TCP-Server und Bildschirmdaten vom TCP-Server übertragen. Die Verbindung ist rein binär und als TCP-Stream realisiert. Somit ist sichergestellt, dass sämtliche zu übertragende Informationen auch in der richtigen Reihenfolge am Ziel – dem µCsimm – ankommen.
Bisher implementierte Befehle:

6.2 Software

Der Janus-Client ist modular aufgebaut und gliedert sich in mehrere Klassen, die jedoch untereinander sehr eng miteinander verbunden sind. Zunächst findet eine Einführung in die Thematik Client und Java statt, anschließend wird auf die Besonderheiten der einzelnen Klassen näher eingegangen. Der Aufbau des ganzen Programms ist in Abbildung "Janus-Programm" zu finden.

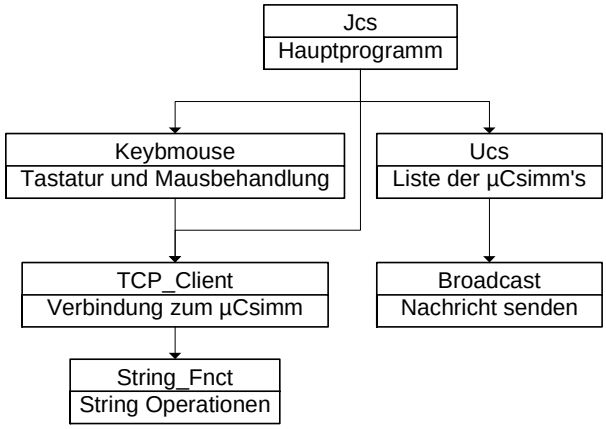


Abbildung 6.1: Janus-Programm

Der Client (Janus) wurde in Java geschrieben, um eine maximale Portabilität und einfache Installation zu erreichen. Allerdings bringt Java einige Besonderheiten mit sich, die im folgenden näher erläutert werden. Es wäre schön gewesen, Janus als Applet laufen zu lassen, was den großen Vorteil gehabt hätte, dass keine Java Runtime Umgebung notwendig gewesen wäre. Jedoch gibt es zwei große Einschränkungen, die diese Vorgehensweise nicht empfehlenswert machen: die eine wäre der beschränkte Dateizugriff, was nicht so problematisch gewesen wäre, da nicht nötig. Die zweite wäre der Zugriff auf das Netzwerk: Java Applets lassen keinen Zugriff auf andere Rechner zu, mit Ausnahme des Servers, von welchem das Applet heruntergeladen wurde. Die Einschränkungen mögen zwar für normale Applets ein großer Vorteil in Bezug auf Sicherheit sein, hier jedoch sind sie sehr hinderlich, da der Applet-Server in diesem Falle eine Proxy Schnittstelle zusätzlich emulieren müsste, um die einzelnen µCsimm's zu erreichen. Es hätte zwar die Möglichkeit gegeben, einen Zertifikatsserver einzurichten (ein Zertifikatsserver ist ein Server im Internet, der bestätigt, dass ein Javaprogramm sicher und authentisch

Kommando	Werte senden	Wert empfangen	Beschreibung
		'o'	Einmal am Anfang; Kennzeichen, dass die Verbindung erfolgreich aufgebaut worden ist.
'k'	a		Tastencode a senden, der <128 ist.
'l'	a & 7Fh		Tastencode a senden, der ≥128 ist.
'm'	a, b,c		Mausposition senden; eine Erweiterung auf das Vorzeichen bei b und c findet auf TCP-Server-Seite statt.
'b'		l, h, [n]	Bildschirm anfordern. Empfangen wird erst die Größe des Puffers (low, high), danach die Daten.
'q'			Verbindung wird getrennt.

Tabelle 6.2: Janus-Befehle

ist), aber diese Möglichkeit stand nicht zur Verfügung. Außerdem wäre dies auch keine Garantie für den Erfolg. Die soeben erwähnte Vorgehensweise hätte zwar einen großen Sicherheitsaspekt, was jedoch im gegebenen Fall nicht so hohe Priorität hat. Im Übrigen ist auch noch der Unterschied zwischen Netscape und Internet Explorer zu berücksichtigen, welche keine hohe Kompatibilität besitzen. Aus den genannten Gründen fiel die Entscheidung auf eine Java Applikation. Eine spätere Umstellung auf Applets ist trotzdem noch gegeben und sollte sich als nicht allzu schwierig erweisen.
Die Installation der Java-Runtime kann auch umgangen werden, da alles aus einem vom Server importierten Verzeichnis gestartet werden kann. Dazu müssen die Dateien unter Windows in das "\bin"-Verzeichnis des jdk kopiert werden.
Das Zusammenspiel zwischen Janus-Client und dem CIA-Server funktioniert problemlos, wobei die Übertragung der Daten als ASCII Kommandos nicht gerade für sehr elegant gehalten wird, da sich hier doch leicht Zeichenketten-Erkennungsprobleme einschleichen können und die Synchronisation gestört werden kann. Das Aufschalten per "telnet" Verbindung ist ein großer Vorteil, da keine gesonderten Programme benötigt werden und die Steuerkommandos direkt über die Tastatur eingegeben werden können.
Die andere Kommunikation zum Steuern des Hosts findet auf binärer Basis statt. Es wird ein binärer TCP-Sockel zur Übertragung sämtlicher Steuerbefehle erzeugt. Da Java nicht für Low Level Bitverarbeitung gedacht ist, treten hier etliche Probleme auf. Dies wird bei der Typumwandlung von Byte und Integer deutlich. Effekte, wie zum Beispiel Konvertierung negativer Werte zu positiven, erwiesen sich als problematisch. Ein großes Problem zeigte sich beim Senden der binären Daten. Hier überträgt Java erstaunlicherweise die Werte zwischen 80h und A5h nicht korrekt, sondern als 63h. Der Grund hierfür konnte nicht gefunden werden, da das Problem beim Empfangen nicht auftrat. Um das Problem zu umgehen, wurde Bit 7 einfach im vorhergehenden Byte mitübertragen und im zweiten ausgeblendet. Dies muss natürlich auf der Gegenstelle wieder rückgängig gemacht werden (siehe Tastaturübertragung). Beim Empfang von Daten ist zu beachten, dass die Zeichenketten nicht unbedingt an einem Stück empfangen werden, sondern dass die "Read"-Methode auch einmal nur ein einziges Zeichen zurückgeben kann. Die so erhaltenen Stücke müssen erst wieder in einem String zusammengebaut werden. Hierbei gibt es zwei Möglichkeiten: bei der Bildschirmübertragung wird zu Beginn erst die Anzahl der nachfolgenden Daten gesendet; somit steht fest, wie lange gelesen werden muss. Im Falle des CIA-Servers steht die Länge nicht fest, daher wird auf ein bestimmtes Zeichen gewartet, meist ":".
Threads unter Java zu erzeugen und zu steuern ist einfach und komfortabel und bereitet keinen großen Aufwand. Ein IP Scan fehlt jedoch, da es unter der zur Zeit aktuellsten Java-Version (1.2) nicht möglich ist ICMP Nachrichten auszusenden. Eine vorläufige Lösung ist die Speicherung der IP-Adressen in der Datei "ip_addr.txt". Dort können die IP-Adressen der µCsimm's im Klartext eingetragen werden. Ein Scan der IP-Adressen ist demnach nur über UDP möglich. Versuche, einen solchen Scan zu verwirklichen, wurden unternommen. Auf der Seite von Java scheint dies schon zu funktionieren, auf der µCsimm Seite ist hier jedoch noch etwas Arbeit angesagt. Die Grundlagen sind in der Datei "udp.c" zu finden.
In den folgenden Abschnitten werden die einzelnen Java-Module (Klassen) näher erklärt und auf deren Funktionsweise eingegangen:

Jcs:
Ist das eigentliche Hauptprogramm. Es sorgt dafür, dass die Bildschirmoberfläche dargestellt wird und leitet die gewählten Aktionen an die anderen Klassen weiter. Eine exakte grafische Bildschirmoberfläche (GUI) ist unter Java nicht so einfach wie unter anderen Programmiersprachen, da die Funktionen kein genaues Platzieren der Steuerelemente zulassen. Die API's der Betriebssysteme bieten die Möglichkeit die Position und Größe festzulegen, Java hingegen macht alles intern und nicht transparent. Programmiersprachen wie C oder Basic halten sich im Normalfall an die Schnittstelle des Betriebssystems, sind dafür aber nicht so portabel. Java stellt für die Gestaltung sogenannte Layoutmanager zur Verfügung, deren Bedienung für Anfänger aber nicht einfach zu verstehen ist. So werden Buttons der Reihe nach an einen Layoutmanager übergeben. Die Darstellung, ob diese nun in einer oder mehr Zeilen erfolgt, erledigt Java vollständig selbst. Ein Einfluss darauf kann nicht erfolgen. Zum Beispiel kann der Bildschirm über den "BorderLayout" Manager in folgende Bereiche eingeteilt werden:

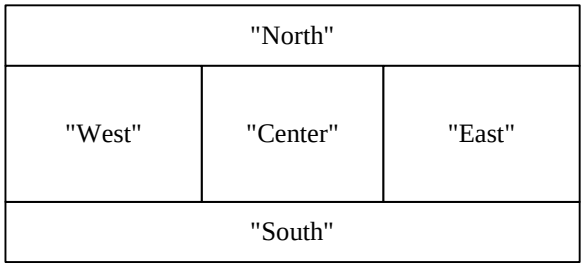


Abbildung 6.2: Border-Layout-Manager

Die Unterbereiche sind über die angegebenen Namen ansprechbar. Eine Festlegung der Größe der einzelnen Bereiche übernimmt wieder Java. Die komplette Darstellung übernimmt der Konstruktor in Klasse "Jcs". Den einzelnen Steuerelementen werden sofort die entsprechenden Ereignisbehandlungsroutinen zugewiesen. "focusGained", "focusLost" und "actionPerformed" sind Ereignismethoden, die Eingaben in das aktuelle Fenster behandeln. Sie verzweigen dann gegebenenfalls in entsprechende Routinen und sind für die richtige Reaktion des Programms verantwortlich. Zu beachten ist hier besonders die Behandlung der Tabulator-Taste. Da Java beim Drücken dieser Taste nicht den Code weitergibt, sondern den Eingabefokus auf den nächsten Knopf setzt, wurde ein kleiner Trick angewandt: wenn

zuvor der Button “Input” aktiv war (den Focus hatte) und jetzt der Knopf “Tab”, wurde die Tabulatortaste gedrückt und der Code der Tabulatortaste wird an das µCsimm weitergegeben. Die Eingabe wird anschließend wieder auf “Input” zurückgesetzt. Ein Mausklick auf die Tabulator-Schaltfläche hat dieselben Auswirkungen. Eingaben “Reset”, “PowerOn” und “PowerOff” werden an die Methode “SendCmd” der Klasse “TCP_Client” weitergeleitet – Ereignisse, welche die linke Auswahlliste der µCsimm‘s betrifft an die Klasse “Ucs” – Mausbewegungen und Tastatureingaben an die Klasse “Keybmouse”. Die Methoden “start” und “stop” dienen dem Aufschalten und starten/stoppen eines Threads, welcher dann unabhängig vom eigentlichen Hauptprogramm arbeitet und für die Bildschirmdarstellung verantwortlich ist.

Keybmouse:
Die Klasse “Keybmouse” verwaltet sämtliche Ereignisse für Tastatur- und Mausbearbeitung. Viele Methoden enthalten keinen Code, da deren Behandlung nicht benötigt wird. Da Java aber eine Referenz auf diese Ereignisse braucht, dürfen sie nicht weggelassen werden, ansonsten weist der Java-Compiler bereits auf diesen Verstoß hin. Auf die meisten Methoden soll hier nicht eingegangen werden, da sie selbsterklärend sind. Interessant ist jedoch, dass Java die Taste “Alt Gr” als eine Abfolge der Taste “Strg” und “Alt” umsetzt. Ein eigener Keycode für diese Taste existiert also nicht, daher muss auf einen kleinen Kniff zurückgegriffen werden: sobald “Strg” gedrückt wird, speichert der JANUS die aktuelle Systemzeit. Wenn die Taste “Alt” gedrückt wird und festgestellt wurde, dass die Taste “Alt” noch gedrückt ist, wird die Zeit zwischen beiden Ereignissen ermittelt. Wenn die Zeitspanne kleiner als 1 ms war – was durch normales Tastendrücken fast nicht erreicht werden kann – wird anstelle der einzelnen Tastenkombinationen der Code für “Alt Gr” übermittelt. Am Ende der Überprüfungen steht dann der zu übermittelnde Tastencode fest. Nun muss noch der zu der Taste gehörende Scancode aus der Tabelle “codeout” gelesen werden. Als Scancodes gibt es in der Tabelle Werte, die 1 oder 2 Byte lang sind. Die Bytes werden in der Reihenfolge High/Low über die Funktion “sendKey” der Klasse “TCP_Client” an das µCsimm gesendet.

Ucs:
Sämtliche Eingaben, welche die Liste der µCsimm’s links betreffen, werden von dieser Klasse behandelt. Als Ereignishandler der Liste wird in “Jcs” die Methode “mouseClicked” angegeben; alle anderen Mausereignisse sind hier nicht weiter von Interesse und werden daher auch nicht behandelt (leere Funktionen). Sobald ein µCsimm ausgewählt wird, sorgt die Methode “mouseClicked” dafür, dass das Modul als gewählt angezeigt und dessen IP-Adresse in Klasse “TCP_Client” eingetragen wird. Eine Mehrfachauswahl wird zur Zeit noch unterbunden. Die Methode “loadAddresses” lädt die in der Datei “ip_addr.txt” gespeicherten IP-Adressen und wird vom Konstruktor “Ucs” aufgerufen. Ein eigenständig arbeitender Thread wird gestartet, der versucht die einzelnen µCsimm’s ständig anzupingen, um herauszufinden welche existieren und welche nicht. Weiterhin soll der Status dieser Module dann später einmal direkt in der linken Liste durch diese Funktion angezeigt werden.

Broadcast:
Die Klasse Broadcast ist noch im Entwicklungsstadium. Später wird sie Scans der IP-Netze übernehmen, was folgendermaßen angedacht ist: in regelmäßigen Abständen werden UDP-Pakete in bestimmte Netze gesendet und auf deren Antwort gewartet. Diese Vorgehensweise ist dann sinnvoll, sobald mehrere µCsimm’s existieren, die komplett dynamisch verwaltet werden sollen. Bisher wurde versucht ein UDP-Paket an ein Netzwerk zu versenden. Im Ansatz funktioniert es bereits, jedoch auf Seite der µCsimm’s wurde das Empfangen und Versenden solcher Nachrichten noch nicht richtig implementiert. Als Befehl wird ein Paket mit “SCAN ” als Inhalt gesendet. Später sollen sich die einzelnen µCsimm’s hierauf melden und ein Paket mit dem aktuellen Status und weiteren Informationen zurücksenden. Als Informationen sind die IP-Adresse des µCsimm’s, die des Hosts und ein Aliasname des µCsimm’s geplant. Der Status soll den aktuellen Zustand (PowerON/PowerOFF) und Warnungen (Temperatur/ Lüfter) des Hosts enthalten. Die so erhaltenen Informationen können dann direkt in der linken Auswahlliste angezeigt werden. Sie informieren den Benutzer sofort über den aktuellen Zustand des Rechners.

TCP_Client:
Der TCP_Client ist der Dreh- und Angelpunkt der ganzen Kommunikation mit den µCsimm’s. Die Abfrage nach den vorhanden Karten wird jedoch die Klasse “Broadcast” übernehmen.
Nun aber zu den einzelnen Methoden: “Login” und “Logout” dienen dazu eine Verbindung zum µCsimm aufzubauen und wieder zu beenden. Über die so aufgebaute Verbindung werden dann die Maus-, Tastatur- und Bildschirmdaten übertragen. Da der Bildschirm gepackt übertragen wird, muss er vor der Darstellung erst wieder entpackt werden. Hierfür ist die “Decode-Methode” verantwortlich, die den Packvorgang wieder rückgängig macht. Die Funktionsweise des Algorithmus kann in Kapitel Grafikinterface nachgelesen werden. Die Darstellung des Bildschirms erfolgt durch die Methode “paintAll” in “Jcs”. “sendMouse” dient dem Übertragen der aktuellen Mauszeigerposition und dem Zustand der Maustasten an das µCsimm. Erwähnenswert hierbei ist, dass keine Werte größer als 50 übertragen werden, um mögliche Beschleuniger der Betriebssysteme auf Seite des Hosts weitestgehend auszuschalten, da sonst Diskrepanzen zwischen dem Mauszeiger des Hosts und JANUS entstehen können. Die so entstehenden Effekte sind eher kosmetischer Natur. Falls ein Bewegungsoffset wirklich einmal größer sein sollte, so wird der Wert in mehrere Pakete aufgeteilt.

Um Tastendrücke zu versenden, wurde die Methode “sendKey” geschrieben. Da der zu sendende Scancode aus mehreren Bytes bestehen kann, ist “sendKey” als rekursive Funktion ausgelegt, welche die ihr übergebenen Werte übermittelt. Hierbei werden die höchstwertigen Bytes zuerst übertragen. In der Tabelle “codeout” können somit einfach die Werte der entsprechenden Taste eingetragen werden, auch wenn die Taste aus einer Abfolge von mehreren Bytes besteht. Die Methode “relOut” ist mit der vorherigen fast identisch, nur dass der Code für das Loslassen einer Taste versendet wird. Hierzu ist eine Konvertierung des Codes in der Tabelle nötig. Die genaue Konvertierung wurde bereits in Kapitel Tastatur- und Mausemulation ausführlich erklärt und wird in der Klasse “Keybmouse” ausgeführt. Die nun so entstandenen Bytes müssen aber noch über die TCP-Schnittstelle übertragen werden. Hierfür sorgt “sendCode”. Da Java ein Problem mit der Übermittlung von Werten größer als 127 hat, werden diese Bytes konvertiert: wenn der Wert größer als 127 ist, so sendet der Java-Client ‚l‘ und das Byte minus 128, ansonsten ‚k‘ und den Wert. Der TCP-Server hat dann die Aufgabe diese Kommandos zu interpretieren und das fehlende höchstwertige Bit wieder herzustellen.
Pollscreen fordert über das Kommando ‚b‘ eine Bildschirmseite an. Sobald die Seite vom µCsimm übertragen wurde, wird sie mit Hilfe der “Decode” Methode entpackt. Der Inhalt der Bildschirmseite wird zurückgegeben. Später einmal soll hier ein anderer Packalgorithmus greifen, welcher effektiver arbeitet (entweder nur Deltas überträgt oder gleich die Steuersequenzen des BIOS interpretiert).
“SendCmd” ist die erste Methode, welche geschrieben wurde. Sie dient der Kommunikation mit dem CIA-Server der µCsimm’s von Stefan Philipp. Um einen Befehl auszuführen, loggt sie sich bei dem CIA-Server des µCsimm ein, überträgt den angegebenen Befehl und trennt danach wieder die Verbindung.

String_Fnct:
Die Klasse “String_Fnct” beinhaltet vor allem die Behandlung von Zeichenketten. “LoginSucc” ist zum Beispiel eine Methode, welche den Erfolg eines Loginversuchs überprüft. Es werden dabei die ersten beiden Zeichen eingelesen und mit “ok” verglichen. GetString hingegen ist eine Methode, die das Vorkommen einer Zeichenfolge in einem Zeichenstrom testet. Hauptsächlich wird sie für die Login-Prozedur benutzt, um zum Beispiel den Zeichenstrom auf die Zeichenkette “pass:” hin zu überprüfen. Folgende Arbeitsweise liegt dieser Methode zugrunde: um die Geschwindigkeit zu erhöhen, wird zuerst nach dem letzten Zeichen gesucht. Sobald dieses gefunden ist, wird auch der vorhergehende Teil verglichen. Die Methode PutString fasst nur zwei Befehle zusammen, was die Lesbarkeit des kompletten Quelltextes verbessert.
In Klasse “String_Fnct” wird sofort klar, dass Java bei diesem Projekt nicht die beste Wahl ist. Der große Vorteil von Java ist wohl die Betriebssystemunabhängigkeit, allerdings ist die Konvertierung von verschiedenen Variablentypen sehr kompliziert. Folgendes Beispiel verdeutlicht dies: die TCP-Daten, welche über das Netz übertragen werden, werden entweder als Char oder als Byte geliefert. Um sie darzustellen, wird erst eine Methode aufgerufen, welche die einzelnen Zeichen zu einem kompletten String zusammensetzt. In der Klasse Broadcast fällt es sogar extrem auf: hier werden die einzelnen Elemente eines Byte-Arrays in ein Char-Array umgewandelt und zwar nicht über eine dafür geeignete Methode, sondern über eine selbst geschriebene for-Schleife Byte für Byte.

ip_addr.txt:
Da der IP-Scan im Moment noch nicht funktioniert, müssen die IP-Adressen der einzelnen µCsimm’s angegeben werden. In der Datei “ip_addr.txt” sind sie zu finden. In jeder Zeile kann genau ein Modul in der Form “aaa.bbb.ccc.ddd” angegeben werden. Später stehen in der Datei nur noch die IP-Netzwerke, welche gescannt werden sollen.

6.3 Bedienung

Sobald der Java-Client gestartet wird, werden die einzelnen µCsimm’s aus der Datei “ip_addr.txt” gelesen und in der linken Liste angezeigt, derzeit maximal 20. Über die Liste links kann jetzt der Host ausgewählt werden, welcher ferngesteuert werden soll (Achtung: die IP-Adresse ist die des µCsimm’s und nicht des Hosts!). Der “Reset”-Knopf dient dazu auf dem Host einen Reset auszulösen. Über “PowerOn” und “PowerOff” kann er ein- beziehungsweise ausgeschaltet werden. Die Knöpfe “Connect” und “Disconnect” dienen dem eigentlichen Aufschalten. “Connect” baut eine IP Verbindung zu dem ausgewählten Rechner auf. Sobald sie etabliert ist, wird mit der Darstellung des Bildschirminhalts begonnen. Das Aktualisierungsintervall kann bisher nur im Quelltext direkt eingestellt werden. Da zur Zeit noch kein Grafikinterface existiert, werden einfach nur drei verschiedene Bildschirme hintereinander dargestellt, die aber bereits vom µCsimm erzeugt und über das Netzwerk übertragen werden. Daher wird sich die Übertragungsgeschwindigkeit nicht mehr groß ändern. Mausbewegungen hingegen werden bereits korrekt übertragen und vom µCsimm an den Host weitergegeben. Für Tastatureingaben gilt dasselbe. Sondertasten wie Tabulator und Windowstasten weisen einige Spezialitäten auf: sie sind auf der rechten Seite gesondert zu finden. Der Tabulator der Tastatur funktioniert, jedoch ist der Tabulator-Knopf erforderlich, siehe Java Klasse “Jcs” und deren Beschreibung oben. Der Knopf “Input” hat keine Funktion, ist aber nötig, um den Tabulator der Tastatur zu realisieren; daher sollte er im aufgeschalteten Zustand die Eingabe (den Fokus) haben. Da die Windowstasten von Java aus nicht abgefragt werden können, existieren die Knöpfe “Windows” und “RMouseKey” als Knopf. Über “Toggle y/z” ist es möglich die Tasten “y” und “z” zu vertauschen. Das ist besonders sinnvoll, falls mit amerikanischen Tastaturen gearbeitet wird oder der Host noch keinen deutschen Tastaturtreiber geladen hat. Beim deutschen Tastaturlayout sind die Beschriftungen “y” und

“z” gegenüber dem amerikanischen vertauscht, nicht jedoch die Scancodes des Layouts. Um ein Soft-Reset auf dem Host auszuführen (Ctrl+Alt+Entf), muss eine leicht abgewandelte Tastenkombination (Ctrl+Alt+Backspace) verwendet werden, da der Soft-Reset nicht abgefangen werden kann und sich auf den lokalen Rechner sofort auswirkt. Bei der Tastenkombination wurde sich an das Programm “ProxyMaster” der Firma Funk gehalten. Über “Disconnect” kann die Verbindung nach dem Arbeiten wieder beendet werden. Es wird nicht empfohlen ein Fenster vor den Java-Client zu ziehen, wenn dieser gerade eine aktive Verbindung aufgebaut hat, da die Übertragung sich sonst drastisch verlangsamt und auch zu kompletten “Hängern” führen kann. Daher auch die Empfehlung das Programm nach C zu portieren, um mehr Leistung zu erreichen und besser auf das Betriebssystem zugreifen zu können.
Die folgende Abbildung gibt die Oberfläche des Java-Clients wieder mit der Beschreibung der einzelnen Bedienelemente.

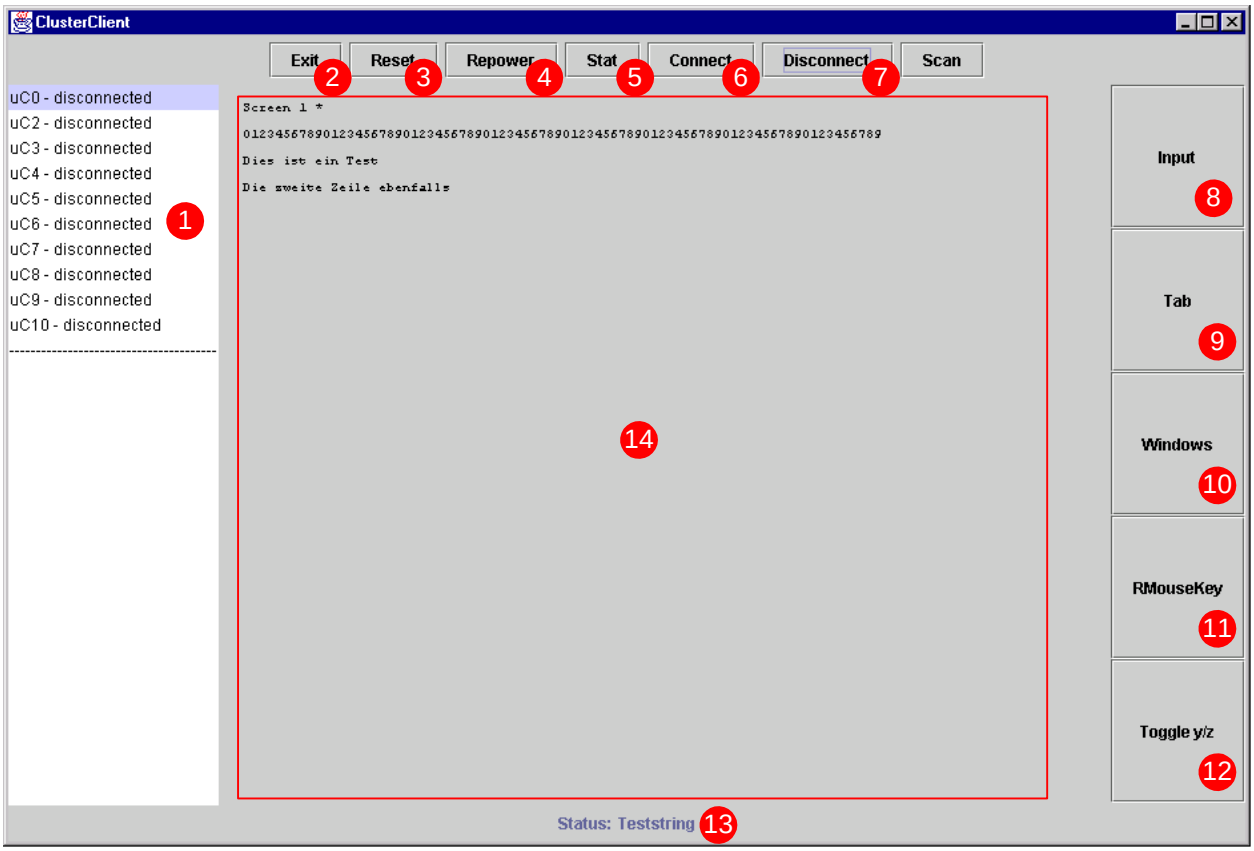


Abbildung 6.3: Janus Oberfläche

1. Liste der auswählbaren µCsimm’s. Bei einem Klick auf ein entsprechendes wird dieses wie folgt angezeigt: “aaa.bbb.ccc.ddd – connectable”. “disconnected” steht dafür, dass zu diesem Modul gerade keine Verbindung vorhanden ist. “connected” bedeutet, dass gerade eine Verbindung existiert.
2. Beendet das komplette Programm.
3. Hierüber wird der ausgewählte Rechner zurückgesetzt.
4. Der ausgewählte Rechner wird ein- und wieder ausgeschaltet.
5. Zeigt den aktuellen Status des ausgewählten µCsimm’s an.
6. Baut eine Verbindung zu dem ausgewählten Rechner auf.
7. Beendet eine Verbindung wieder.
8. Nicht benutzter Knopf.
9. Emuliert die Tabulator-Taste. Diese Taste funktioniert aber auch über die Tastatur.
10. Emuliert die Windowstaste.
11. Emuliert die rechte Maustaste,
12. Vertauscht die Tasten ‚y‘ und ‚z‘.
13. Falls Stat ausgewählt wurde, wird hier das Ergebnis angezeigt.

7 Boot-Server

Da verschiede Daten abgelegt werden müssen, ist ein solcher Server unabdingbar. Er erfüllt gleich mehrere Aufgaben, die jedoch auf verschiedene Server verteilt werden können.
Zur Zeit der Entwicklung ist es sinnvoll, nicht alle Daten des TCP-Servers auf jedem µCsimm zu halten, sondern auf einem zentralen Server, welcher die Dateien über eine NFS-Freigabe zur Verfügung stellt. Ebenfalls ist es sinnvoll eine Version des CIA- und TCP-Servers direkt in das Flash der µCsimm's mit aufzunehmen, um bei einem Netzausfall dennoch die Server starten zu können. Falls eine Netzverbindung vorhanden ist, wird die aktuelle Software direkt vom NFS-Server heruntergeladen. Eine Warnung sei hier noch erwähnt: ein Update des Kernels aller µCsimm's wird eine geraume Zeit dauern! Auf einem Server werden auch die Images der Disketten abgelegt. Die µCsimm's können sie dann von dort herunterziehen und im weiteren Verlauf an den Host weitergeben. Eine Speicherung der Daten der µCsimm IP-Scans ist hier sicherlich sinnvoll, um Statistiken über Ausfälle machen zu können – dies wird jedoch bisher noch nicht unterstützt.

Es ist auch noch zu erwähnen, dass eine rechnerabhängige Boot-Floppy-Konfiguration sinnvoll wäre. Das ließe sich von zwei verschiedenen Seiten aus lösen: zum einen kann das µCsimm versuchen ein entsprechendes Image zu finden. Falls es eines gefunden hat, speichert es die Information darüber ab, am besten direkt auf dem Boot-Server (als Schlüssel bietet sich die MAC-Adresse an). Die zweite Möglichkeit wäre ein komplettes Programm, das auf einem Server läuft und von dort aus die ganze Verwaltung übernimmt. Persönlich würde ich die erste Methode empfehlen, da hier keine Abhängigkeit existiert. Der Janus-Client kann direkt lokal installiert werden. In diesem Falle muss zuerst Java installiert und anschließend die Dateien des Java-Clients in das Java-Verzeichnis kopiert werden. Die Vorgehensweise ist für Rechner zu empfehlen, welche nicht direkter Bestandteil des Netzwerks der CIA-Karten sind. Für Rechner innerhalb dieses Netzwerkes ist es sinnvoll, alle Daten inklusive der Java-Runtimeumgebung auf einem zentralen Server bereitzustellen. Dies wurde bei Siemens über die Dateifreigabe eines Novellservers bewerkstelligt. Unter Unix ist die Freigabe durch NFS sicherlich sinnvoller.

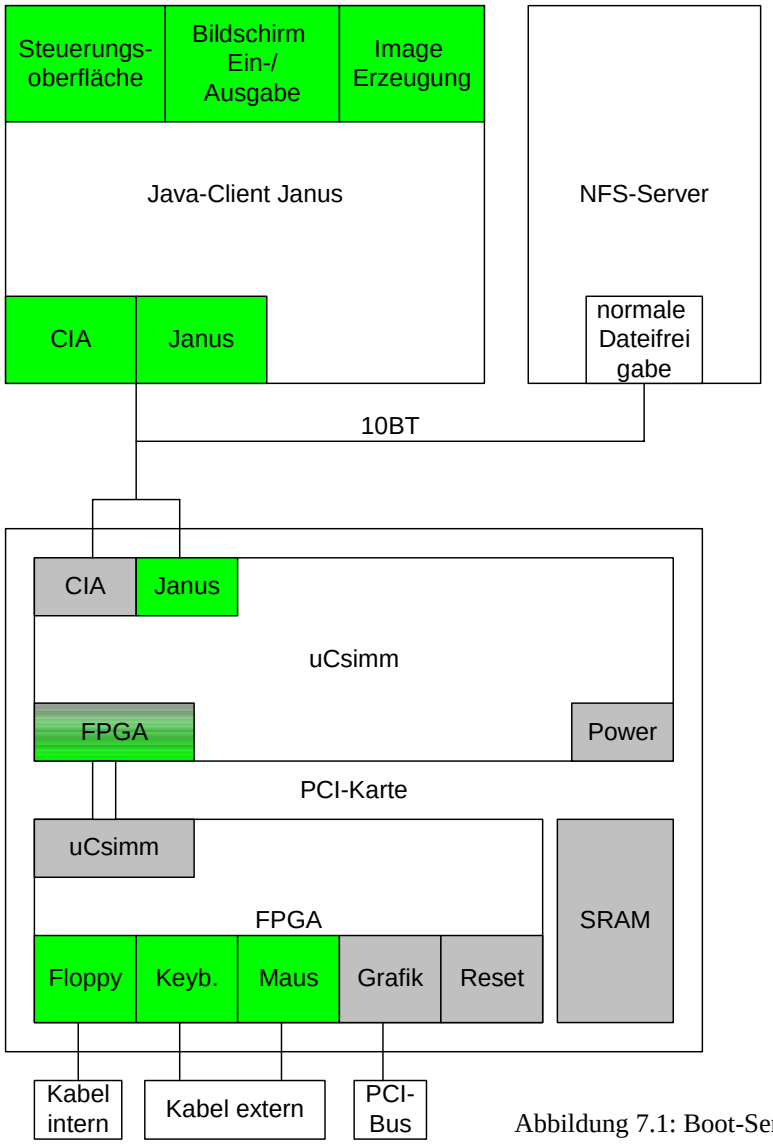


Abbildung 7.1: Boot-Server Konstellation

8 Verschiedenes

Da wegen der Größe des Projektes eine einzelne Person nicht alles in dem gegebenen Zeitrahmen bewältigen kann, wurde es in mehrere Teile zerteilt, die weitgehend unabhängig voneinander sind. Es wurde darauf geachtet, so wenig Schnittstellen wie möglich zu haben, damit bei einem Fehlschlag der einen die andere nicht darunter leidet. Der eine Teil, den ich übernommen habe, wurde bereits beschrieben. Der andere Teil, welcher von meinem Kommilitonen Stefan Philipp ebenfalls als Diplomarbeit bearbeitet wurde, wird im folgenden kurz beschrieben. Da bisher noch nicht alles realisiert wurde, was geplant war, wird es noch weitere Arbeiten über dieses Thema geben. Die noch ungelösten Teile werden in einem Kapitel am Schluss aufgelistet. Nun zuerst zu den Teilen, die Stefan Philipp bearbeitet hat:

8.1 FPGA-Interface

Zentralstück zwischen µCsimm und FPGA ist das FPGA-Interface. Auf diesem basieren alle Anwendungen, die in dieser Diplomarbeit bearbeitet wurden. Das Interface besteht aus zwei Teilen. Der eine Teil im µCsimm besteht aus einem C-Programm, das folgende Funktionen zur Verfügung stellt:

- Fpga_read_conf()
  Diese Funktion liest ein Konfigurationsregister aus der FPGA. In den Registern werden die einzelnen Module wie Tastatur und Maus abgebildet. In nachstehender Tabelle werden die einzelnen Register aufgezeigt.
- Fpga_write_conf()
  Hierüber können Werte in ein Konfigurationsregister geschrieben werden.
- Fpga_read_sram()
  Daten, welche im externen SRAM liegen, werden mit dieser Funktion eingelesen.
- Fpga_write_sram()
  Über diese Funktion können Werte in das externe SRAM geschrieben werden.
- Fpga_init()
  Bevor Daten von/zur FPGA übertragen werden, muss die Verbindung hierüber initialisiert werden.

In Tabelle „CIA-Registerbelegung“ auf der nächsten Seite werden die aktuellen Registermappings der einzelnen Module aufgelistet.
Das SRAM wird im Moment nur dazu verwendet einen Track der Floppy zwischenzuspeichern. Später einmal soll das RAM hauptsächlich als Grafikspeicher dienen. Ein Arbiter, der ein Sharing des Speichers zwischen mehreren Modulen managt, müsste noch geschrieben werden.
Der Zugriff auf die Register erfolgt byteweise. Falls ein Modul mehr als 8 Bit benötigt, werden einfach mehrere Register benutzt. Der Zugriff auf die PS/2-Schnittstelle findet zum Beispiel über zwei Register statt. Das eine enthält die Daten zum Senden beziehungsweise die Daten, welche empfangen wurden. Das andere dient der Ablaufsteuerung des PS/2 Protokolls.
Da das Design von Stefan Philipp zu groß für die benutzte FPGA war und außerdem ein 32-Bit breiter Datenbus zum RAM benutzt wurde, der hier ebenfalls nur 8-Bit Breite hatte, wurde als Schnittstelle ein eigener Treiber benutzt. Über diesen kann immer nur eine Entity getestet werden, was zum Funktionstest der einzelnen Komponenten jedoch vollkommen ausreicht. Als Anbindung wurde kein großartiges Protokoll benutzt, lediglich eine Leitung für die Richtung der Daten, eine zum Schreiben und eine zur Wahl des Registers. Ein kompletter Zusammenbau der beiden Arbeiten steht also noch aus. Er ist aber erst dann sinnvoll, wenn eine erste Fassung der späteren CIA-Karte existiert.

8.2 Port 80

Port 80 ist ein Register im Hauptspeicher eines jeden PC’s. Beim Start legt hier jeder PC den aktuellen Status (Initialisierung der Geräte, Speicherscan) ab. Falls ein Rechner hängen bleibt, ist dann eine Diagnostik möglich, an welcher Stelle genau der PC stehen geblieben ist und welches Gerät einen Fehler erzeugt hat. Das Register soll später einmal vom µCsimm ausgelesen und dort mitprotokolliert werden. Der Name Port 80 kommt daher, dass dieses Register an Adresse 80h im I/O-Bereich eines PC’s von IBM implementiert worden ist. Andere BIOS-Hersteller (Compaq) verwenden zum Teil andere Adressen. Daher muss es möglich sein, die Adresse zu ändern. Da dieser Port während der Entwicklung der Grafikkarte ebenfalls von Interesse war, wird in nachfolgender Tabelle ein kleiner Ausschnitt der Codes wiedergegeben.

Wert	Bedeutung
05h	Software-Reset/Power-on erkannt, Cache wird ausgeschaltet, wenn nötig
2Dh	Video-ROM-Kontrolle fertig
2Fh	EGA/VGA nicht gefunden, Grafikspeicher Lese-/Schreibtest beginnt
37h	Grafikmodus gesetzt, Power-on-Message wird dargestellt
4Eh	Speichertest gestartet
80h	Tastaturtest gestartet, Outputbuffer werden
91h	Floppy-Setup fertig, Harddisk-Setup wird ausgeführt
9Dh	Coprozessor initialisiert
00h	Kontrolle wird an #19 zum Booten übertragen

Tabelle 8.1: Port80 Codes des AMI-WinBIOS

Ein µCsimm-Modul hierfür muss noch entwickelt werden, genauso wie die Darstellung über den Java-Client.

Adresse	7	6	5	4	3	2	1	0	Bedeutung
00h-02h									Kennung der FPGA: "CIA"
03h	•	•	•	•					Version, derzeit 1.
					•	•	•	•	Revision, derzeit .0
08h									Interrupt Register, =1 falls Interrupt aufgetreten ist
						•			PS/2 Maus
							•		PS/2 Tastatur
								•	Floppy
0Ch									7 Segment Anzeige, 0=POST-Werte, 1=Floppy-Track
									µCsimm Komponente, zur Fehlersuche über PCI eingebaut
10h									Kontrollsignale für MUX Ansteuerung SRAM
						•			uc_write
							•		uc_read
								•	uc_enable
12h									Aktueller Zustand der State-Machine
13h									Aktuelles Kommando
14h-15h									Aktuelle Adresse
18h-1Bh									uc_data_in, Daten zum µCsimm
1Ch-1Fh									uc_data_out, Daten vom µCsimm
									Floppy Komponente, Signale zur Steuerung/Fehlersuche
20h									Steuerregister für µCsimm, =1 falls Signal aktiv ist
					•				fc_reset
						•			fc_dskchng
							•		fc_ack
								•	fc_req
23h									Signale von Floppy-Kabel, =0 für aktives Signal (inverse Logik)
				•					fe_head
					•				fe_step
						•			fe_dir
							•		fe_motor
								•	fe_drvsel
28h									Aktueller Track
29h									Aktueller Zustand der Statemachine
2Ah-2Bh									Aktuelle SRAM Adresse der Diskettendaten
2Ch-2Fh									Aktuelles Datenword, welches kodiert wird, 32Bit
									PS/2 Tastaturkomponente
30h									PS/2-Tastatur Kontrollregister, =1 falls Signal aktiv ist
				•					kb_irqreq
					•				kb_reset
						•			kb_error
							•		kb_ack
								•	kb_req
31h									kb_data_in, an den Host zu sendendes Byte
32h									kb_data_out, vom Host empfangenes Byte
33h									Aktueller Zustand der Statemachine
									PS/2-Mauskomponente
34h									PS/2-Maus Kontrollregister, =1 falls Signal aktiv ist
				•					ms_irqreq
					•				ms_reset
						•			ms_error
							•		ms_ack
								•	ms_req
35h									mms_data_in, an den Host zu sendendes Byte
36h									mms_data_out, vom Host zu empfangenes Byte
37h									Aktueller Zustand der Statemachine
									POST Komponente
38h									Der letzte Wert, welcher an die IO Adresse geschrieben wurde
3Ah-3Bh									Zu überwachende IO Adresse

Tabelle 8.1: CIA-Registerbelegung

8.3 Minitools

Es existiert eine Reihe kleinerer Tools, die zum Beispiel Register auslesen, das SRAM testen, die PCI-Schnittstelle überprüfen sowie weitere Aufgaben erledigen können. Diese sind entstanden, um alles auf korrekte Funktionalität zu testen.

8.4 CIA-Server

Es existiert des weiteren noch ein zusätzliches Programm, welches auf dem µCsimm läuft. Dieses wurde von Stefan Philipp geschrieben und dient der Strom- und Resetverwaltung des Hosts. Bei diesem Programm kann sich der Java-Client oder sogar ein normaler Benutzer über eine Telnet-Sitzung einloggen. Das Programm ist über Port 77 der entsprechenden IP-Adresse zu erreichen und bietet Befehle wie: PowerOn, PowerOff und Reset. Über den Befehl „help“ kann eine Übersicht über alle Befehle angezeigt werden. Die Oberfläche ist ähnlich der einer Telnet Sitzung, Die Befehle desselben wurden bereits im Kapitel „Janus-Client“ beschrieben. Es war vorgesehen, den Host noch über eine andere Schnittstelle bedienen zu können, falls das Hauptprogramm “tcpserv” einmal abstürzt. Das Programm war von Stefan Philipp geplant gewesen und sollte eigentlich das einzige sein, das auf einem µCsimm läuft. Es erschien jedoch nicht sinnvoll, die komplette Abwicklung der Daten über dieses Programm abzuwickeln. Außerdem wäre hierbei ein viel größeres Schnittstellenproblem aufgetreten.

9 Konklusion

9.1 Ergebnisse

Mit dieser Diplomarbeit wurde gezeigt, dass es möglich ist, einen Rechner komplett von der Ferne aus zu administrieren. Bedauerlicherweise war das Thema so umfangreich, dass am Ende der Diplomarbeit kein kompletter Test durchgeführt werden konnte. Es wurden aber alle Module einzeln getestet und dies mit Erfolg. Da die einzelnen Module so gut wie keine Überschneidungen haben, wird es sicherlich beim Zusammensetzen der einzelnen Teile keine Probleme geben. Der einzige unbekannte Faktor ist dann die Leistungsfähigkeit des Prozessors, auf dem das Programm läuft(µCsimm). Hier kann es durchaus sein, dass das µCsimm in gewissen Situationen überfordert ist. Abhilfen hierfür dürften jedoch nicht problematisch sein, da es genügend andere, sehr viel leistungsfähigere Prozessoren auf dem Markt gibt (siehe auch Excalibur).
Alle Teile haben auf einer einfachen PCI-Karte genügend Platz. Falls irgendwann einmal noch einige weitere Erweiterungen vonnöten sind, können diese problemlos noch auf einer langen PCI-Karte untergebracht werden. Da auf der Karte später nur noch eine FPGA mit externem RAM/FLASH sitzen wird (vielleicht wird es auch irgendwann einmal eine echte ASIC-Version geben), dürfte sich der Preis der kompletten Karte von der reinen Herstellungsseite nicht über 250,-€ bewegen. Dies bedeutet für den normalen Markt eine gute Chance. Im Vergleich zu der Karte von AMI sind hier sehr viel weniger Bauteile vorhanden. Auch die Stromversorgung ist wesentlich besser zu managen. Das µCsimm verbraucht circa 100 mA; dieser Strom steigt unter Belastung nicht weiter als 120 mA an. Es gibt sogar einen Modus, in dem das Modul weniger als 10 mA verbraucht. Bei der FPGA wird man hier sicherlich höhere Werte erhalten, kann diese jedoch gegebenenfalls abschalten, sofern der Rechner ausgeschaltet ist. Es würde jedoch in der Variante mit dem NIOS-Prozessor nicht funktionieren, da dieser als Soft-Core auf der FPGA läuft. Altera hat bereits eine Variante angekündigt (Excalibur), in der dieser Prozessor als ASIC-Version in der FPGA vorkommt. Hierüber stehen jedoch noch keine genaueren Details fest.

9.2 ToDo

Die im folgenden aufgelisteten Dinge müssen noch erledigt werden:

Port 80: Mitprotokollierung der Port 80 Kommandos durch das µCsimm und Darstellungsmöglichkeit durch den Java-Client.

FPGA-Schnittstelle: Implementierung der Schnittstelle zwischen FPGA und µCsimm. Bisher wurde nur eine eigene Schnittstelle benutzt. Ein Test mit der Schnittstelle von Stefan Philipp wurde bereits erfolgreich durchgeführt. Eine Implementation sollte keine Probleme bereiten, ist aber erst bei vorhandener CIA-Karte sinnvoll.

Grafikkarte: Eine vollständig funktionierende Grafikkarte fehlt noch. Viele Teile wurden bereits gelöst, die Anbindung des Video-Speichers an den PCI-Bus fehlt jedoch noch. Sobald sie existiert, kann das Grafikmodul im µCsimm fertiggestellt werden.

Mausprotokoll: Das Mausprotokoll wurde bisher nur teilweise getestet. Beim Starten des Betriebssystems wurde eine Originalmaus angeschlossen, danach wurde die Emulation erfolgreich getestet. Janus bereitet die Daten bereits korrekt auf. Das aktuelle C-Tastaturmodul, welches funktioniert, wurde zwar für die Maus angepasst, jedoch aus Zeitgründen nicht mehr getestet. Beim Testen sollten normalerweise keine Störungen auftreten, im Zweifelsfalle müsste die Erkennungssequenz einer echten Maus mitgeschnitten und analysiert werden. Dann ist es auch möglich, das Betriebssystem mit erfolgreicher Einbindung der Maus hochzufahren.

TCP-Server: Falls TCP-Pakete ankommen, sind diese nicht direkt den Modulen mit einer switch-case Abfrage zuzuordnen, sondern – über eine verkettete Liste – alle Module der Reihe nach durchzugehen. Das würde das Einbinden neuer Module noch weiter erleichtern.

9.3 Verbesserungsvorschläge

Folgende Punkte sind nicht unbedingt notwendig, jedoch wird durch die Maßnahmen vieles komfortabler und einfacher.

Java-Client: Mehrfachaufschalten: das heißt, es ist möglich, mehrere µCsimms gleichzeitig zu steuern. Dies bietet sich gerade bei der Installation von neuer Software an.

Scriptsteuerbar

Automatischer Scan nach µCsimm's

Ist jemand aufgeschaltet, kann dieser benachrichtigt werden.

Umschreiben nach C, da hier die Performance insbesondere bezüglich der späteren Grafikausgabe besser ist.

SRAM: RAM-Arbiter, der die Möglichkeit bietet, einen gleichzeitigen Zugriff auf das SRAM durch µCsimm, Floppy-Modul und PCI-Bus zu managen.

DHCPCD: Ein DHCPCD fehlt noch auf Seite des µCsimm's. Eine Portierung wäre nicht schlecht, jedoch gilt es zu bedenken, dass auf anderen Plattformen (µClinux auf ARM) ein solcher bereits existiert. Unter der Voraussetzung, dass das µCsimm nicht benutzt wird, erscheint dies eher obsolet und sollte erst in der Zielversion angegangen werden.

9.4 Aussichten

In der folgenden Auflistung werden Punkte angesprochen, die theoretisch alle möglich wären und die Karte zu einem abgerundeten Produkt machen würden.

- Eine IDE-Schnittstelle sollte kein Problem darstellen, da dies als Wishbone-Version zur Verfügung steht. Wishbone ist ein offener Standard eines Bus-Systems. Dieser ist für das Design von Hardware gedacht, insbesondere mit VHDL und Verilog.
- USB: für diese gibt es mehrere Möglichkeiten. Einmal steht wie bei IDE eine Wishbone-Version zur Verfügung, zum anderen kann dies auch mit Hilfe eines externen Chips zum Beispiel von Cypress erfolgen.
- USV-Schnittstelle
- GSM-Schnittstelle
- Websteuerbar (sollte kein Problem sein, da Webserver bereits integriert)
- Mikrofon zur Überwachung von Lüftern, Festplatten und des eingebauten PC-Lautsprechers
- InfraRedPort zur Kommunikation mit Handhelds
- Sleepmode benutzen, um Strom zu sparen
- Akkugepufferter Betrieb

9.5 Zeitplan

In der folgenden Tabelle ist der Zeitplan abgebildet, welcher den Verlauf der einzelnen Teile widerspiegelt.

In der Softwarebranche gibt es das sogenannte Verhältnis 1:Pi. Dies bedeutet, dass für alle realen Termine eine Verzögerung um die dreifache Zeit ein Durchschnittswert wäre. Die Termine im einzelnen wurden recht gut eingehalten, allerdings war festzustellen, dass die angegebenen Termine diejenigen für einen ersten erfolgreichen Test waren. Im Nachhinein wurde noch viel Zeit investiert, die einzelnen Komponenten zu überarbeiten, deren Timing zu verbessern und stabiler zu machen. Unter Berücksichtigung der Nachbearbeitungszeit wurde der Wert 1:Pi fast exakt erreicht. Einige Teile (wie Maus und VGA) sind noch nicht komplett implementiert. Der eigentlichen Implementation sollte mit den hier erarbeiteten Informationen jedoch nichts im Wege stehen.

9.6 Tipps

Auf dem µCsimm sollte die Option „Autoboot“ benutzt werden, um ein einfacheres Arbeiten zu erreichen.
Der Begriff “deprecated” bei Java bedeutet, dass eine Funktion veraltet ist und bei neueren Projekten nicht mehr verwendet werden sollte.
Die Pull-UP und Pull-Down Widerstände im Prozessor sollten genutzt werden.
Ctrl+R führ auf dem µCsimm einen Reset aus. Dies Funktioniert jedoch nur über die serielle Schnittstelle!

Geplanter Termin	Realer Termin	Status	Thema
15.12.	15.12.		TCP/IP Kommunikation verstanden und lauffähig
01.01.	20.12.		Janus: grundlegend implementiert (CIA-Server)
15.03.	15.01.	VGA-Karte fehlt	Janus: Übertragung von Bildinformationen
01.03.	15.01.		Floppy: Diskettenlaufwerk wird erkannt
01.03.	01.03.		Floppy: Lesen von Daten möglich
15.04.	01.04.		Tastatur geht
15.04.	01.04.		Maus geht
15.04.	01.10.		Tastaturerkennung geht
15.04.	-		Mauserkennung
01.06.	01.05.		Interrupttreiber
01.06.	15.05.		Signale versenden
01.10.	01.09.		VGA-BIOS
01.02.	01.08.	µCsimm SW fehlt	IP-Scan
04.09.	08.09.		Schreibbeginn Diplomarbeit
04.12.	04.12.		Diplomarbeit fertig

Tabelle 9.1: Zeitplan

10 Anhang

In diesem Anhang werden die einzelnen Utilities und deren Installation beschrieben.

10.1 Interrupttreiberinstallation

Eine gute Dokumentation zu diesem Thema findet sich im Buch von Allesandro Rubini[P5]. Um den notwendigen Interrupttreiber zu installieren, sind folgende Schritte notwendig (bezieht sich auf µClinux-Version 2.0.39):
Als ersten Schritt muss der Treiber sichtbar gemacht werden, damit er später mitkompilierbar ist. Dazu muss in der Datei (µClinuxdir)/linux/arch/m68knommu/config.in nachfolgenden Zeilen:

```
mainmenu_option next_comment
comment 'Character devices'
```

folgendes einfügt werden:

```
#Definition for FPGA by Martin Kirsch
if [ „$CONFIG_µCSIMM“ = „y“ ]; then
    bool 'FPGA-Driver' CONFIG_UCFPGA
fi
#Definition for FPGA END
```

Dies zeigt dann den Treiber in make menuconfig an. Der eigentliche Treiber (FPGA.c) muss in das Verzeichnis (µClinuxdir)/linux/drivers/char/ kopiert werden. Im Makefile im gleichen Verzeichnis muss nach folgender Zeile:

```
ifdef CONFIG_BAUD_RATE
CFLAGS += -
DCONSOLE_BAUD_RATE=$(CONSOLE_BAUD_RATE)
endif
```

das folgende eingefügt werden:

```
# Definition by Martin Kirsch for Communication
with FPGA
ifeq ($(CONFIG_UCFPGA),y)
    L_OBJS += FPGA.o
endif
# Definition by Martin Kirsch END
```

Damit der Treiber beim Systemstart automatisch eingebunden wird, muss er in die mem.c Routine eingefügt werden. Nach:

```
int chr_dev_init(void)
{
  if(register_chrdev(MEM_MAJOR,“mem“,
                     &memory_fops))
    printk(„unable to get major %d for
                     memory devs\n“,
                     MEM_MAJOR);
    rand_initialize();
    tty_init();
```

bitte folgendes einfügen:

```
#ifdef CONFIG_UCFPGA
    fpga_init();
#endif
```

Des weiteren wird noch die Datei fpga.h in (µClinuxdir)/linux/include/linux/ mit folgendem Inhalt benötigt:

```
/*FPGAUni*/
extern int fpga_init(void);
```

Der Treiber muss nun noch registrierbar sein. Dazu wird der Befehl mknod benötigt:

```
cp (µClinuxdir)/testenv/src/bin/mknod
    (µClinuxdir)/testenv/romdisk/sbin/mknod
```

Sinnvoll ist es noch den Treiber sofort beim Systemstart als Char-Device zuzuweisen. Dazu die Datei rc im Verzeichnis (µClinuxdir)/testenv/romdisk/etc/rc editieren und folgendes einfügen:

```
/sbin/mknod /var/fpga c 50 0
```

Nach diesen Schritten kann über “make menuconfig” der Treiber aktiviert, über make kompiliert und mithilfe des flashloader programmiert werden. Bei erfolgreichem Start von µClinux wird dieser Treiber angezeigt. Eine Überprüfung kann in folgenden Dateien im /proc Verzeichnis gefunden werden:

```
Modules – hier muss der Treiber sichtbar sein
Devices – hier sieht man dann das zugewiesene
    Device
Interrupts – Falls man Interrupts auslöst, werden
    diese hier gezählt
```

Um den Treiber nun zu benutzen, ist lediglich das Device zu öffnen und die eigene Prozessnummer, als Zahl mit einem p vorangestellt, zu senden. Falls nun ein Interrupt auftritt, wird ein Signal (SIG_USR1) an diesen Prozess gesendet. Nun kann auf den Interrupt entsprechend reagiert werden.

10.2 Java-Installation

Zum Zeitpunkt der Diplomarbeit war die Version 1.2 des JDK's (von der Firma Sun Microsystems) aktuell. Diese Version steht kostenlos im Internet zur Verfügung. Alle Angaben beziehen sich auf diese Version. Es sollte bei anderen Versionen keine Probleme geben, sofern sie Java2 kompatibel sind.
Die Installation vom JDK stellt kein Problem dar und sollte von jedem etwas erfahreneren PC-Benutzer durchgeführt werden können. Falls eigene Programme geschrieben werden, können sie direkt in den Ordner “jdk1.2\bin” kopiert werden. Dies hat den Vorteil, dass keine weiteren Angaben erfolgen müssen. Wichtig ist noch, dass die Variable “CLASSPATH” auf Windows Systemen auf dieses Verzeichnis zeigt, da sonst

importierte Klassen nicht gefunden werden können. Um das Java-Programm zu kompilieren, ist folgender Befehl in einer Eingabeaufforderung einzugeben:

```
C:\jdk1.2\bin>javac Jcs.java
```

Wichtig ist die Endung hinter dem Klassennamen. Die importierten Klassen werden schließlich automatisch auch kompiliert, sofern dies nötig ist. Ist das Programm fertig kompiliert, kann es mit:

```
C:\jdk1.2\bin>java Jcs
```

gestartet werden.

10.3 VGA Konfigurationsregister und Interrupts

Da einige der unten dargestellten Informationen in neueren Büchern nicht mehr abgedruckt werden, sind sie hier aufgeführt, um es nachfolgenden Programmierern zu erleichtern. Die detailliertesten Informationen müssten im Buch “IBM PS/2 Model 50 and 60 Technical Reference” zu finden sein. Leider stand dieses Werk nicht zur Verfügung. Die abgedruckten Informationen sind hauptsächlich aus [1].

Zusammenstellung allgemeiner Daten des VGA-Standards:

Farbtabelle	256 Farben
Speicherbereiche	03B0h-03DFh A0000h-BFFFFh (256kB)
horizontale Ablenkfrequenz	256 Farben
vertikale Ablenkfrequenz	31,5 kHz
horizontale Pixel (max)	50 bis 70 Hz
vertikale Pixel (max)	720
Zeichenmatrix	480
Zeichenmatrix	9x16
effektive Zeichengröße	7x9
Lichtgriffel	nein
analoger Ausgang	ja

Tabelle 10.1: VGA-Eigenschaften

Verwendung der E/A-Kanäle:

Registername	Registertyp	Farbe	Mono	beide	R/W
Ausgang für verschiedene Zwecke	allgemein			3C2h	W
Ausgang für verschiedene Zwecke	allgemein			3CCh	R
Eingabe Statusregister 0	allgemein			3C2h	R
Eingabe Statusregister 1	allgemein	3BAh	3BAh		R
Eigenschaften Steuerregister	allgemein	3BAh	3BAh		W
Eigenschaften Steuerregister	allgemein			3CAh	R
Video-Untersystem einschalten	allgemein			3C3h	RW
Adressregister	Attribut			3C0h	RW
weiteres Attributregister	Attribut			3C0h	W
weiteres Attributregister	Attribut			3C1h	R
Indexregister	CRT Kontroller	3B4h	3D4h	3D4h	RW
weiteres CRT Controller-Register	CRT Kontroller	3B5h	3D5h	3D5h	RW
Adressregister	Sequenzer			3C4h	RW
weiteres Sequenzer-Register	Sequenzer			3C5h	RW
Adressregister	Grafik			3CEh	RW
weiteres Grafikregister	Grafik			3CFh	RW
PEL Adress-Schreibmodus *)	Video DAC			3C8h	RW
PEL Adress-Lesemodus	Video DAC			3C7h	RW
DAC Statusregister	Video DAC			3C7h	R
PEL Datenregister	Video DAC			3C9h	RW
PEL Maskenregister	Video DAC			3C6h	RW

*) PEL bedeutet Pixel-Element

Tabelle 10.2: VGA-Register

Eine Auflistung der einzelnen Modi und deren Eigenschaften, welche über den Befehl 10h gesetzt werden können:

Modus	Typ	Reihen	Spalten	Auflösung	Farben
0	Zeichen	25	40	320x200	16
1	Zeichen	25	40	320x200	16
2	Zeichen	25	80	640x200	16
3	Zeichen	25	80	640x200	16
4	Grafik	25	40	320x200	4
5	Grafik	25	40	320x300	4
6	Grafik	25	80	640x200	2
7	Zeichen	25	80	720x350	Mono
13	Grafik	25	40	320x200	16
14	Grafik	25	80	640x200	16
15	Grafik	25	80	640x350	Mono
16	Grafik	25	80	640x350	16
17	Grafik	30	80	640x480	2
18	Grafik	30	80	640x480	16
19	Grafik	25	40	320x200	256

Tabelle 10.3: VGA-Bildschirmmodi

Zeichen die gerade dargestellt werden, können noch zusätzliche Attribute habe. Diese Attribute kennzeichnen die Hinter- und Vordergrundfarbe und lassen den Text blinken:

Zeichenattribute bei Textdarstellung (Modi 0-3)

7	6	5	4	3	2	1	0	Funktion	zulässige Werte
●								blinken	0 = nicht blinkend, 1 = blinkend
	●	●	●					Hintergrund	000 = Schwarz 001 = Blau 010 = Grün 011 = Türkis(Cyan) 100 = Rot 101 = Purpur (Magenta) 110 = Braun 111 = Weiß
				●				Helligkeit	0 = normal, 1 = erhöht
					●	●	●	Vordergrund	000 = Schwarz/Grau*) 001 = Blau/Hellblau 010 = Grün/Hellgrün 011 = Türkis/Helles Türkis 100 = Rot/Rosa 101 = Purpur/Purpurrosa 110 = Braun/Gelb 111 = Weiß/Strahlendweiß

Tabelle 10.4: VGA-Zeichenattribute-1

*) Falls Helligkeit erhöht, gilt die Angabe hinter dem Strich

Zeichenattribute bei Textdarstellung (Modus 7)

7	6	5	4	3	2	1	0	Funktion	zulässige Werte
●								Blinken	0 = nicht blinkend, 1 = blinkend
	●	●	●					Hintergrund	000 = Schwarz 111 = Weiß
				●				Helligkeit	0 = normal, 1 = erhöht
					●	●	●	Vordergrund	000 = Schwarz/Grau*) 001 = Unterstichen 111 = Weiß/Strahlendweiß

Tabelle 10.5: VGA-Zeichenattribute-2

*) Falls Helligkeit erhöht, gilt die Angabe hinter dem Strich

Zeichenattribute bei Grafikdarstellung (Modus 15)

Pixelebene				
3	2	1	0	Funktion
	0		0	schwarzes Zeichen
	0		1	weißes Zeichen
	1		0	blinkendes weißes Zeichen
	1		1	helle weiße Zeichen

Tabelle 10.6: VGA-Zeichenattribute-3

Zeichenattribute bei Grafikdarstellung (Modi 13, 14 und 16)

Pixelebene				
3	2	1	0	Funktion
●				blaue Pixelkomponente
	●			grüne Pixelkomponente
		●		rote Pixelkomponente
			●	Helligkeitskomponente

Tabelle 10.7: VGA-Zeichenattribute-4

Die folgenden Tabellen geben die Speichernutzung bei Text- und Grafikdarstellung an.

Textdarstellung (Modi 0-3):

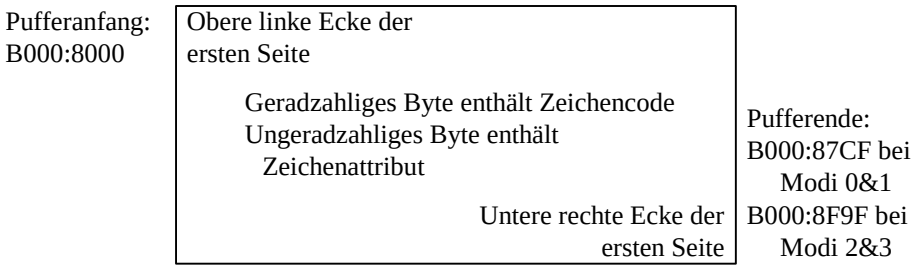


Abbildung 10.1: VGA-Speichernutzung-1

Bis zu 8 aufeinander folgende Seiten in den Modi 0 und 1,
Bis zu 4 aufeinander folgende Seiten in den Modi 2 und 3.

Grafikdarstellung mittlere Auflösung (Modi 4 und 5, 320x200):

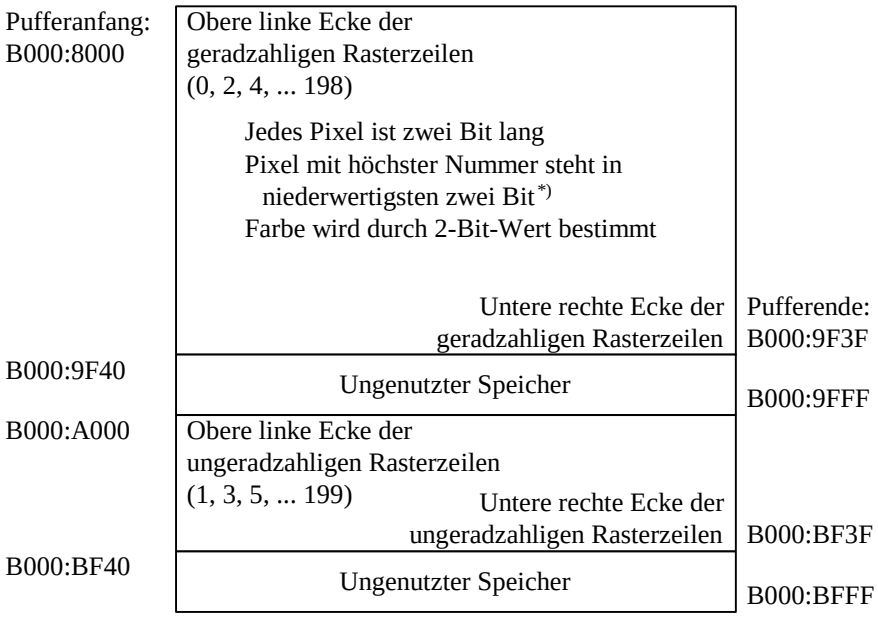


Abbildung 10.2: VGA-Speichernutzung-2

Grafikdarstellung hohe Auflösung (Modus 6, 640x200):

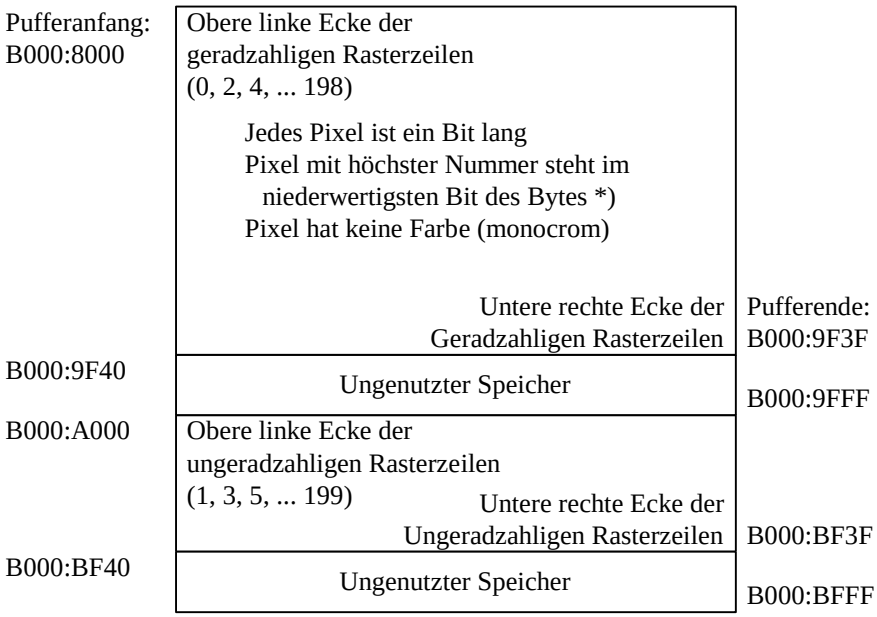


Abbildung 10.3: VGA-Speichernutzung-3

Grafikdarstellung mittlere Auflösung (Modus 13, 340x200):

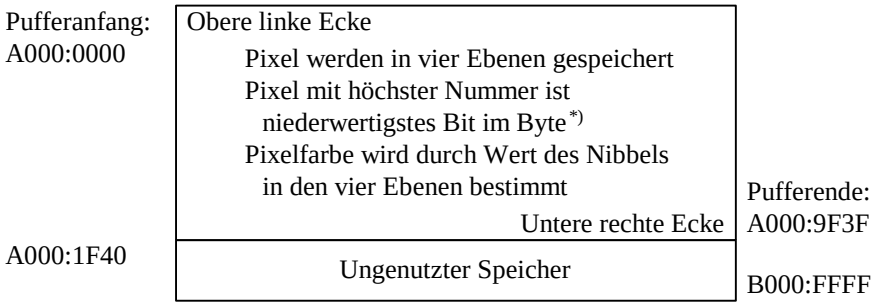


Abbildung 10.4: VGA-Speichernutzung-4

Grafikdarstellung hohe Auflösung (Modus 14, 640x200):

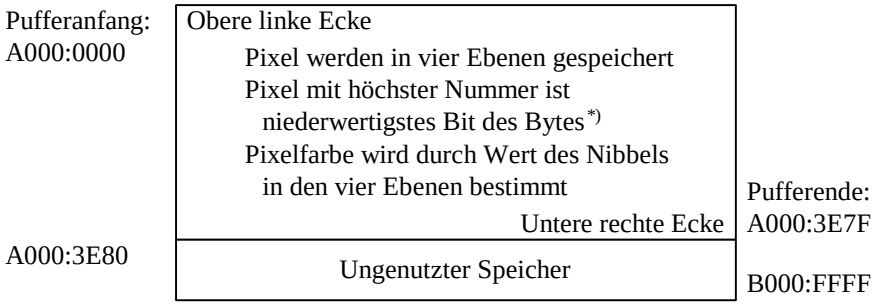


Abbildung 10.5: VGA-Speichernutzung-5

Grafikdarstellung hohe Auflösung (Modus 16, 640x350):

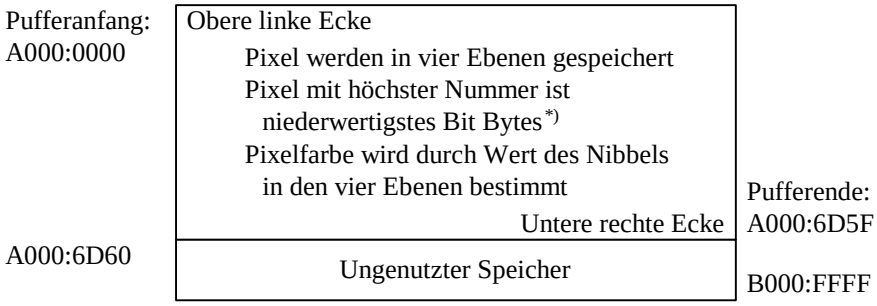


Abbildung 10.6: VGA-Speichernutzung-6

Grafikdarstellung hohe Auflösung (Modus 15, 640x350):

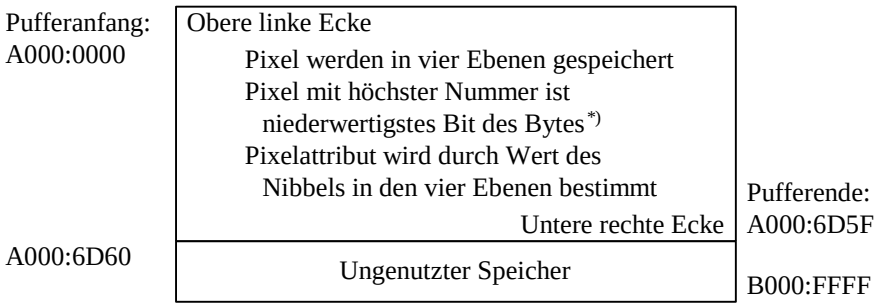


Abbildung 10.7: VGA-Speichernutzung-7

Grafikdarstellung hohe Auflösung (Modus 17, 640x480):

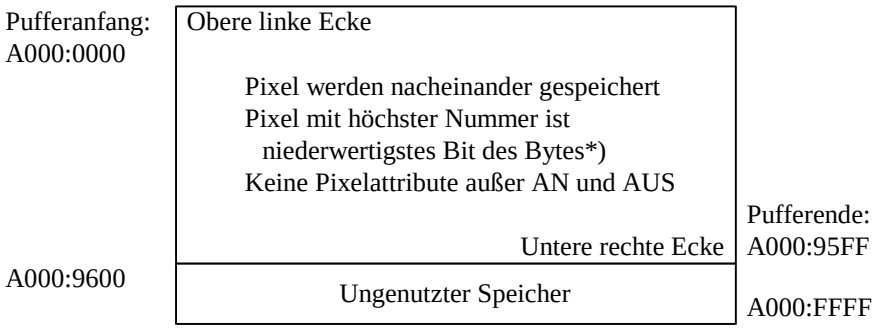


Abbildung 10.8: VGA-Speichernutzung-8

Grafikdarstellung hohe Auflösung (Modus 18, 640x480):

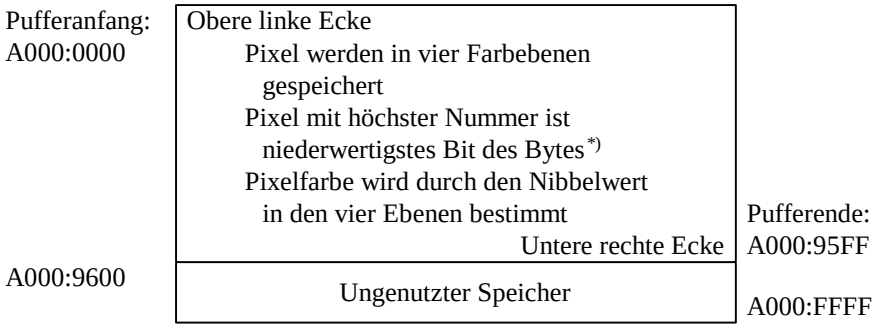


Abbildung 10.9: VGA-Speichernutzung-9

Grafikdarstellung mittlere Auflösung (Modus 19, 320x200):

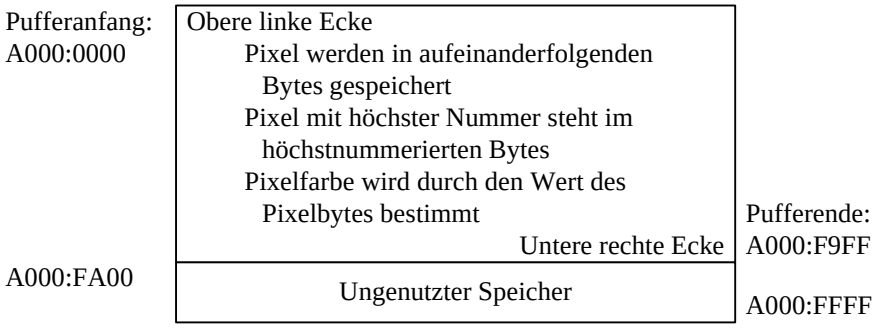


Abbildung 10.10: VGA-Speichernutzung-10

*)Mit anderen Worten: das Pixel mit der höchsten Nummer steht in dem (den) niederwertigsten Bit(s), das Pixel mit der niedrigsten Nummer steht in dem (den) höchstwertigen Bit(s). Beispiel:

-Das erste Byte bei mittlerer Auflösung:

Bitnummer	7 6	5 4	3 2	1 0
Pixelnummer	1	2	3	4

-Das erste Byte bei hoher Auflösung:

Bitnummer	7 6	5 4	3 2	1 0
Pixelnummer	1 2	3 4	5 6	7 8

Falls eine getreue Darstellung der Zeichen gewünscht wird, helfen die beiden nachfolgenden Abbildungen.

VGA-Zeichenmatrix Modi 0-3:

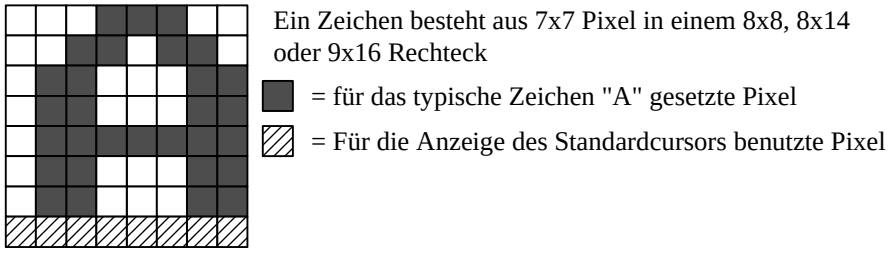
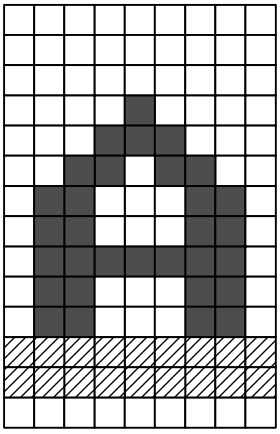


Abbildung 10.11: VGA-Zeichenmatrix-1

VGA-Zeichenmatrix Modi 7 und 15:



Ein Zeichen besteht aus 7x7 Pixel in einem 9x14 oder 9x16 Rechteck

= für das typische Zeichen "A" gesetzte Pixel

= Für die Anzeige des Standardcursors benutzte Pixel

Abbildung 10.12: VGA-Zeichenmatrix-2

Am Ende soll noch eine kurze Referenz der einzelnen Interruptroutinen gegeben werden. Diese werden über Int10h aufgerufen.

Int10h, Funktion 00h – Setzen des Videomodus
Diese Funktion wählt den aktuellen Videomodus und initialisiert ihn. Hierbei wird der Inhalt ebenfalls gelöscht. Über diese Funktion lässt sich auch der Bildschirm löschen, indem der alte Modus einfach beibehalten wird.

Register	Übergabewert	Rückgabewert
AH	00h	-
AL	Videomodus, siehe Tabelle 11.3	-
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 01h – Erscheinungsbild des Cursors definieren
Mit dieser Funktion werden die Größe und Attribute wie Blinken des Cursors gewählt. Da dies hauptsächlich nur die grafische Darstellung betrifft, wird nicht näher darauf eingegangen.

Register	Übergabewert	Rückgabewert
AH	01h	-
CH	Parameter für Cursor	unverändert
CL	Parameter für Cursor	unverändert
SI,DI	-	wurden intern verändert
BX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 02h – Positionierung des Cursors
Der Cursor wird an die angegebene Position gesetzt. Der blinkende Cursor wird nur versetzt, wenn es die aktuelle Bildschirmseite ist. Um den Cursor verschwinden zu lassen, wird dessen Position einfach auf eine nicht mehr darstellbare Zeile gesetzt.

Register	Übergabewert	Rückgabewert
AH	02h	-
BH	Nummer der Bildschirmseite	unverändert
DH	Bildschirmzeile (0-24)	unverändert
DL	Bildschirmspalte (0-39/79)	unverändert
SI,DI	-	wurden intern verändert
BX, CX, SS, CS, DS	-	unverändert

Int10h, Funktion 03h – Position des Cursors auslesen
Die aktuelle Position des Cursors und die Start- und Endzeile des blinkenden Bildschirmcursors werden ausgelesen. Alle Angaben beziehen sich auf die angegebene Bildschirmseite und dort auf das Textkoordinatensystem.

Register	Übergabewert	Rückgabewert
AH	03h	-
BH	Nummer der Bildschirmseite	übergebener Wert
DH	-	Bildschirmzeile des Cursors
DL	-	Bildschirmpalte des Cursors
CH	-	Anfangszeile des blinkenden Bildschirmcursors
CL	-	Endzeile des blinkenden Bildschirmcursors
SI,DI	-	wurden intern verändert
BX, SS, CS, DS	-	unverändert

Int10h, Funktion 04h – Position des Lichtstiftes (LightPen) auslesen
Diese Funktion wird von VGA nicht unterstützt.

Int10h, Funktion 05h – Auswahl der aktuellen Bildschirmseite
Falls der Textmodus aktiv ist, wird die aktuelle Bildschirmseite gewählt und dargestellt. Der blinkende Cursor wird an die Position des Textcursors gesetzt. Der Inhalt der Seite wird nicht verändert. Eine Bildschirmseite muss nicht aktiv sein, um Text darin schreiben zu können.

Register	Übergabewert	Rückgabewert
AH	05h	-
AL	Nummer der Bildschirmseite	-
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 06h – Textzeile nach oben schieben (scrollen)
Der Inhalt der aktuellen Bildschirmseite wird um AL nach oben geschoben. Die Angabe von 00h löscht die komplette Seite. Hierbei werden Leerzeichen (ASCII-Code 32) verwendet. Die herausgeschobenen Zeilen sind unwiderruflich gelöscht. Um den Bildschirm zu löschen, sollte besser die Funktion 00h verwendet werden.

Register	Übergabewert	Rückgabewert
AH	06h	-
AL	Anzahl der Zeilen, um die das Fenster nach oben geschoben werden soll, 00h = Löschen	-
CH	Bildschirmzeile der oberen linken Fensterecke	unverändert
CL	Bildschirmspalte der oberen linken Fensterecke	unverändert
DH	Bildschirmzeile der unteren rechten Fensterecke	unverändert
DL	Bildschirmspalte der unteren rechten Fensterecke	unverändert
BH	Farbe/Attribut für Leerzeilen	unverändert
SI,DI	-	wurden intern verändert
BX, SS, CS, DS	-	unverändert

Int10h, Funktion 07h – Textzeile nach unten schieben (scrollen)
Der Inhalt der aktuellen Bildschirmseite wird um AL nach unten geschoben. Die Angabe von 00h löscht die komplette Seite. Hierbei werden Leerzeichen (ASCII-Code 32) verwendet. Die herausgeschobenen Zeilen sind unwiderruflich gelöscht. Um den Bildschirm zu löschen, sollte besser die Funktion 00h verwendet werden.

Register	Übergabewert	Rückgabewert
AH	07h	-
AL	Anzahl der Zeilen, um die das Fenster nach unten geschoben werden soll, 00h = Löschen	-
CH	Bildschirmzeile der oberen linken Fensterecke	unverändert
CL	Bildschirmspalte der oberen linken Fensterecke	unverändert
DH	Bildschirmzeile der unteren rechten Fensterecke	unverändert
DL	Bildschirmspalte der unteren rechten Fensterecke	unverändert
BH	Farbe/Attribut für Leerzeilen	unverändert
SI,DI	-	wurden intern verändert
BX, SS, CS, DS	-	unverändert

Int10h, Funktion 08h – Auslesen eines Zeichens/Farbe
Der ASCII-Code des Zeichens an der aktuellen Cursorposition und dessen Farbe/Attribut wird ausgelesen. Diese Funktion kann auch im Grafikmodus aufgerufen werden, funktioniert dann jedoch etwas anders.

Register	Übergabewert	Rückgabewert
AH	08h	Farbe/Attribut
AL	-	ASCII-Code
BH	Nummer Bildschirmseite	-
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 09h – Schreiben eines Zeichens/Farbe
Der ASCII-Code und die Farbe/Attribute werden an der aktuellen Cursorposition geschrieben. Steuercodes (Bell, Carrige Return, Tabulator) werden nicht als solche erkannt, anstatt dessen wird deren Code dargestellt, sofern dieser existiert. Diese Funktion kann auch im Grafikmodus aufgerufen werden, funktioniert dann jedoch etwas anders. Es muss darauf geachtet werden, dass alle Zeichen in die aktuelle Zeile passen.

Register	Übergabewert	Rückgabewert
AH	09h	-
AL	ASCII-Code	-
BH	Nummer Bildschirmseite	unverändert
BL	Farbe/Attribut	unverändert
CX	Anzahl, wie oft das Zeichen hintereinander geschrieben werden soll	unverändert
SI,DI	-	wurden intern verändert
DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Ah – Schreiben eines Zeichens
Der ASCII-Code wird an der aktuellen Cursorposition geschrieben. Steuercodes (Bell, Carrige Return, Tabulator) werden nicht als solche erkannt, statt dessen wird deren Code dargestellt, sofern dieser existiert. Die Farbe/Attribute bleiben unverändert. Diese Funktion kann auch im Grafikmodus aufgerufen werden, funktioniert dann jedoch etwas anders. Es muss darauf geachtet werden, dass alle Zeichen in die aktuelle Zeile passen. Der Cursor wird auf die nächste Bildschirmposition gesetzt.

Register	Übergabewert	Rückgabewert
AH	0Ah	-
AL	ASCII-Code	-
BH	Nummer Bildschirmseite	unverändert
CX	Anzahl, wie oft das Zeichen hintereinander geschrieben werden soll	unverändert
SI,DI	-	wurden intern verändert
BX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Bh, Unterfunktion 0 – Auswahl der Rahmen-/Hintergrundfarbe
Die Rahmen- und Hintergrundfarbe für den Text- beziehungsweise Grafikmodus werden über diese Funktion gewählt. Im Textmodus wird nur die Rahmenfarbe festgelegt, da jedes Zeichen eine eigene Hintergrundfarbe haben kann.

Register	Übergabewert	Rückgabewert
AH	0Bh	-
BH	00h	unverändert
BL	Rahmen-/Hintergrundfarbe (0-15)	unverändert
SI,DI	-	wurden intern verändert
CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Bh, Unterfunktion 1 – Auswahl der Farbpalette
Diese Funktion wird nur im Grafikmodus 320x200 benutzt.
Palette 0: Grün, Rot, Gelb
Palette 1: Cyan, Magenta, Weiß

Register	Übergabewert	Rückgabewert
AH	0Bh	-
BH	01h	unverändert
BL	Farbpalette (0 oder 1)	unverändert
SI,DI	-	wurden intern verändert
CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Ch – Schreiben eines Grafikpunktes
Der Farbwert für einen Bildschirmpunkt wird gesetzt. Im 640x200-Modus sind die Werte 0 und 1 erlaubt. Im 320x200-Modus die Werte 0-3. 0 steht für die Hintergrundfarbe, 1 für die erste Farbe der gewählten Palette, ...

Register	Übergabewert	Rückgabewert
AH	0Ch	-
AL	Farbwert	-
DX	Bildschirmzeile	unverändert
CX	Bildschirmspalte	unverändert
SI,DI	-	wurden intern verändert
BX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Dh – Grafikpunkt lesen
Der Farbwert für einen Bildschirmpunkt wird ausgelesen. Im 640x200-Modus sind die Werte 0 und 1 erlaubt. Im 320x200-Modus die Werte 0-3. 0 steht für die Hintergrundfarbe, 1 für die erste Farbe der gewählten Palette, ...

Register	Übergabewert	Rückgabewert
AH	0Dh	-
AL	-	Farbwert
DX	Bildschirmzeile	unverändert
CX	Bildschirmspalte	unverändert
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Eh – Schreiben eines Zeichens

Ein Zeichen wird an die aktuelle Cursorposition geschrieben. Die Farbe des alten Zeichens bleibt erhalten. Zusätzlich interpretiert diese Funktion den ASCII-Code des Zeichens. Anstelle des Bell-Zeichens wird dann ein Piepston ausgegeben. Nach Ausgabe des Zeichens wird die Cursorposition weitergesetzt. Wenn der Cursor die letzte Bildschirmposition erreicht hat, wird der Bildschirm hochgeschoben(gescrollt). Die Vordergrundfarbe richtet sich nach dem aktuellen Grafikmodus. Im 640x200-Modus sind nur die Werte 0 und 1 erlaubt. Im 320x200-Modus sind nur die Werte 0 bis 3 erlaubt. Siehe auch Funktion 0Ch.

Register	Übergabewert	Rückgabewert
AH	0Eh	-
AL	ASCII-Code	-
BL	Vordergrundfarbe des Zeichen (nur im Grafikmodus)	unverändert
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

Int10h, Funktion 0Fh – Auslesen des Videomodus

Die Nummer des aktuellen Videomodus, die Anzahl der Zeichen pro Zeile und die Nummer der aktuellen Bildschirmseite werden ausgelesen.

Register	Übergabewert	Rückgabewert
AH	0Fh	Anzahl Zeichen pro Zeile
AL	-	Videomodus, siehe Tabellen
BH	Nummer der aktuellen Bildschirmseite	unverändert
SI,DI	-	wurden intern verändert
BX, CX, DX, SS, CS, DS	-	unverändert

10.4 Tastatur-Scancodes

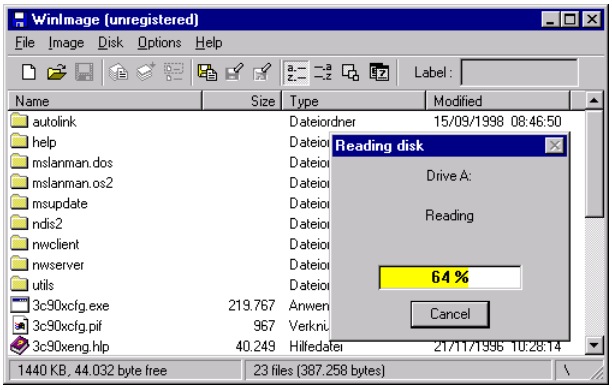
In nachfolgender Tabelle sind die einzelnen Scancodes aufgelistet. Um einen einfachen Vergleich zwischen Set 1, Set 2 und Set 3 zu bekommen, wurden alle in einer einzigen Tabelle dargestellt.

Taste	Set 1		Set 2		Set 3	
	Make	Break	Make	Break	Make	Break
A	1E	9E	1C	F0,1C	1C	F0,1C
B	30	B0	32	F0,32	32	F0,32
C	2E	AE	21	F0,21	21	F0,21
D	20	A0	23	F0,23	23	F0,23
E	12	92	24	F0,24	24	F0,24
F	21	A1	2B	F0,2B	2B	F0,2B
G	22	A2	34	F0,34	34	F0,34
H	23	A3	33	F0,33	33	F0,33
I	17	97	43	F0,43	43	F0,48
J	24	A4	3B	F0,3B	3B	F0,3B
K	25	A5	42	F0,42	42	F0,42
L	26	A6	4B	F0,4B	4B	F0,4B
M	32	B2	3A	F0,3A	3A	F0,3A
N	31	B1	31	F0,31	31	F0,31
O	18	98	44	F0,44	44	F0,44
P	19	99	4D	F0,4D	4D	F0,4D
Q	10	19	15	F0,15	15	F0,15
R	13	93	2D	F0,2D	2D	F0,2D
S	1F	9F	1B	F0,1B	1B	F0,1B
T	14	94	2C	F0,2C	2C	F0,2C
U	16	96	3C	F0,3C	3C	F0,3C
V	2F	AF	2A	F0,2A	2A	F0,2A
W	11	91	1D	F0,1D	1D	F0,1D
X	2D	AD	22	F0,22	22	F0,22
Y	15	95	35	F0,35	35	F0,35
Z	2C	AC	1A	F0,1A	1A	F0,1A
0	0B	8B	45	F0,45	45	F0,45
1	02	82	16	F0,16	16	F0,16
2	03	83	1E	F0,1E	1E	F0,1E
3	04	84	26	F0,26	26	F0,26
4	05	85	25	F0,25	25	F0,25
5	06	86	2E	F0,2E	2E	F0,2E
6	07	87	36	F0,36	36	F0,36
7	08	88	3D	F0,3D	3D	F0,3D
8	09	89	3E	F0,3E	3E	F0,3E
9	0A	8A	46	F0,46	46	F0,46
`	29	89	0E	F0,0E	0E	F0,0E
-	0C	8C	4E	F0,4E	4E	F0,4E
=	0D	8D	55	FO,55	55	F0,55
\	2B	AB	5D	F0,5D	5C	F0,5C
BKSP	0E	8E	66	F0,66	66	F0,66
SPACE	39	B9	29	F0,29	29	F0,29
TAB 0F	0F	8F	0D	F0,0D	0D	F0,0D
CAPS	3A	BA	58	F0,58	14	F0,14
L SHFT	2A	AA	12	F0,12	12	F0,12
L CTRL	1D	9D	14	F0,14	11	F0,11
L WIN	E0,5B	E0,DB	E0,1F	E0,F0,1F	8B	F0,8B
L ALT	38	B8	11	F0,11	19	F0,19
R SHFT	36	B6	59	F0,59	59	F0,59
R CTRL	E0,1D	E0,9D	E0,14	E0,F0,14	58	F0,58
R WIN	E0,5C	E0,DC	E0,27	E0,F0,27	8C	F0,8C
R ALT	E0,38	E0,B8	E0,11	E0,F0,11	39	F0,39
APPS	E0,5D	E0,DD	E0,2F	E0,F0,2F	8D	F0,8D
ENTER	1C	9C	5A	F0,5A	5A	F0,5A
ESC	01	81	76	F0,76	08	F0,08

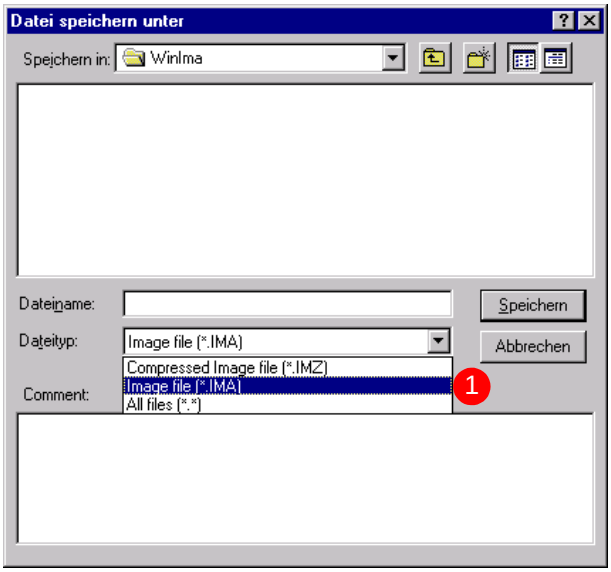
Taste	Set 1		Set 2		Set 3	
	Make	Break	Make	Break	Make	Break
F1	3B	BB	05	F0,05	07	F0,07
F2	3C	BC	06	F0,06	0F	F0,0F
F3	3D	BD	04	F0,04	17	F0,17
F4	3E	BE	0C	F0,0C	1F	F0,1F
F5	3F	BF	03	F0,03	27	F0,27
F6	40	C0	0B	F0,0B	2F	F0,2F
F7	41	C1	83	F0,83	37	F0,37
F8	42	C2	0A	F0,0A	3F	F0,3F
F9	43	C3	01	F0,01	47	F0,47
F10	44	C4	09	F0,09	4F	F0,4F
F11	57	D7	78	F0,78	56	F0,56
F12	58	D8	07	F0,07	5E	F0,5E
PRNT SCRN	E0,2A,E0,37	E0,B7,E0,AA	E0,12,E0,7C	E0,F0,7C,E0, F0,12	57	F0,57
SCROLL	46	C6	7E	F0,7E	5F	F0,5F
PAUSE	E1,1D,45,E1, 9D,C5	-NONE-	E1,14,77,E1, F0,14,F0,77	-NONE-	62	F0,62
[	1A	9A	54	F0,54	54	F0,54
INSERT	E0,52	E0,D2	E0,70	E0,F0,70	67	F0,67
HOME	E0,47	E0,97	E0,6C	E0,F0,6C	6E	F0,6E
PG UP	E0,49	E0,C9	E0,7D	E0,F0,7D	6F	F0,6F
DELETE	E0,53	E0,D3	E0,71	E0,F0,71	64	F0,64
END	E0,4F	E0,CF	E0,69	E0,F0,69	65	F0,65
PG DN	E0,51	E0,D1	E0,7A	E0,F0,7A	6D	F0,6D
U ARROW	E0,48	E0,C8	E0,75	E0,F0,75	63	F0,63
L ARROW	E0,4B	E0,CB	E0,6B	E0,F0,6B	61	F0,61
D ARROW	E0,50	E0,D0	E0,72	E0,F0,72	60	F0,60
R ARROW	E0,4D	E0,CD	E0,74	E0,F0,74	6A	F0,6A
NUM	45	C5	77	F0,77	76	F0,76
KP /	E0,35	E5,B5	E0,4A	E0,F0,4A	4A	F0,4A
KP *	37	B7	7C	F0,7C	7E	F0,7E
KP -	4A	CA	7B	F0,7B	4E	F0,4E
KP +	4E	CE	79	F0,79	7C	F0,7C
KP EN	E0,1C	E0,9C	E0,5A	E0,F0,5A	79	F0,79
KP .	53	D3	71	F0,71	71	F0,71
KP 0	52	D2	70	F0,70	70	F0,70
KP 1	4F	CF	69	F0,69	69	F0,69
KP 2	50	D0	72	F0,72	72	F0,72
KP 3	51	D1	7A	F0,7A	7A	F0,7A
KP 4	4B	CB	6B	F0,6B	6B	F0,6B
KP 5	4C	CC	73	F0,73	73	F0,73
KP 6	4D	CD	74	F0,74	74	F0,74
KP 7	47	C7	6C	F0,6C	6C	F0,6C
KP 8	09	89	75	F0,75	75	F0,75
KP 9	49	C9	7D	F0,7D	7D	F0,7D
]	1B	9B	5B	F0,5B	5B	F0,5B
;	27	A7	4C	F0,4C	4C	F0,4C
'	28	A8	52	F0,52	52	F0,52
,	33	B3	41	F0,41	41	F0,41
.	34	B4	49	F0,49	49	F0,49
/	35	B5	4A	F0,4A	4A	F0,4A

10.5 Rohdatenerzeugung

Um ein Image einer Diskette zu erhalten, von dem Daten gelesen werden können, müssen folgende Schritte ausgeführt werden:
Mit dem Programm WinImage muss zuerst ein komplettes Abbild einer Diskette erzeugt werden. Dazu wird zunächst ein neues Projekt über „File -> New“ angewählt. Anschließend kann eine Diskette mit „Disk->ReadDisk“ eingelesen werden:



Sobald die Diskette erfolgreich in den Hauptspeicher eingelesen ist, müssen die Daten über „File->SaveAs...“ abgespeichert werden:



Hierbei ist strengstens darauf zu achten, dass bei 1 die richtige Option (ImageFile *.IMA) ausgewählt wird, da sonst die Daten komprimiert gespeichert und später nicht weiterverarbeitet werden können.
Das so erhaltenen Image ist ein Abbild der eigentlichen Nutzdaten auf der Diskette. Die Daten sind zwar in der richtigen Reihenfolge abgelegt, jedoch fehlen die Rahmeninformationen zur Emulation wie CRC, IDAM, GAP. Um diese zu erzeugen, wurde ein kleines C-Programm unter Windows geschrieben, das die fehlenden Informationen erzeugt. Die Bedienung ist sehr einfach:
Über die Befehlszeile wird das Programm „Image2Floppy <Quelle> <Ziel>“ mit dem Quellimage (Image, das von WinImage erzeugt wurde) und dem Zielimage (Image, das später zur Emulation benutzt wird) aufgerufen. Hierbei wächst die Größe des Images von 1.44 MB auf 2MB an:

```
c:>image2floppy boot.img boot.dat
DiskImage created successfully
c:>
```

11 Literaturverzeichnis

Verwendete Bücher:

VHDL:

- [V1] The Designer's Guide to VHDL – Peter J. Ashenden – Morgan Kaufmann Publishers, Inc. – 1-55860-270-4
- [V2] The VHDL Reference – Ulrich Heinkel et. al. – Wiley – ISBN 0-471-89972-0

PC-Technik:

- [T1] PC Hardwarebuch – Hans-Peter Messmer – Addison Wesley – ISBN 3-8273-1461-5
- [T2] PC Intern 2– Tischer – Data Becker – ISBN 3-89011-331-1
- [T3] Die PC-Referenz für Programmierer, Thom Hogan – Microsoft Press
- [T4] The Programmer's Technical Reference – Williams D. – Sigma Press – 9781850581994
- [T5] PCI System Architecture – MindShare, Inc.(Tom Shanley, Don Anderson) – Addison Wesley – 0-201-30974-2

Programmieren:

- [P1] PC-Programmierung in Maschinensprache – Peter Monadjemi – Markt&Technik – ISBN 3-89090-957-4
- [P2] Programmiererhandbuch MS-DOS – Ray Duncan – Microsoft Press/Vieweg – ISBN 3-528-04631-7
- [P3] Programmieren in C – Brian W. Kernighan, Dennis M. Ritchie – Hanser – ISBN 3-446-15497-3
- [P4] Java2 – Laura Lemay, Rogers Cadenhead – Markt&Technik – ISBN 3-8272-5578-3
- [P5] Linux Device Drivers – Alessandro Rubini – O'Reilly – ISBN 1-55592-292-1

Literatur aus dem Internet kann auf folgenden Seiten gefunden werden:

- http://www.opencores.net
  Wishbone Interface
- http://www.hardwarebook.net
  Dokumentation über Hardware
- http://www.openhardware.net
  Nachbauten des µCsimm's
- http://cbothamy.free.fr/projects/vgabios
  das eigentliche VGA-Bios
- http://www.erd.epson.com/vdc/html/products.htm
  VGA-BIOS von Epson (Assembler)
- http://www.dosemu.org
  Dosemulationsprogramme für Unix (Tools86) darauf basiert das VGA-BIOS
- http://www.beyondlogic.org
  Dokumentation über µCsimm und PS/2
- http://www.chipdir.org
  Dokumentation über Bauteile
- http://www.storagereview.com
  Speichermedien wie Disketten
- http://uclinux.home.at
  Linkseite für µClinux und zugehörige Hardware
- http://uclinux.org
  Homepage vom µCsimm
- http://govschl.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm
  Seite über PS/2
- http://lxr.linux.no
  Browsbarer Linux-Source-Code
- http://oreilly.com
  Quelltexte zum Buch von A. Rubini
- http://www.funk.com
  Programm zum Fernsteuern von Windows Rechnern
- http://java.sun.com
  Homepage von Java (hier kann auch das JDK kostenlos heruntergeladen werden)
- http://home1.pacific.net.sg/~ttn/chpt8/chpt8.html
  Diskettenaufbau LowLevel
- http://www.winimage.com
  WinImage / Homepage

Weiter hilfreiche Dokumentation im Internet:

- http://www.calsoft.co.in/tech/modified.html
  Java und TCP-Praogrammierung
- http://www.openhardware.net/cgi-bin/cvsweb.cgi
  µClinux Mainpage
- http://gcc.gnu.org/onlinedocs/gcc_toc.html
  Dokumentation über GCC
- http://www.gweep.net/~professor/tuxbot
  Dokumentation über Interrupthandler
- http://home.worldonline.dk/~finth
  Dokumentation über VGA

Index

Erklärung:

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den ....
Unterschrift