Diplomarbeit

Implementierung und Test einer Emulationsplattform für die Hardware- Softwarepartitionierung eingebetteter Systeme

Carsten Nitsch

Universität Leipzig
Institut für Informatik
Prof. Dr. U. Kebschull

Forschungszentrum Informatik
Sytementwurf in der Mikroelektronik
Prof. Dr. W. Rosenstiel

Betreuer: Dipl.-Ing. K.Weiß

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die aufgeführten Hilfsmittel verwendet habe.

Karlsruhe, den 23.05.1999

.....
Carsten Nitsch

Inhaltsverzeichnis

Kapitel 1 Einleitung 13

1.1 Eingebettete Systeme 14

Kapitel 2 Motivation 15

2.1 Entwurfsprobleme 15
2.1.1 Eingebettete Systeme mit 4-Bit-Mikrocontrollern 15
2.1.2 Eingebettete Systeme mit 8-Bit-Mikrocontrollern 15
2.1.3 Eingebettete Systeme mit 16-Bit-Mikrocontrollern 16
2.1.4 Eingebettete Systeme mit 32-Bit-Mikrocontrollern 16
2.1.5 Fazit 17
2.2 Inhalte dieser Arbeit 19
2.2.1 Erhöhung der Produktqualität 19
2.2.2 Weiterentwicklung zur Forschungsplattform 20

Kapitel 3 Grundlagen 21

3.1 Echtzeitbetriebsysteme 21
3.2 Programmierbare Logikbausteine 24
3.2.1 Complex Programmable Logic Devices (CPLD) 24
3.2.2 SRAM basierte FPGA 26
3.2.2.1 Die XC-4000 Serie 26
3.2.2.2 Virtex 33
3.2.3 Antifuse basierte FPGA (Actel) 35
3.3 Speicherbausteine 37
3.3.1 EEPROMs 37
3.4 Rekonfigurationsmethoden für FPGA 39
3.4.1 Compile Time Rekonfiguration (CTR) 39
3.4.2 Run Time Reconfiguration (RTR) 39

Kapitel 4 Entwicklungsumgebung 43

4.1 Die Emulationsumgebung Spyder-Core-P1 43
4.2 Entwicklungsumgebung SDS 45
4.3 Echtzeitbetriebsystem RTEMS 46
4.4 Echtzeitbetriebsystem VxWorks 47

Kapitel 5 Implementierung 49

5.1 Diagnosesoftware 49
5.1.1 Grundtest 1 50

5.1.2 Grundtest 2 ... 52
5.1.3 Grundtest 3 ... 54
5.1.4 DRAM Test ... 56
5.1.5 DRAM Test mit DPRAM-Interface ... 58
5.1.6 Test des On-Chip-UART ... 61
5.1.7 Test der zweiten seriellen Schnittstelle ... 63
5.1.8 Ethernet Interface ... 65
5.1.9 Vollständiger Systemtest ... 67
5.2 Bewertung des Laufzeitverhaltens eines eingebetteten Systems ... 69
5.2.1 Loader ... 70
5.2.2 Meßumgebung ... 72
5.2.2.1 Benchmark für Interrupt Reaktionszeit ... 72
5.2.2.2 Benchmark für Task Umschaltzeit ... 74
5.2.3 Meßwerte ... 76
5.2.4 Zusammenfassung ... 77
5.3 Compile Time Rekonfiguration ... 78
5.3.1 Funktionsprinzip ... 78
5.3.2 Programmieralgorithmus für ATMEL EEPROMs ... 79
5.3.3 Programmierung von Atmel EEPROMS auf Spyder-Core ... 81
5.3.3.1 Dateiformate ... 81
5.3.3.2 Programmierung ... 82
5.4 Run Time Rekonfiguration ... 84
5.4.1 DOS Filesystem für Flash-RAM ... 85
5.4.2 DOS Filesystem für DRAM (RAM-Drive) ... 89
5.4.3 Gerätetreiber für XC4000 FPGA ... 90
5.4.3.1 Treiberintegration unter VxWorks ... 90
5.4.3.2 Umsetzung des Konzepts für Spyder-Core-P1 ... 91
5.5 Demonstrator: Erweiterer FTP-Server ... 94
5.5.1 Konzept ... 94
5.5.2 Realisierung ... 95

Kapitel 6 Zusammenfassung und Ausblick 97

Anhang A Referenzen 99

Anhang B Loader 101

Anhang C Benchmarks 119

Messung der Interrupt-Umschaltzeit ... 119
Messung der Task-Umschaltzeiten ... 130

Anhang D Ladeprogramm für EEPROM 133

Anhang E Testprogramme 155

Grundtest 1 155
Grundtest 2 157
Grundtest 3 162
DRAM Test 165
DRAM Test mit DPRAM-Interface 170
Test des On Chip UARTs 178
Test der zweiten seriellen Schnittstelle 186
Test der Ethernetschnittstelle 190

Anhang F Treiber 193

Treiber für Flash-RAM 193
Treiber für RAM Drive 200
Treiber für Xilinx FPGA XC-4000 202

Anhang G FTP Server 208

Abbildungsverzeichnis

Abbildung 2.1 Qualitativer Anstieg der Entwurfsproblematik bei eingebetteten Systemen (Quelle: [KW99]) 17

Abbildung 3.1 Xilinx XC9500 CPLD-Architektur ([Xil98/1]) 24

Abbildung 3.2 Xilinx XC4000 konfigurierbarer Logikblock (Quelle: [Xil98/1]) 27

Abbildung 3.3 Xilinx XC4000 Dual-Port-SRAM Block (Quelle:[Xil98/1]) 28

Abbildung 3.4 Xilinx XC4000 Ein- Ausgabeblock (Quelle: [Xil98/1]) 30

Abbildung 3.5 High Level Routing Diagramm für XC4000 Serie (Quelle: [Xil98/1]) ..31

Abbildung 3.6 Switchmatrizen (Quelle: [Xil98/1]) 32

Abbildung 3.7 Virtex Architektur (Quelle: [Xil99]) 33

Abbildung 3.8 2-Slice Virtex CLB (Quelle: [Xil99]) 34

Abbildung 3.9 ACT 1 Device (Quelle: [Act94]) 35

Abbildung 3.10 ACT-1 Logikblock 36

Abbildung 3.11 EEPROM Speichertransistor (Quelle: [HM93]) 37

Abbildung 3.12 Compile Time Rekonfiguration([HuWi95]) 39

Abbildung 3.13 Run Time Rekonfiguration ([HuWi95]) 39

Abbildung 3.14 Globale Run-Time Rekonfiguration ([HuWi95]) 40

Abbildung 3.15 Lokale Run-Time Rekonfiguration ([HuWi95]) 41

Abbildung 4.1 Blockschaltbild von Spyder-Core-P1 (Quelle: [KW99]) 44

Abbildung 4.2 SDS Monitor Software 45

Abbildung 4.3 RTEMS Architektur (Quelle: [Army96]) 46

Abbildung 4.4 Entwicklungsumgebung für VxWorks (Quelle: [KW99]) 47

Abbildung 5.1 Busaktivitäten 52

Abbildung 5.2 Spyder-Core-P1 mit LEDs 54

Abbildung 5.3 Grundtest 3 55

Abbildung 5.4 Fehler auf Datenbus 57

Abbildung 5.5 Dual Port RAM Interface 58

Abbildung 5.6 Speicherauszug durch DOS-Programm Debug 60

Abbildung 5.7 Schnittstellentest (Pseudocode) 61

Abbildung 5.8 Loop-Back-Beschaltung einer 9-PIN Sub-D Buchse 62

Abbildung 5.9 Externe serielle Schnittstelle als Bestandteil der I/O Bank 63
Abbildung 5.10 Loop-Back Beschaltung für den Test der externen seriellen Schnittstelle . 64
Abbildung 5.11 MB86960 ([Fu97]) 65
Abbildung 5.12 Testalgorithmus für den Test des Ethernet Interfaces 66
Abbildung 5.13 Nutzung des Dual-Port-RAM durch den Loader 70
Abbildung 5.14 Ermitteln der Interrupt Reaktionszeit 72
Abbildung 5.15 Cachekonfiguration im Betriebsystem RTEMS 73
Abbildung 5.16 Messen der Taskumschaltzeiten 74
Abbildung 5.17 Messen der Taskumschaltzeiten 2 75
Abbildung 5.18 Datenfluß beim Laden des seriellen EEPROM 78
Abbildung 5.19 Bitformate 79
Abbildung 5.20 Messageformat für die Programmierung eines AT17Cx EEPROM (Quelle: [Atmel1]) 80
Abbildung 5.21 Device Address Byte (Quelle: [Atmel1]) 80
Abbildung 5.22 Address Bytes (Quelle: [Atmel1]) 81
Abbildung 5.23 Xilinx Bitstream Format 81
Abbildung 5.24 Intel MCS Format 82
Abbildung 5.25 Programmierung eines EEPROM auf Spyder-Core-P1 82
Abbildung 5.26 Run Time Rekonfiguration auf Spyder-Core-P1 84
Abbildung 5.27 Non-Block Devices vs. Block Devices (Quelle:[Wind97]) 85
Abbildung 5.28 Flash-RAM als Laufwerk 86
Abbildung 5.29 Erstellen einer Parametertabelle für Block Device Treiber 87
Abbildung 5.30 Formatieren und automatisch mounten 87
Abbildung 5.31 Mounten eines existierenden Filesystems 88
Abbildung 5.32 Erzeugen und Mounten einer RAM-Disk 89
Abbildung 5.33 Treiberinitialisierung für Non-Block Devices (Quelle:[Wind97]) 90
Abbildung 5.34 Hinzufügen von Geräten (Quelle:[Wind97]) 91
Abbildung 5.35 Hardware-Initialisierungssequenz der Funktion Open() 92
Abbildung 5.36 Datenübertragung 92

Abbildung 5.37 Integration des Spyder-Core Systems in die Benutzeroberfläche des Hostrechners 94
Abbildung 5.38 Assoziation zwischen Verzeichnissen und Gerätetreibern 95
Abbildung 6.1 Performance von Spyder Core P1 bei der Rekonfiguration des FPGA ..98

Tabellenverzeichnis

Klassische versus Echtzeitsysteme ([MS97]) 22
Look Up Table 27
Unterstützte RAM Modes (Quelle:[Xil98/1]) 29
Fetching 51
Dual Port ROM Interface: Adressen 59
Parameter für Schnittstellentest (On-Chip-UART) 61
Parameter für Schnittstellentest (externer UART) 63
Konfiguration des MB86960 65
Taskumschaltzeiten für PPC403 GA (40 Mhz) 76
Taskumschaltzeiten für PPC 403GCX (40 Mhz) 76
Interruptreaktionszeiten für PPC 403GA (40 Mhz) 76
Interruptreaktionszeiten für PPC 403GCX (40 Mhz) 76
Block Device Parameter unter VxWorks 86
Parameter für RAM-Drive 89
Signale und Adresse für Spyder-Core-P1 93

Kapitel 1

Einleitung

Vor über 40 Jahren galten Computer noch als etwas Besonders. Riesige Rechenmonster wie die UNIVAC füllten ganze Häuser mit lärmenden Lochkartenlesern, Relais, Fernschreibern und ähnlichem Equipment. Wer zu dieser Zeit eine Rechenmaschine betrieb, brauchte viel Platz und eine leistungsfähige Energieversorgung.

Kaum jemand glaubte ersthaft daran, daß die Welt mehr als eine wenige dieser Wunderwerke benötigen wird. Heute wissen wird, daß diese Prognose wahrscheinlich der größte Irrtum der Zukunftsforschung war. Computer sind aus unserem Alltag nicht mehr wegzudenken. Sie sind in der Arbeitswelt wie auch in Privathaushalten weit verbreitet. Ein stetiger Preisverfall sorgt dafür, daß die Rechentechnik zum Alltagsgut wird. Nach dem Gesetz von Moore verdoppelt sich die Anzahl der Transistoren pro Chip aller 18 Monate. Momentan sind Taktfrequenzen von 400-600 Mhz üblich. In den Labors der Chiphersteller laufen bereits Prototypen mit einem Takt von 1 GHz. Die Leistungsfähigkeit der Computer steigt immer stärker und damit auch die Anzahl der potentiellen Einsatzgebiete.

Im Gegensatz zu Desktopcomputern verlief die größte technische Revolution unbemerkt. Schauen wir uns einmal in unserer Wohnung um. Wieviele Mikroprozessoren besitzen wir? Sind es fünf, zehn oder etwa doch viel mehr? Den wenigsten Menschen ist bewußt, daß in Videorecordern, ISDN-Anlagen, Mobiltelefonen, Küchengeräten, Stereoanlagen, Waschmaschinen und Autos Computer „versteckt“ sind. Selbst auf Smartcards verrichten diese Systeme unauffällig ihren Dienst.

Diese Systeme müssen oft ähnlich komplexe Aufgaben lösen wie Desktopcomputer, unterliegen im Gegensatz zu diesen jedoch weitaus strengeren Restriktionen in Bezug auf Platzbedarf und Kosten. Daher werden im Bereich der integrierten Funktionseinheiten die für den Betrieb unbedingt benötigten Funktionseinheiten auf einem Chip integriert, z.B. serielle Schnittstellen, ISDN Interfaces oder Speicherkontroller.

1.1 Eingebettete Systeme

In der Fachliteratur hat sich für diese integrierten Systeme der Begriff *eingebettetes System* etabliert. Es existieren Definitionen für eingebettete Systeme, wie z.B. in [Wo94] beschrieben. Eine allgemein akzeptierte Definition hat sich aber jedoch noch nicht durchgesetzt. Ich möchte das Definitionsproblem an einem Beispiel veranschaulichen:

- Ein handelsüblicher PC wird mit einem Touch Screen und einem Thermodrucker ausgestattet und in einem stabilen Stahlgehäuse eingeschlossen. Dieses System dient dann z.B. als Informationsterminal auf einem Bahnhof, mit dessen Hilfe man Fahrpläne und Zugverbindungen abfragen kann. Man kann dieses System in den Bereich der allgemeinen Rechentechnik einordnen, immerhin verrichtet Standardhardware die notwendige Arbeit. Andererseits spricht nichts dagegen, das Informationsterminal als eingebettetes System zu betrachten, da es sich um spezialisierte Hardware handelt.

- Es gibt spezielle Smartcards, welche z.B. in Hotels eingesetzt werden. Mit Hilfe dieser Smartcards wird eine Zugangskontrolle realisiert, sie dienen zum Bezahlen, Telefonieren usw. Einige haben sogar ein integriertes Display. Dieses Smartcards könnte man sowohl als eingebettetes System als auch als *Application Specific Integrated Processor (ASIP)* betrachten, da alle Funktionen in einem Chip realiisert sind..

Ein wesentliches Kriterium von eingebetteten Systemen ist, daß sie für spezielle Anwendungen entwickelt wurden. Auf den Systemen wird man einen Mikrocontroller finden. Auf diesen Mikrocontrollern läuft eine spezielle Software, welche einen bestimmten Teil der Systemfunktionen erfüllt. Funktionen, die sich nicht in Software realisieren lassen, erfordern die Bereitstellung bestimmter Hardware.

In [KW99] wird ein eingebettetes System wie folgt definiert:

Definition 1.1: Ein eingebettetes System besteht aus einem anwendungsspezifischen Softwareteil, der auf einem Mikrocontroller abgearbeitet wird und aus einem anwendungsspezifischen Hardwareteil. Beide Teile kommunizieren untereinander und mit der Umgebung. Insbesondere die Umgebung gibt bestimmte Randbedingungen vor, die von dem eingebetteten System eingehalten werden müssen.

Kapitel 2

Motivation

2.1 Entwurfsprobleme

In eingebetteten Systemen findet man Mikrocontrollerarchitekturen verschiedener Komplexität.

2.1.1 Eingebettete Systeme mit 4-Bit-Mikrocontrollern

4 Bit Mikrocontroller sind sehr einfache Architekturen. Ihre Leistungsfähigkeit ist begrenzt, dafür sind sie extrem billig. Die geringe Komplexität erlaubt die Programmierung in Assemblersprache. Die Programmierung erfolgt auf einem PC. Der entstehende ausführbare Maschinencode ist vom Umfang sehr klein und wird in sogenannten EPROMS abgelegt.

4 Bit Mikrocontroller finden sich überall dort, wo einfache Aufgaben zur Systemsteuerung zu erfüllen sind, z.B. in Waschmaschinen. Hier ist der Einsatz der 4 bit Controller wesentlich preiswerter und robuster als die alternative Verwendung einer mechanischen Steuerung.

2.1.2 Eingebettete Systeme mit 8-Bit-Mikrocontrollern

Diese Architektur ist zur Zeit am weitesten im Embedded Bereich vertreten. Man findet sie sowohl im Consumerbereich als auch in sicherheitskritischen Bereichen der Luft- und Raumfahrt.

Viele dieser Architekturen stammen aus dem PC und Workstationbereich, z.B. der Z80. Es existieren zahlreiche Derivate. Die heute eingesetzten Prozessoren verfügen gegenüber den ursprünglichen Versionen zusätzlich über eine umfangreiche Peripherie, welche auf dem Chip integriert ist. Dazu zählen Ein- und Ausgabeeinheiten wie UARTS[1], aber auch Speicher, Timer,

1. Serielle Schnittstellen

Interrruptcontroller und vieles mehr. Die Fortschritte der Fertigungstechnologien für integrierte Schaltungen erlauben außerdem, die Bausteine mit wesentlich höheren Taktraten zu betreiben. Die CPUs bleiben dabei gegenüber ihren Vorgängern registerkompatibel. Die lange Verfügbarkeit ermöglicht eine extreme Zuverlässigkeit der Systeme mit diesen Mikroprozessoren.

8 Bit Systeme sind etwas komplexer als die 4 Bit Architekturen. Ein solches System läßt sich immer noch relativ leicht in reinem Assembler programmieren, es existieren jedoch auch Compiler für höheren Programmiersprachen. Diese Compiler laufen auf einem PC und generieren Maschinencode, welcher auf dem 8 bit System lauffähig ist.

Die Architektur der Mikrocontroller ist einfach, ebenso die Initialisierung der Peripherie. Wie bei früheren Homecomputern kann eine Anwendung die gesamte Hardware selbst steuern. Dies ermöglicht eine sehr effiziente Programmierung. Die simple Systemarchitektur und die exklusive Programmierung der Hardware durch die Applikation erlauben Laufzeitvorhersagen durch Simulatoren.

2.1.3 Eingebettete Systeme mit 16-Bit-Mikrocontrollern

Diese Architekturen sind den 8 bit Controllern ähnlich. Sie besitzen jedoch breitere Register und Busse.

Gegenüber den 8bit Mikrocontrollern nimmt die Komplexität der peripheren Bausteine zu. Während bei den 8bit Systemen standardisierte Peripherie geinger Komplexität dominiert, finden sich hier zunehmend sogenannte *ASICs*. ASICs sind anwendungsspezifische integrierte Schaltkreise. Diese entlasten die CPU durch intelligente Eigenfunktion und ermöglichen somit eine komplexere Ein- und Ausgabefunktionalität (z.B. Ethernet).

Problematisch ist jedoch die unter Umständen sehr komplexe Initialisierung dieser Chip, welche durch den Mikrocontroller erfolgt. Die Softwareentwicklung wird dadurch komplizierter und fehleranfällig.

2.1.4 Eingebettete Systeme mit 32-Bit-Mikrocontrollern

Moderne 32 bit Prozessoren verfügen in der Regel über Daten und Befehlscaches. Diese Caches minimieren Zugriffe auf externe Speicher. Dadurch werden unnötige Wartezyklen vermieden und die Performance des Systems gesteigert. Weitere Features sind Pipelinestrukturen, parallele Ausführeinheiten, Superskalarität und „Out-of-order-Execution“. Eine nähere Erklärung dieser Begriffe findet sich in [HePa96].

Diese Eigenschaften führen zu einer sehr starken Erhöhung der Rechenleistung. Leider verkompliziert sich durch diese Konzepte aber auch die Programmierung. Der Maschinencode wird von der CPU nicht mehr in der Reihenfolge abgearbeitet, wie er im Assemblerprogramm notiert ist, Speicherzugriffe bewirken nur unter bestimmten Umständen einen Buszugriff usw. Die Komplexität der Peripherie steigt gegenüber den 16-Bit-Systemen nochmals stark an.

Die Entwicklung von Software in reinem Assembler wird sehr kompliziert bis praktisch unmöglich. Auch die Programmierung mit Hochsprachen bleibt anspruchsvoll, wenn eine Applikation das gesamte Hardware-Handling selbst übernehmen muß.

Eine weiteres Problemfeld entsteht durch die hohe Integrationsdichte moderner Chips. Integrierte Schaltkreise besitzen oft mehrere hundert Pins. Es ist nicht mehr möglich, Prototypen

im Labor „von Hand“ zu bauen. Vielmehr ist es von Anfang an erforderlich, gedruckte Platinen mit mehreren Lagen zu verwenden. Die Herstellung dieser Platinen ist entsprechend teuer. Funktioniert ein solcher Prototyp nicht, ist die Fehlersuche kompliziert. Bauformen wie SMD[1] und Ballgrid-Arrays (bis 600 Pins !) erschweren oder versperren Meßgeräten wie Logic Analyzern den Zugriff auf das System.

2.1.5 Fazit

Wie Abbildung 2.1 [KW99] zeigt, steigen die Entwurfsprobleme bei 32-Bit-Systemen überproportional an. Die Entwicklung von Anwendungen wird sehr teuer, wenn dem Programmierer sehr tiefgreifende Systemkenntnisse abverlangt werden (Schulung, Zeitaufwand). Das Problem verschärft sich, wenn Anwendungen entwickelt werden, welche auf verschiedenen eingebetteten Systemen laufen sollen.

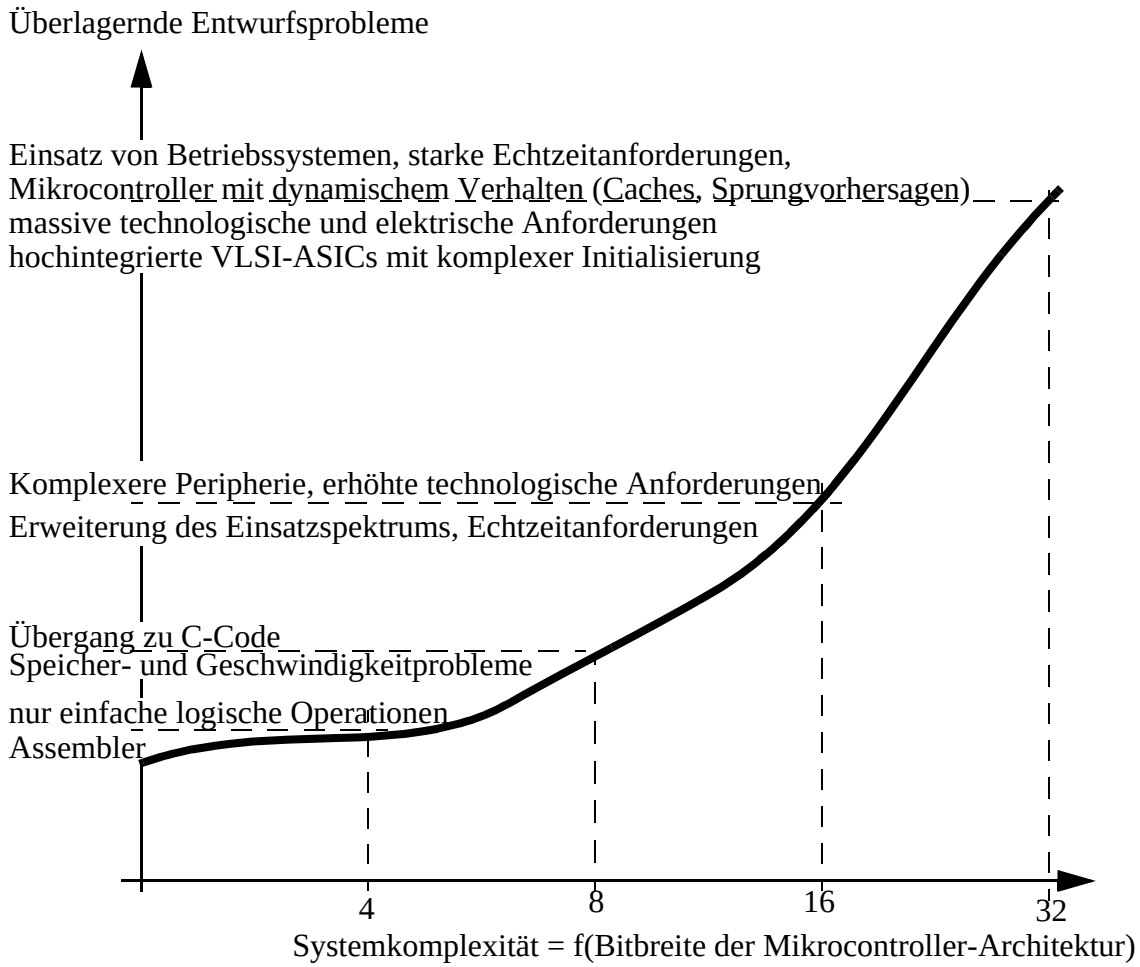


Abbildung 2.1 Qualitativer Anstieg der Entwurfsproblematik bei eingebetteten Systemen (Quelle: [KW99])

Um dennoch eine effiziente Softwareentwicklung zu ermöglichen, ist es erforderlich, die Hardwarestruktur vor den Anwendungsprogrammen zu verbergen. Dies ist Aufgabe von Be-

1. Surface Mounted Device

triebssystemen. Betriebssysteme für eingebettete Systeme stellen im wesentlichen eine Bibliothek von Funktionen dar. Diese Funktionen bilden eine Schnittstelle zwischen der Applikation und der Hardware. Die Applikation kann dadurch auf einem abstrakteren Niveau auf das System zugreifen. So ist es z.B. möglich, auf Flash-Speichern, Floppydisks oder auch SCSI-Festplatten auf die gleiche Art und Weise zuzugreifen, indem ein Filesystem benutzt wird. Alle Zugriffe auf dieses Filesystem werden vom Betriebssystem auf spezielle Hardwareoperationen abgebildet. Dies geschieht für die Anwendung völlig transparent.

Desweiteren entlasten Betriebssystemroutinen den Anwendungsprogrammierer von aufwendigen Verwaltungsaufgaben. So bieten Betriebssysteme wie VxWorks auch für eingebettete Systeme einen TCP/IP Stack, so daß die Kommunikation über das Internet ermöglicht wird. [Wind97]

Für den Hardwareentwickler ist es essentiell, Komponenten bereits vor der Fertigung testen zu können. So benutzen ASIC Entwickler programmierbare Logbausteine, sogenannte *FPGAs* [1]. Die Logik des Chips wird in einer Sprache wie VHDL [2] beschrieben und auf logische Funktionalität getestet. Dannach wird durch ein Synthesetool ein spezielles Bitfile erstellt. Dieses wird auf das FPGA übertragen, wodurch dieses programmiert wird. Der Baustein kann auf seine Funktionalität getestet werden. Wird ein Fehler im Design entdeckt, wird dieser korrigiert und das FPGA neu programmiert. Erst wenn die integrierte Schaltung fehlerfrei arbeitet, wird sie industriell auf billigeren, nicht programmierbaren Chips gefertigt.

1. Field Programmable Array
2. (Very High Speed Integrated Circuits) Hardware Description Language

2.2 Inhalte dieser Arbeit

Für die schnelle und kostengünstige Entwicklung eines eingebetteten System sind also zwei wesentliche Komponenten erforderlich:

- Ein System mit Mikroprozessor, auf welchem ein Echtzeitbetriebssystem bereits lauffähig ist. Dieses System dient dazu, die Applikation testen zu können, ohne daß die eigentliche Zielhardware benutzt werden muß. Das Emulationssystem gestattet eine bessere Testbarkeit, indem z.B. Schnittstellen wie Ethernet für die Kommunikation mit dem Entwicklungs-PC vorgesehen sind. Weiterhin ist es möglich, Software zu entwikkeln, während die Zielhardware noch nicht fertiggestellt ist. Die Entwicklungszeit verringert sich, was einen entscheidenten Wettbewerbsvorteil bedeutet.

- Eine Umgebung, welche die vollständige Funktionsprüfung eines ASIC-Prototypen gestattet, der als FPGA vorliegt. Vollständige Funktionsprüfung erfordert Meßbarkeit aller Signale. Desweiteren muß die Möglichkeit bestehen, entsprechende Testmuster an den Chip zu senden, um seine Funktion zu testen. Die Programmierung des FPGA sollte schnell und einfach sein, d.h. die Rekonfigurierung muß erfolgen können, während das FPGA auf dem Board verbleibt. Ziel ist, Hardware so einfach wie Software abändern zu können.

Das Emulationssystem SPYDER-CORE-P1 vereint beide Komponenten in einem System. Eine genauere Beschreibung der Plattform findet sich in einem späteren Kapitel.

Inhalt dieser Arbeit war, durch die Implementierung geeigneter Softwarekomponenten die Leistungsfähigkeit des Spyder-Systems deutlich zu erhöhen und sein Einsatzfeld zu erweitern. Ich möchte dabei zwei wesentliche Punkte herausstellen:

2.2.1 Erhöhung der Produktqualität

Spyder-Core wurde von Angang an mit der Zielstellung entwickelt, einen Platz als innovatives Entwicklungswerkzeug in den Entwicklungslabors der Industrie zu finden. Um dieses Konzept erfolgreich umsetzen zu können, ist eine ständige Verbesserung des *Produktwertes* von essentieller Bedeutung. Für den Kunden sind dabei vor allem Faktoren wie *Zuverlässigkeit* sowie *Benutzerfreundlichkeit* wesentliche Entscheidungskriterien. Um diesen Ansprüchen gerecht werden zu können, habe ich das Spyder-System um folgende Komponenten erweitert:

- **Test- und Diagnosepaket:** Dieses Paket besteht aus einer Reihe von Testroutinen. Mit Hilfe dieser Routinen ist es möglich, die korrekte Funktionalität des Systems systematisch zu überprüfen, bevor es an einen Kunden ausgeliefert wird. Die Konzeption dieses Paketes beeinhaltet im wesentlichen, den Bereich der *Qualitätssicherung* für das Spyder-Core System abzudecken.

- **Benchmarkprogramme:** Ein Entwickler eingebetteter Systeme muß in der Lage sein, die Echtzeitfähigkeit seines Produktes zu garantieren. Dazu sind insbesondere quantitative Aussagen darüber erforderlich, wie das System auf externe Ereignisse wie z.B Interrupts reagiert. Im Verlauf dieser Arbeit habe ich die für das Betriebssystem VxWorks existierenden Benchmarkprogramme verifiziert.

- **Entwicklungstool für Compile Time Rekonfiguration:** Dieses Werkzeug erlaubt die komfortable Programmierung eines elektrisch löschbaren EPROMS vom Entwick-

lungs-PC aus, wobei der Chip auf dem Board verbleiben kann. Dem Anwender wird durch dieses neue Feature die zeitraubende und fehlerträchtige Programmierung der Speicherbausteine mittels eines externen Programmiergerätes erspart, wodurch sich der Produktivitätsgewinn durch den Einsatz von SPYDER-CORE-P1 weiter erhöht.

2.2.2 Weiterentwicklung zur Forschungsplattform

Die Kerngedanke der Arbeit am Forschungszentrum Informatik[1] ist es, innovative Ergebnisse der wissenschaftlichen Forschung in kurzer Zeit nutzbar zu machen. Dies erfordert neben einer praxisorientiert Umsetzung von Forschungsergebnissen natürlich auch die ständige Erforschung wissenschaftlichen Neulandes. Bisher betand die Aufgabe des Spyder-Core System vor allem darin, als leistungsfähiges Werkzeug die Effizienz der Arbeit in den Entwicklerlabors zu erhöhen.

Da die Architektur von SPYDER-CORE-P1 die Programmierung eines FPGA durch den Mikroprozessor gestattet, eröffnet sich für dieses System nun ein neues interessantes Gebiet, welches bisher noch Gegenstand der Forschung ist.

Man stelle sich vor, daß man ein System entwickeln muß, welches eine sehr komplexe Hardware benötigt. Das Design ist dann so umfangreich, daß unter Umständen mehrere teure FPGA benötigt werden, um es zu implementieren. Teure Eingebettete Systeme sind aber „schlechte“ Systeme. Dies gilt jedenfalls dann, wenn man ein solches System kommerziell vertreiben will. Wie kann man eine Maschine realisieren, welche das vorgegebene Pflichtenheft erfüllt und trotzdem einen geringen Hardwareaufwand benötigt?

Eine Möglichkeit eröffnet sich, wenn folgende Vorraussetzung erfüllt ist: Unter Umständen werden nie alle Komponenten der Hardware gleichzeitig benötigt. Es macht also keinen Sinn, das vollständige Hardwaredesign zur Verfügung zu stellen. Das System sollte sich die zu einem Zeitpunkt benötigte Hardware einfach generieren und anschließend wieder verwerfen. Dies Konzept erscheint auf den ersten Blick nicht realisierbar, schließlich kann man Hardware nicht einfach erzeugen und wieder verschwinden lassen. Mit Hilfe von FPGAs ist die Idee jedoch zu verwirklichen. Durch die Programmierbarkeit dieser Bausteine reicht es aus, die Design Files auf einem geeigneten Medium zu speichern und bei Bedarf in den programmierbaren Logikchip zu laden. Dieser verhält sich danach wie ein „klassischer“ ASIC. Diese Hardwareprogrammierung „nach Bedarf“ nennt man *Run-Time-Rekonfiguration* (RTR). Ein späterer Abschnitt widmet sich ausführlicher den Möglichkeiten, welche sich durch die Anwendung von RTR ergeben.

Um das Spyder-Core-System als Forschungsplattform für Run-Time-Rekonfiguration einsetzen zu können, mußte es um einige weitere Features erweitert werden:

- Bereitstellung eines Filesystems zur lokalen Speicherung der Design Files
- Implementierung einer Schnittstelle, welche einer Applikation die einfache Rekonfiguration eines FPGA ermöglicht

1. Das Forschungszentrum Informatik (FZI) in Karlsruhe ist ein Partnerinstitut dr Universität Leipzig. Die Ergebnisse dieser Arbeit enstanden im Rahmen meiner (externen) Diplomarbeit am FZI.

Kapitel 3

Grundlagen

In diesem Kapitel möchte ich in einige prinzipielle Grundlagen bezüglich der Hard- und Softwarekomponenten einführen, welche für das weitere Verständnis dieser Arbeit notwendig bzw. hilfreich sind. Dieses Kapitel erhebt keinen Anspruch auf Vollständigkeit. Der interessierte Leser sei also auf die weiterführende Literatur verwiesen. So bauen die Ausführungen über die Hardwarekomponenten im wesentlichen auf die im Referenzteil gelisteten Manuals und Data Sheets der Hersteller sowie [KW99] auf, so daß ich diese Quellen für ein ausführlicheres Studium empfehle.

3.1 Echtzeitbetriebsysteme

Die Software eines eingebetteten Systems kommuniziert über Peripheriegeräte mit der Außenwelt. Sie registriert Ereignisse, analysiert diese und führt daraufhin entsprechende Aktionen aus. Es liegt in der Natur vieler Ereignisse, daß diese asynchron auftreten. So kann z.B. ein Sensor in einer Fertigungsstraße zu einem beliebigen Zeitpunkt ein Signal an das System senden, worauf dieses reagieren muß. In der Praxis unterliegt die Zeitspanne zwischen dem Auftreten des Signals und der Reaktion des eingebetteten Systems oft harten Zeitrestriktionen. Damit diese Restriktionen eingehalten werden können, muß das Verhalten des Systems vorhersagbar sein. Für das Betriebssystem bedeutet dies, daß Reaktionszeiten auf Ereignisse garantiert werden müssen. Bei klassischen Betriebssystemen ist dies nicht der Fall. Hier kann es durchaus vorkommen, daß ein gerade laufender Prozeß durch einen anderen verdrängt und seine Abarbeitungszeit dadurch extrem verlängert wird. Das System reagiert auf ein externes Ereignis dann viel zu spät, was zu katastrophalen Folgen führen kann.

Definition 3.1: Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind derart, daß die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorbestimmten Zeitpunkten anfallen. (DIN 44300)

Definition 3.2: Real-Time systems are those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.[1] [St90]

Echtzeitanforderungen lassen sich in „hard real time“ und „soft real time“ einteilen. Während bei harten Echtzeitbedingungen das Ergebnis durch eine Überschreitung der Antwortzeit vollkommen unbrauchbar wird, ist bei Systemen mit „weichen“ Echtzeitanforderungen solch eine Verletzung für einen Teil der Ergebnisse noch tolerierbar. Im weiteren Verlauf meiner Ausführungen werde ich mich auf die Diskussion harter Echtzeitbedingungen beschränken.

Die folgende Tabelle macht deutlich, durch welche Eigenschaften sich Echtzeitbetriebssysteme gegenüber klassischen Betriebssystemen auszeichnen, um den Anforderungen gerecht zu werden [MS97]

Klassisches System	Echtzeitsystem
hohe Antwortzeiten	Latenzzeiten gering (typ. 20 µs)
hohe Schedulingzeiten (ab 10 ms)	Scheduling im Mikrosekundenbereich (30-200 µs)
lange Interruptsperren	spezielle, schnelle Interruptbehandlung
nichtunterbrechbare Kernelaufrufe	teilweise preemptive Kernel
gleichmäßige Rechenzeitaufteilung	Prioritätsklassen
Annahme jedes Prozesses	Last- und Möglichkeitsanalyse

Tabelle 3.1: Klassische versus Echtzeitsysteme ([MS97])

Ein Echtzeitbetriebssystem zeichnet sich also keinesfalls nur dadurch aus, besonders „schnell“ zu sein, da eine Minimierung der mittleren Antwortzeiten nicht ausreichend ist.

Um ein Echtzeitbetriebssystem zu implementieren besteht zum einen die Möglichkeit, ein Unixsystem um einen preemptiven Microkernel zu erweitern. Dies ermöglicht die Erfüllung harter Echtzeitbedingungen. Der Codeumfang der Routinen läßt sich klein halten, so daß diese Systeme für den Embedded-Bereich mit seinen teils beschränkten Speicherresourcen geeignet sind. Die Implemetierungsanforderungen sind jedoch relativ hoch, da aufwendige Lock- und Synchronisationsverfahren realisiert werden müssen. Vertreter dieser Systeme sind QNX von QNX Software, Chorus (Chorus) sowie VxWorks, welches von der Firma WindRiver Systems vertrieben wird. Eine nähere Betrachtung zu VxWorks ist im folgenden Kapitel zu finden.

Eine andere Variante der Implementierung besteht darin, einen Real-Time-Scheduler zu entwerfen. Dieser läßt dann den eigentlichen Betriebssystemkernel als Idle-Task laufen. Diese Lösung führt zu einem sehr schnellen System, auf welchem Standardanwendungen weiterhin lauffähig sind. Leider sind solche Systeme nur für synchrone Echtzeitanforderungen geeignet. Ein geplantes Scheduling ist ebenfalls nicht möglich. Vertreter sind EP/LX auf Lynx OS (Control Data) sowie Linux mit Realtime Patch.

1. „Echtzeitsysteme sind solche Systeme, in welchen die korrekte Funktion des Systems nicht nur von den logischen Ergebnissen der Berechnungen abhängt, sondern auch von der Zeitspanne, in der diese erzeugt werden“

Das Scheduling erfolgt bei Echtzeitbetriebssystemen nach Klassen. Ein Hardware Dispatcher verwaltet Interrupts, während der Software Dispatcher den einzelnen Prozessen bzw. Threads[1] Rechenzeit zuteilt:

- Realtime Gruppe für deterministisches Echtzeitverhalten. Das Scheduling erfolgt streng auf der Basis von Prioritäten und Zeitscheibe
- System Gruppe für System Threads: Die Prioritäten für Vertreter dieser Gruppe sind bereits beim Start des Betriebssystems fest vorgegeben. Darunter fallen z.B. Swaper und Pager.
- Time Sharing Gruppe: Alle unkritischen Prozesse laufen im Timesharing mit einer Zeitscheibe von 10..100 ms.

1. Die Begriffe Thread und Prozeß werden im Rahmen dieser Abhandlung als Synonyme behandelt

3.2 Programmierbare Logikbausteine

3.2.1 Complex Programmable Logic Devices (CPLD)

Um Peripheriebausteine wie serielle Schnittstellen, Flash-Memory oder auch FPGAs in ein eingebettetes System zu integrieren, muß man die Registerstrukturen dieser Chips in den Adreßraum des Mikroprozessors einbinden. Dazu schließt man die Daten- und Adreßleitungen der Chips an den Systembus an. Da mehrere Peripheriegeräte „parallel“ am Bus angeschlossen sind, muß vor jedem Schreib- oder Lesezugriff ein Chip-Select-Signal generiert werden, welches einen Baustein auswählt. Steuersignale wie Chip-Select werden erzeugt, indem eine kombinatorische Logik die Bussignale des Mikrocontroller auswertet.

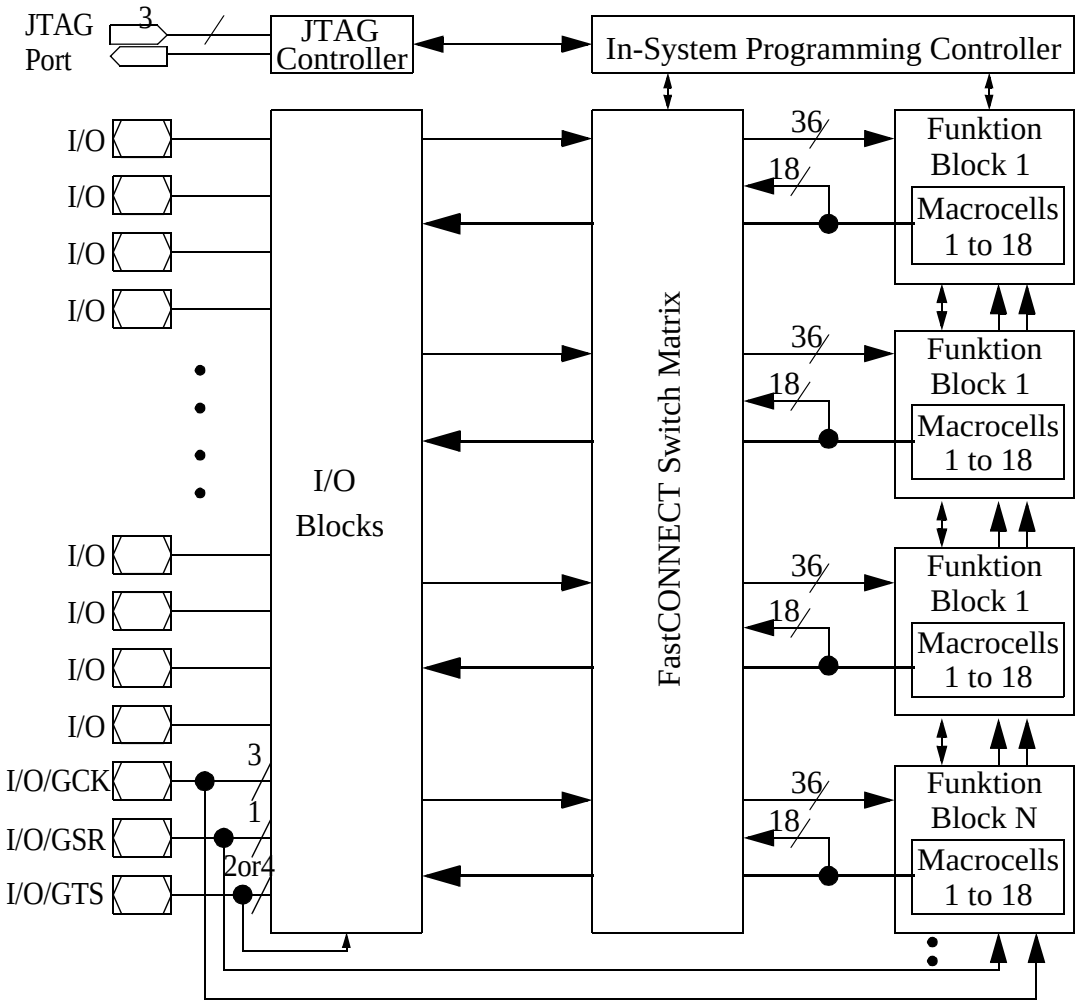


Abbildung 3.1 Xilinx XC9500 CPLD-Architektur ([Xil98/1])

Für die Implementierung solcher und anderer Funktionen werden in der Regel sogenannte *komplexe programmierbare Logikbausteine* (engl. Complex Programmable Logic Devices) eingesetzt. Das Einsatzgebiet der CPLDs liegt also hauptsächlich dort, wo gegenüber den FPGAs (siehe folgende Abschnitte) relativ wenige Logikgatter und viele I/O Blöcke benötigt werden. Die Verzögerungen sind gering. Typische Signallaufzeiten von Pin zu Pin liegen im

Bereich von nur 5 Nanosekunden. Die Verzögerungszeit läßt sich bei CPLDs vorhersagen. Dies ist möglich, weil die Verzögerungswerte der einzelenen Komponenten fest ist.

Die CPLD-Architektur besteht im wesentlichen aus bis zu 16 Funktionsblöcken, einer FastConnect-Switch-Matrix sowie bis zu 192 I/O Blocks. Eine Übersicht über die Architektur findet sich in Abbildung 3.1. Im folgenden möchte ich diese Komponenten kurz erläutern:

- **Funktionsblöcke (engl. Function Blocks - FB)**: Ein Funktionsblock besteht aus 18 Makrozellen, welche die kombinatorische oder speichernde Logikfunktion erfüllen. In jeden FB führen 36 Signale hinein, welche aus der FastCONNECT Switch Matrix kommen. Jedes dieser Signale steht auch negiert zur Verfügung, so daß 72 Eingangssignale bereitstehen. Der Funktionsblock generiert 18 Ausgangssignale, die wieder zur FastCONNECT Matrix weitergeführt werden. Darüber hinaus ist jeder FB mit dem globalen Set/Reset Signal (GTS), den globalen System Taktleitungen (3 x GCK) und der System Reset Leitung (GSR) verbunden.

  Im FB geschieht nun folgendes: Die 72 Eingangssignale bilden den Input in ein programmierbares AND-Array. Dieses Array kann 90 Produktterme bilden. Jeder dieser Terme kann jeder Makrozelle zugeführt werden, wobei die Anzahl der Terme pro Makrozelle auf fünf beschränkt ist. Die Zuführung erfolgt dabei über einen sogenannten Produktterm-Allokator. In der Makrozelle werden die Produktterme in einer OR oder XOR Stufe zu einer kombinatorischen Funktion zusammengefaßt. In jeder Makrozelle ist ein Speicherelement vorhanden. Dieses kann als D- bzw. T-FlipFlop benutzt werden.

- **Schnelle Verbindungsmatrix (engl. FastCONNECT Switch Matrix)**: Dieser Matrix stehen alle I/O Blocks und alle Ausgangssignale der Funktionsblöcke zur Verfügung. Daraus können bis zu 36 Signale ausgewählt werden, die dann als Eingangssignale für die Funktionsblöcke dienen.

- **Eingangs und Ausgangsblöcke (engl. Input/Output Blocks - I/OB)**: I/O Blocks bilden die Schnittstelle zu den externen Pins des CPLD. Jeder dieser Blocks verfügt über einen Eingangs- und einen Ausgangstreiber. Es besteht die Möglichkeit, diese Treiber in einen hochohmigen Zustand zu schalten, welcher auch als *Tri-State* bezeichnet wird. Diese Maßnahme ist notwendig, um undefinierte Logikpegel zu vermeiden.

Die Speicherung der Informationsbits in den Funktionsblöcken, der Switch Matrix und den I-O Blöcken beruht auf der FLASH Technologie. Flashspeicher behalten auch nach dem Abschalten der Betriebsspannung ihre Information.

3.2.2 SRAM basierte FPGA

FPGA (Field Programmable Gate Arrays) ermöglichen im Gegensatz zu CPLDs die Implentierung wesentlich komplexerer Logikdesigns. Damit eignen sie sich zur Emulation von anwendungsspezifischen Hardwarekomponenten. Im folgenden möchte ich SRAM basierte FPGA Architekturen anhand der XC4000 und Virtex Serie der Firma Xilinx im Überblick erläutern. Die Beschreibung einer Anwendung der XC4000 Serie folgt in einem späteren Kapitel.

3.2.2.1 Die XC-4000 Serie

Im wesentlichen besteht ein XC4000 FPGA aus folgenden Komponenten:

- Konfigurierbaren Logikblöcken (Configurable Logic Blocks -CLB)
- Ein- und Ausgabeblöcken (IOBs)
- Konfigurierbaren Verbindungen der CLBs und IOBs
- Takttreibern, Tri-State Treibern und anderen speziellen Funktionseinheiten
- SRAM-Zellen, die der Speicherung der Konfiguration dienen

Wesentlich ist, daß die Speicherung der Konfiguration in SRAM-Zellen geschieht. Im Gegensatz zum FLASH basierten CPLD geht diese Information nach dem Abschalten der Betriebsspannung verloren. Die Initialisierung eines FPGA muß also nach jeder Inbetriebnahme des Systems erneut erfolgen. Dies kann z.B. durch den Mikroprozessor erfolgen. Eine weitere Möglichkeit besteht darin, die Konfigurationsdaten in einem seriellen EEPROM zu halten. Der FPGA ist in der Lage, nach dem Einschalten der Betriebspannung diese Informationen auszulesen und sich selbst zu konfigurieren (serieller Master Mode). Eine im FPGA implementierte Zustandsmaschine ermöglicht dies. Sowohl die Konfigurierung durch den Mikroprozessor als auch mit Hilfe des EEPROM waren Gegenstand dieser Arbeit und werden in folgenden Kapiteln näher erläutert.

Im weiteren Verlauf dieses Abschnittes möchte ich die Komponenten der XC4000 Serie vorstellen:

CLBs:

Abbildung 3.2 zeigt den Aufbau eines konfigurierbaren Logikblocks. Diese Blöcke implementieren die eigentlichen Logikfunktionen.

Zwei Funktionsgeneratoren (F und G) verfügen über jeweils 4 Eingangssignale. Aus diesen können sie jede beliebige logische Funktion berechnen (F’ und G’). Die Funktion wird dazu in Form einer Look-Up-Table (LUT) beschrieben, z.B:

F_1	F_2	F_3	F_4	F'
0	0	0	0	F'_1
0	0	0	1	F'_2
0	0	1	0	F'_3
0	0	1	1	F'_4
...	...	...	...	...
1	1	1	1	F'_{16}

Tabelle 3.2: Look Up Table

Die Eingänge adressieren 16 SRAM Zellen, in denen Funktionswerte als 0 bzw. 1 abgelegt sind.

Desweiteren findet sich im CLB ein dritter Funktionsgenerator (H) mit 3 Eingängen. An die Inputs von H werden die Ausgangsignale der Generatoren F und G geführt, an den dritten Eingang Signale, welche ihren Ursprung außerhalb des CLB haben. Aus diesen 3 Inputs kann ebenfalls jede logische Funktion (H') generiert werden.

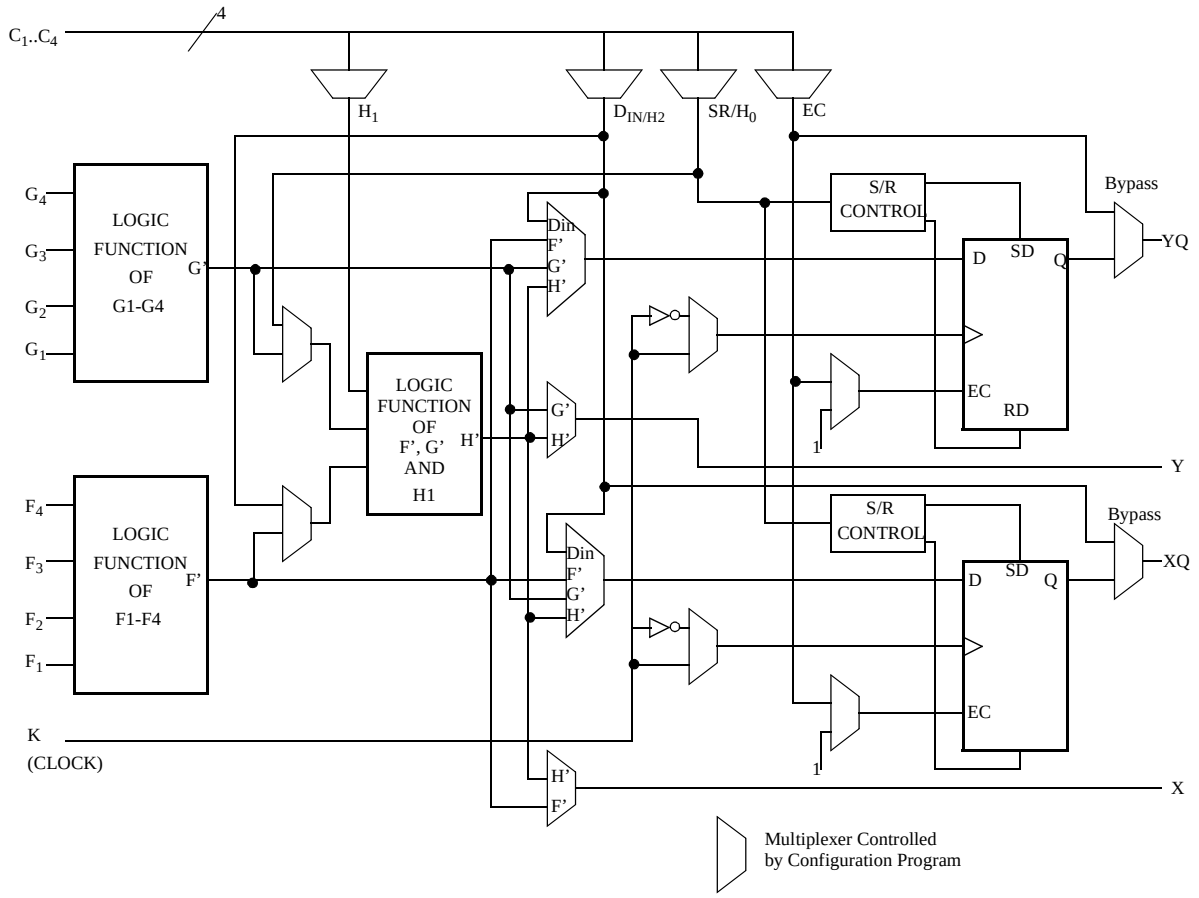


Abbildung 3.2 Xilinx XC4000 konfigurierbarer Logikblock (Quelle: [Xil98/1])

Zwei flankengesteuerte D-FlipFlops erlauben die Speicherung der Funktionsergebnisse F', G' und H' sowie eines externen Signals. Die gespeicherten Werte werden über die Ausgänge XQ und YQ dem Verbindungsnetzwerk zur Verfügung gestellt.

Der CLB besitzt zwei weitere Ausgänge. F' oder G' können mit dem ersten Output X verbunden werden, H' mit dem zweiten Ausgang Y.

Ein CLB kann für die Implementierung jeder der folgenden Funktionen[1] benutzt werden:

- Jede Funktion mit bis zu 4 Variablen, eine zweite Funktion mit bis zu vier unabhängigen Variablen plus einer dritten Funktion mit maximal 3 unabhängigen Variablen. Sollen drei Funktionen gleichzeitig generiert werden, muß ein Ergebnis mittels eines FlipFlops gespeichert werden
- Jede Funktion mit maximal fünf Variablen
- .Jede Funktion mit fünf Variablen plus einigen Funktionen mit sechs Variablen
- Einige Funktionen mit bis zu neun Variablen

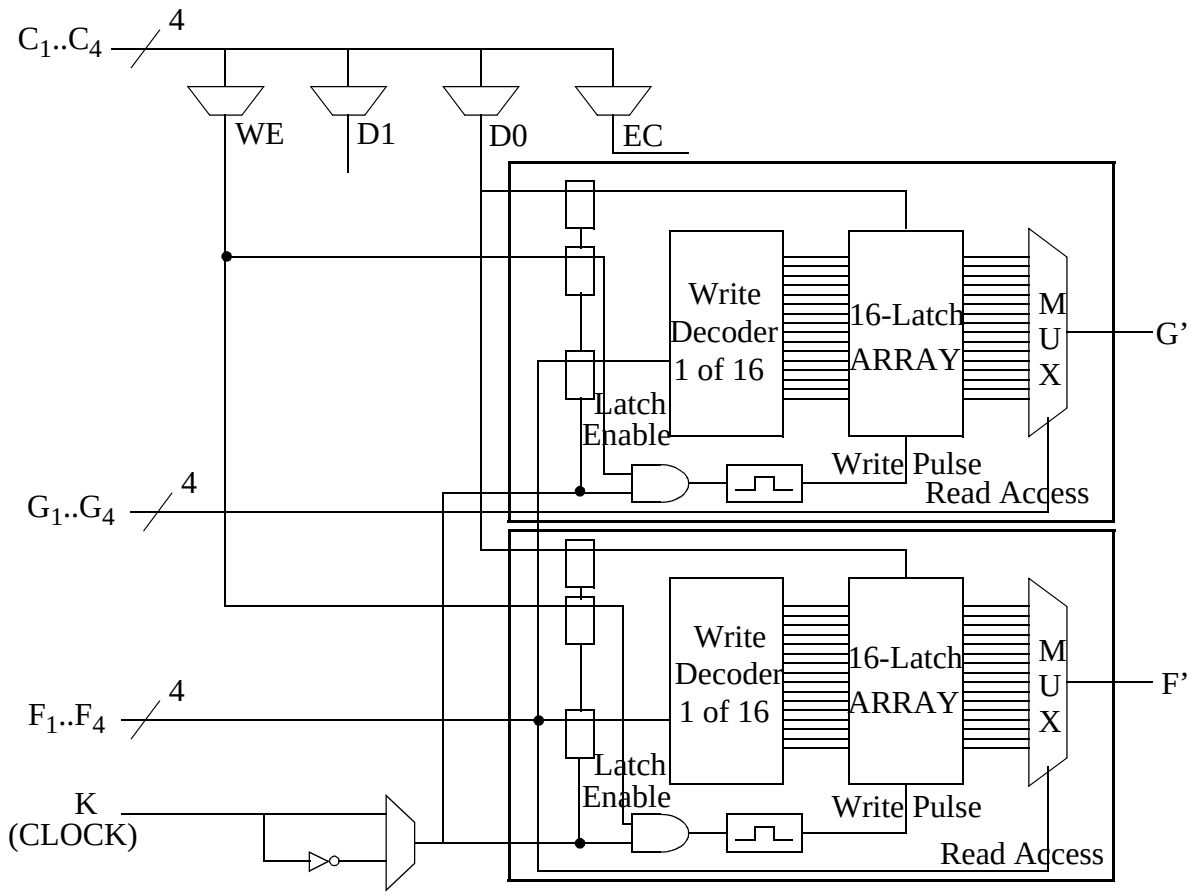


Abbildung 3.3 Xilinx XC4000 Dual-Port-SRAM Block (Quelle:[Xil98/1])

1. Im Rahmen dieser Abhandlung werden nur logische Funktionen betrachtet

Ein CLB muß nicht zwingend für die Berechnung logischer Funktionen verwendet werden. Es ist möglich, die SRAM-Zellen, welche eigentlich zur Speicherung der Look-Up-Tables dienen, als feinkörniges On-Chip-SRAM zu nutzen. Diese Eigenschaft wird in der Literatur auch oft als *fine-grained SRAM capability* bezeichnet.

Die SRAM Blöcke können *level sensitive*, *edge triggered* oder *dual port edge triggered* konfiguriert werden. Abhängig vom Betriebsmode kann ein einzelner CLB als 16x2, 32x1 oder auch als 16x1 Bit-Array betrieben werden, wie Tabelle 3.3 zeigt.

	16x1	**16x2**	**32x1**	**Edge Triggered Timing**	**Level Sensitive Timing**
Single-Port	✓	✓	✓	✓	✓
Dual-Port	✓			✓	

Tabelle 3.3: Unterstützte RAM Modes (Quelle:[Xil98/1])

Ein Funktionsgenerator kann also zur Speicherung von 16 Bit benutzt werden (16x1). Durch Kombination von zwei Generatoren ist die Speicherung von 32 Bits möglich (32x1). Alternativ können auch 16 Worte mit einer Breite von zwei Bit abgelegt werden (16x2)

Im Level-Sensitive-Mode dient ein externes Signal (Write Enable) als write-strobe, während im flankengesteuerten Betrieb dieses Signal mit dem Taktsignal verknüpft wird. Das Timing im flankengesteuerten Betrieb ist also im Gegensatz zum Level-Sensitive-Mode synchron.

Abbildung 3.3 beschreibt einen CLB im 16x1 Dual Port Mode mit einem flankengesteuerten Timing. Für Funktionsgenerator F ist die Lese- und Schreibadresse immer identisch, da F1 bis F4 sowohl mit dem Write-Decoder als auch mit dem Read Decoder (MUX) verbunden sind. F verhält sich also wie ein 16x1 Single-Port-Array. Bei Funktionsgenerator G hingegen sind Lese- und Schreibadresse entkoppelt. Die Leseadresse bildet sich aus den CLB-Eingängen G1 bis G4, während die Schreibadresse durch F1 bis F4 bestimmt wird. Den Ausgang des Generators G bezeichnet man Dual-Port-Out, kurz DPO. Benutzt man [G1..G4] als Leseport und [F1..F4] als Schreibport, so läßt sich ein FIFO realisieren, auf das gleichzeitige Lese- und Schreibzugriffe möglich sind. Datenausgang ist dann DPO.

Ein / Ausgangsblöcke (IOB):

Um die interne Logik mit den externen Gehäusepins des FPGA zu verbinden, dienen sogenannte Ein/Ausgangsblöcke. In der Literatur ist ebenfalls die englische Bezeichnung In/Output Blocks (IOB) üblich. Ein IOB kann sowohl als Eingang, als Ausgang oder auch bidirektional konfiguriert werden.

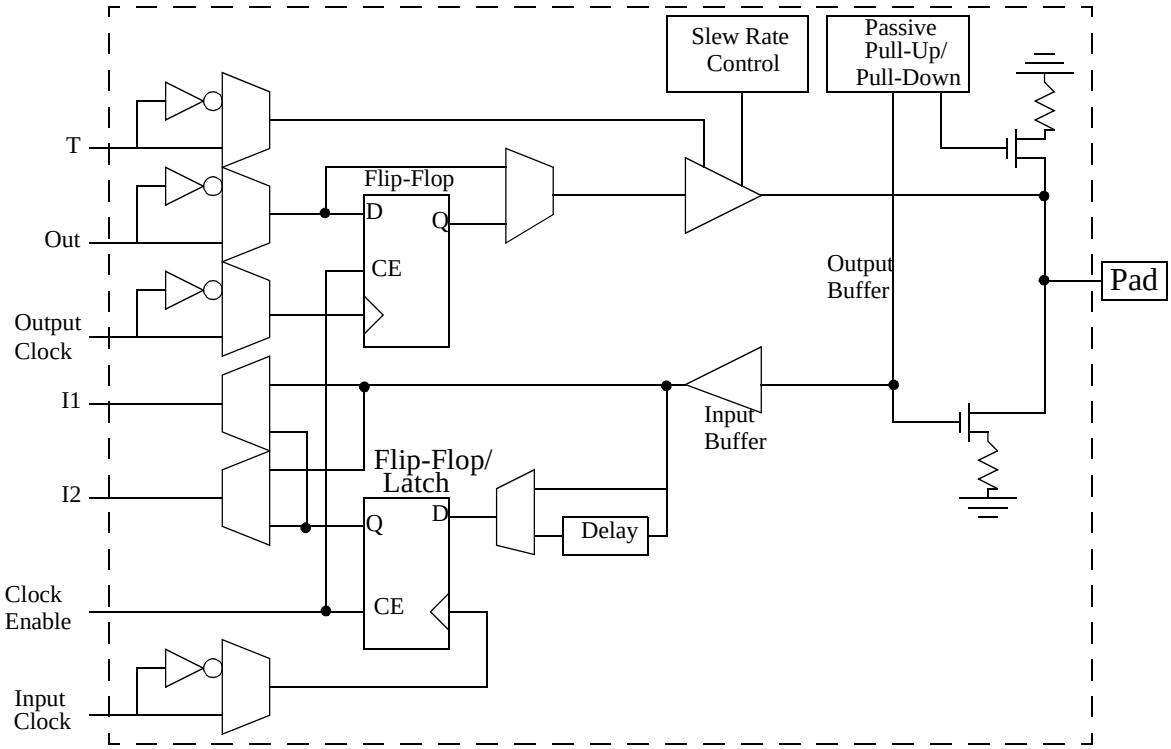


Abbildung 3.4 Xilinx XC4000 Ein- Ausgabeblock (Quelle: [Xil98/1])

Die Eingänge der XC4000E Serie sind *global* für TTL oder CMOS Pegel konfigurierbar, ebenso die Ausgänge. Die Einstellungen für Ein- und Ausgangsblöcke sind unabhängig voneinander. XC4000XL FPGAs sind ebenfalls TTL- und CMOS-kompatibel, allerdings mit einem CMOS Pegel von 3.3 V

Eingangssignale können mittel eines flankengetriggerten FlipFlops oder eines Level-Sensitiven Latches gespeichert werden.

Das Ausgangssignal kann invertiert oder direkt zum Pad weitergereicht werden. Eine Zwischenspeicherung ist ebenso möglich (flankengesteuertes D-FlipFlop). Die Ausgangspuffer lassen sich in einen hochohmigen Zustand schalten. Die Implementierung von Tri-State-Outputs bzw. oder Tri-State-Ausgängen für bidirektionale Signale wird damit möglich.

Konfigurierbare Verbindungen

Alle internen Verbindungen bestehen aus metallischen Leitungssegmenten. Um eine konfigurierbare Verbindung zwischen den Komponenten des FPGA zu ermöglichen, existieren programmierbare Verbindungspunkte und Verbindungsmatrizen.

Für die Beschreibung dieser Verbindungen ist auch der Begriff *Routing Resources* üblich.

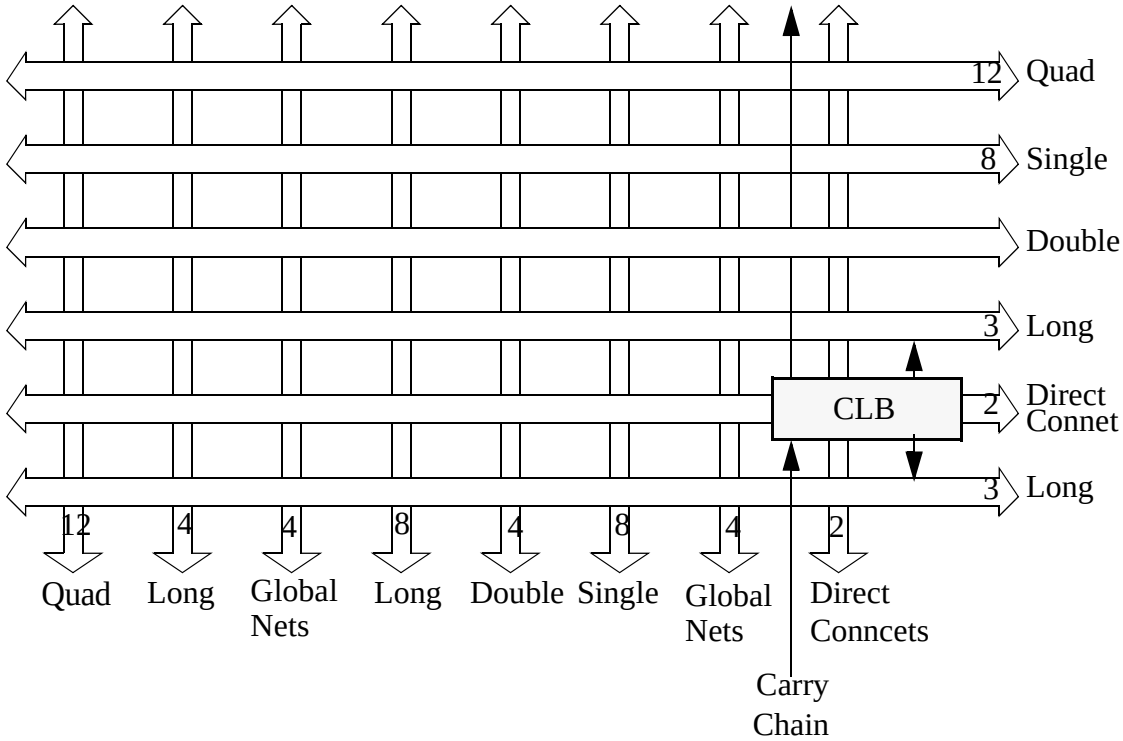


Abbildung 3.5 High Level Routing Diagramm für XC4000 Serie (Quelle: [Xil98/1])

Wie funktioniert die Programmierung dieses Netzes? Das Prinzip besteht darin, die Verbindung durch einen *Pass-Transistor* herstellen oder trennen zu lassen. Dieser Transistor wird vom Ausgang einer Speicherzelle (SRAM) getrieben. Somit ist es möglich, auch das Routing Design in SRAM-Zellen zu speichern.

Wie Abbildung 3.5 zeigt, existieren ein einem XC4000 FPGA verschiedene Gruppen von Verbindungsleitungen: *Single-*, *Double-* und *Quad-Length-Lines* sowie sogenannte *Direct-Connects*. Desweiteren finden sogenannte *Long-Lines* und *Global-Nets* Verwendung.

Single-Length-Lines verbinden Switchmatrizen paarweise untereinander. Pro Verdrahtungskanal laufen immer acht dieser Leitungen sowohl in horizontaler als auch in vertikaler Richtung.

Double-Length-Lines verbinden ebenfalls Switchmatrizen, im Gegensatz zu den Single-Length-Lines aber immer nur jede zweite Matrix. Es gibt vier Leitungen für die horizontale und noch einmal vier Stück für die vertikale Richtung.

Quad-Length-Lines überspringen immer zwei Switchmatrizen. Es existieren pro Kanal je 12 Leitungen für beide Richtungen.

Direct Connects verbinden keine Switchmatrizen, sondern konfigurierbare Logikblöcke untereinander. Die Verbindung erfolgt in vertikaler Richtung. Es stehen pro Kanal zwei Leitungen zur Verfügung. Da Direct Connects die Swichtmatrizen umgehen, bieten sie eine sehr schnelle Verbindung zwischen zwei CLBs.

Long-Lines erstrecken sich über den ganzen Verdrahtungskanal. Es gibt von ihnen pro Kanal drei Stück, die horizontal verlaufen. Hinzu kommen acht für die vertikale Richtung.

Um spezielle Signale (z.B Clock, Reset usw.) zu verteilen, existieren die *Global Nets.* Es gibt acht getrennte Global Nets in der XC4000-Architektur. Die Leitungen verlaufen über die gesamte Fläche des FPGA

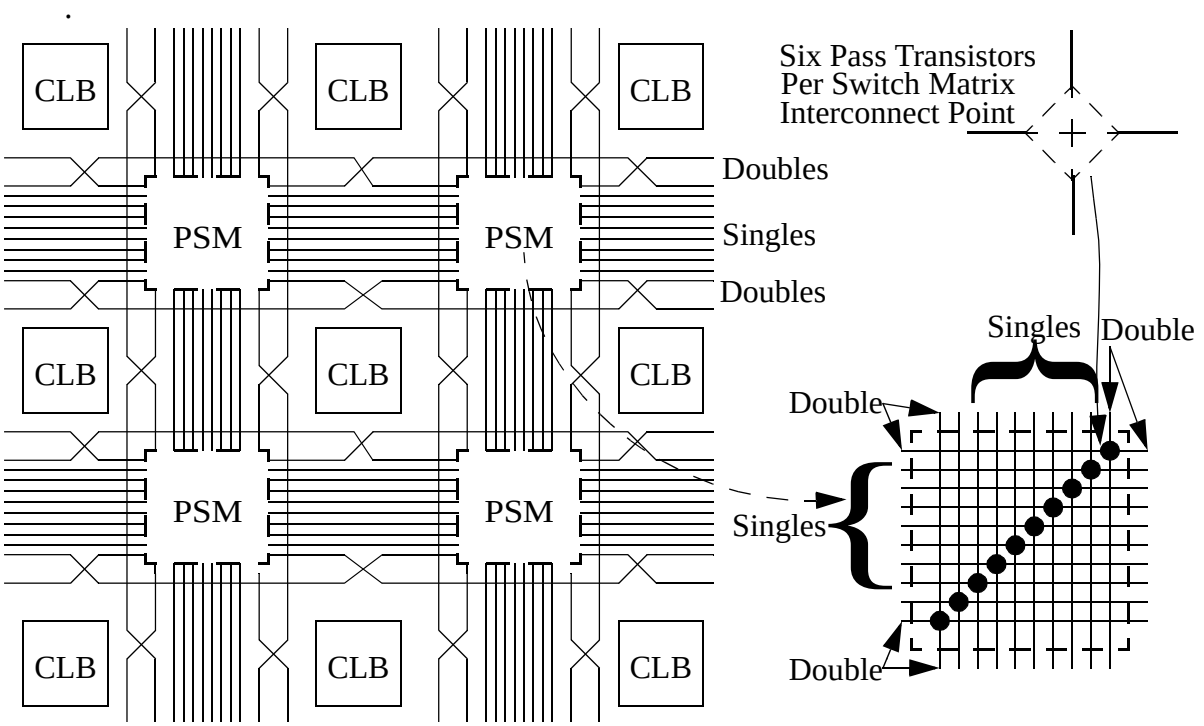


Abbildung 3.6 Switchmatrizen (Quelle: [Xil98/1])

Bei der Beschreibung des Verbindungsleitungen wurden bereits mehrmals die Switchmatrizen erwähnt. Diese Matrizen dienen dazu, Leitungssegmente entweder miteinander zu verbinden (bei gleichbleibender Richtung) oder eine Richtungsänderung zu bewirken. Also “Weiche“ wird dient ein bereits zu Beginn des Abschnitts erwähnter *Pass-Transistor*, welcher durch eine SRAM-Zelle gesteuert wird.

Abbildung 3.6 zeigt, wie die Switchmatrizen in die FPGA Architektur integriert sind. Die mit „CLB“ gekennzeichneten Elemente sind konfigurierbare Logikblöcke, „PSM“ die (programmierbaren) Switchmatrizen.

Zusammenfassung

In diesem Abschnitt wurde die Architektur der Xilinx XC4000 Serie vorgestellt. Folgende wesentliche Eigenschaften sollte der Leser im Gedächtnis behalten, da sie vor allem für das Verständnis der Abschnitte von Bedeutung sind, die sich mit der Problematik der Run Time Rekonfiguration auseinandersetzen:

- Ein XC4000 FPGA ist beliebig oft rekonfigurierbar.
- Bei einer Konfigurierung wird der *gesamte* FPGA neu geladen, eine bereichsweise Konfigurierung ist nicht möglich.
- Die Konfiguration dauert lange (mehrere Millisekunden, abhängig vom Typ).

3.2.2.2 Virtex

Die Virtex Serie ist die z.Z. leistungsfähigste FPGA Architektur der Firma Xilinx. Taktfrequenzen bis zu 200 Mhz und die Möglichkeit, bis zu 1 Million Gates zu implementieren (Quelle: [Xil99]), eröffnen dieser Architektur Anwendungen im High-End-Bereich.

Wie die XC4000 Architektur verfügt jeder Virtex-FPGA über konfigurierbare Logikblöcke sowie über Ein-Ausgabeblöcke (IOBs).

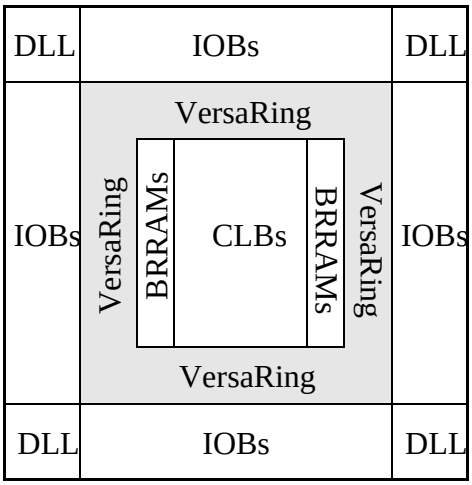


Abbildung 3.7 Virtex Architektur (Quelle: [Xil99])

Die CLBs sind durch eine *General Routing Matrix (GRM)* miteinander verbunden. Diese Matrizen beinhalten ein Array von Routing Switches, welche sich an den Verbindungspunkten der vertikalen und horizontalen Verbindungskanäle befinden.

Die Verbindung zwischen den CLBs und den IOBs erfolgt durch einen sogenannten Versa Ring, welcher die CLBs umschließt.

Abbildung 3.7 zeigt die Virtex-Architektur im Überblick. Neben den bisher erwähnten Komponenten sind dort sogenannte *Block RAMs* (*BRAMs)* zu erkennen. BRAMs sind eingebettete SRAM-Blöcke. Neben den bereits in der XC4000-Architektur vorhandenen SRAM-Fähigkeiten in den Look-Up-Tables (LUT) stellen diese eine weitere Ressource für schnelle Speicherkapazitäten dar.

Weiterhin bietet die Virtex-Architektur für jeden globalen Clock-Input-Buffer ein sogenanntes *Delay Locked Loop (DLL).* Diese Einrichtung dient dazu, Signalverschiebungen zu beseitigen, welche zwischen dem Clock-Input-Pad des Chips und den internen Clock-Input-Pins durch Laufzeitunterschiede entstehen. Die DLLs sorgen also dafür, daß im gesamten Chip ein synchrones Taktsignal angeboten wird.

Die Konfigurierbaren Logikblöcke bestehen bei Virtex-FPGAs aus zwei identischen Teilen *(Slices)*. In jedem Slice finden sich zwei *Logic Cells (LCs)*. Eine solche Zelle besteht aus einem Funktionsgenerator, Carry-Logik und einem Speicherelement (FlipFlop). Die Carry-Logik ermöglicht die effiziente Implementierung sehr schneller Arithmetikfunktionen. So kann z.B. ein Volladdierer mit nur einer Logic Cell realisiert werden.

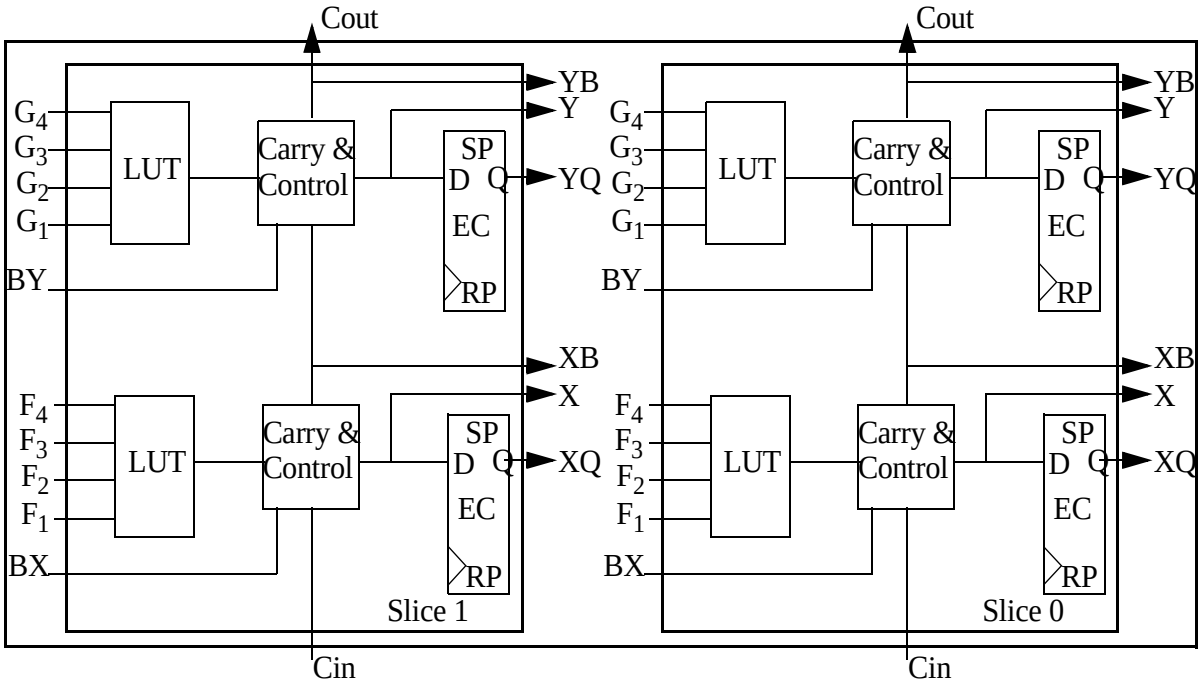


Abbildung 3.8 2-Slice Virtex CLB (Quelle: [Xil99])

Die IOBs der Virtex Architektur sind wesentlich flexibler konfigurierbar als die der XC-4000 FPGAs. Die I/O Blöcke sind in acht Gruppen zusammengefaßt, wobei jeder Gruppe eine andere Versorgungsspannung zugeordnet werden kann. Virtex FPGA sind daher zusammen mit unterschiedlichsten elektrische Schnittstellen einsetzbar, z.B. PCI und ISA Bussen.

Virtex FPGAs sind wesentlich schneller und flexibler konfigurierbar als XC-4000 Bausteine. So existiert im Gegensatz zur XC4000-Architektur ein *paralleler* Lademodus. Desweiteren ist es nicht zwingend notwendig, den gesamten FPGA neu zu konfigurieren. Die Virtex-Architektur erlaubt es, ausgewählte Bereiche zur Laufzeit neu zu laden.

Zusammenfassung:

Ich möchte folgende Eigenschaften der Virtex-FPGA festhalten:

- Virtex FPGA verfügen über Eigenschaften wie hohe Taktfrequenz, flexible I/O Anbindung und die Möglichkeit, sehr umfangreiche Logikschaltungen zu implementieren.
- Die Virtex Architektur erlaubt eine schnelle und partielle Rekonfigurierbarkeit.

Die FPGAs der Virtex Serie sind damit zumindest potentiell geeignet, spezielle Hardware zu entwerfen, welche für spezielle Aufgaben durch Anwendung von Run Time Rekonfiguration eine Performance erzielen kann, die über der von aktuellen High End Prozessoren (z.B. Power PC G3) liegt.

Leider befindet sich die Hardware für eine Virtex basierte Emulationsplattform am Forschungszentrum Informatik Karlsruhe zur Zeit noch in Entwicklung, so daß sie im Rahmen dieser Arbeit nicht weiter untersucht werden konnte.

3.2.3 Antifuse basierte FPGA (Actel)

Actel FPGAs sind im Gegensatz zu den SRAM basierten Bausteinen von Xilinx nicht rekonfigurierbar. Ist ein Actel FPGA programmiert, so ist diese Konfiguration dauerhaft. Dies hat auch zur Folge, daß die Konfiguration im Chip auch nach dem Abschalten der Betriebsspannung erhalten bleibt.

Ich möchte im folgenden die Antifuse-Technologie am Beispiel der Actel Act-1 Serie kurz vorstellen. Antifuse basierte FPGAs bestehen im wesentlichen aus einer Anzahl von Logikmodulen, I/O Buffern sowie horizontalen und vertikalen Verbindungsleitungen (Abbildung 3.9). An den Kreuzungspunkten der Leitungen befinden sich sogenannte *PLICE Antifuses*, welche die Leitungen zunächst voneinander isolieren. Während der Programmierung werden diese Antifuses adressiert und programmiert. Dadurch werden die Logikmodule so verschalten, wie es die Schaltung erfordert, welche implementiert werden soll.

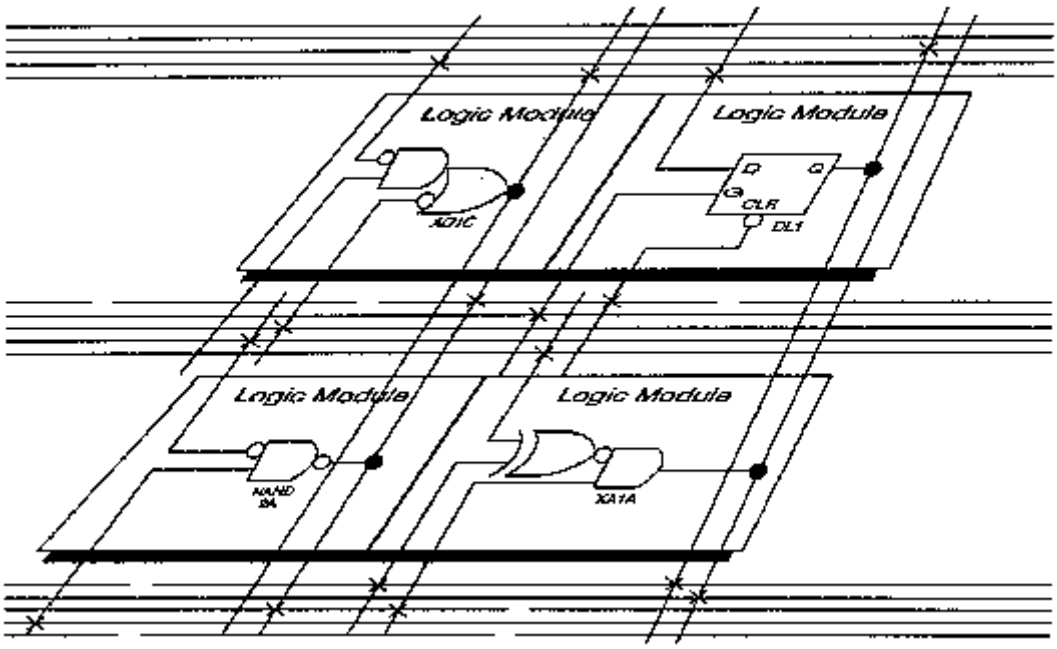


Abbildung 3.9 ACT 1 Device (Quelle: [Act94])

Ein Logikmodul der ACT-1 Serie ist relativ einfach aufgebaut. Es besteht aus nur wenigen logischen Gattern. Jedes Logikmodul kann folgende logische Funktion generieren ([KW99]):

$$Y = \bar{S}_1 \wedge \bar{B}_0 \wedge D_{00} \vee \bar{S}_1 \wedge B_0 \wedge D_{01} \vee S_1 \wedge \bar{B}_1 \wedge D_{10} \vee S_1 \wedge B_1 \wedge D_{11} \text{ mit } S_1 = A_0 \vee A_1$$

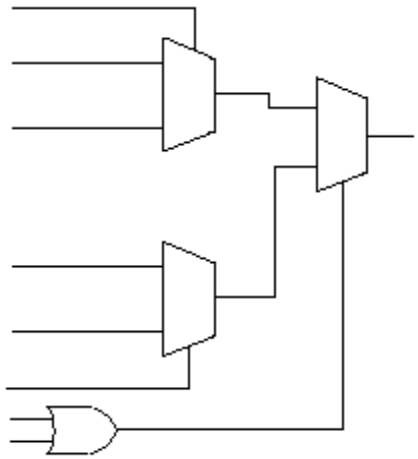


Abbildung 3.10 ACT-1 Logikblock

Die Daten und Steuerleitungen der Multiplexer werden mit den Signalen der Verdrahtungskanäle verbunden. Dabei kann auch auf Signale zugegriffen werden, die permanent Low (logisch 0) oder High (logisch 1) sind. Durch entsprechende Verdrahtung kann ein solcher Logikblock vier logische Grundfunktionen (NAND, AND, OR, NOR) für zwei, drei oder vier Variablen berechnen. Ebenso ist die Implementierung von D-Latches und Funktionen wie XOR , AND-OR und OR-AND Funktionen möglich.

3.3 Speicherbausteine

3.3.1 EEPROMs

Wie *EPROMs*[1]speichern auch Speicherbausteine der *EEPROM*[2]-Technologie Informationen dauerhaft, d.h. auch nach dem Abschalten der Betriebsspannung.

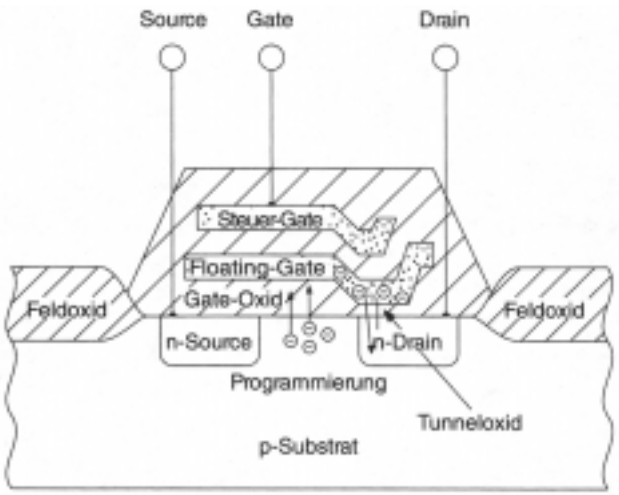


Abbildung 3.11 EEPROM Speichertransistor (Quelle: [HM93])

Die dauerhafte Speicherung der Daten wird mit einem Speichertransistor realisiert. Dieser Transitor ist ein MOSFET[3], welcher zusätzlich über ein sogenanntes *Floating Gate* verfügt. Dieses Floating Gate befindet sich zwischen dem Substrat und dem Steuer Gate und ist vollständig isoliert.

Das Gate hingegen ist üblicherweise mit einer Wortleitung verbunden, das Drain mit einer Bitleitung und die Source mit einem Referenzpotential (z.B. Vcc). Der Speichertransistor wirkt als Schalter zwischen Wort- und Bitleitung.

Ein normaler MOS-Transistor würde bei aktivierter Wortleitung über Drain, Kanal und Source Vcc verbinden und somit auf High-Pegel treiben, der Transitor speichert eine Eins.

Ist das Floating Gate neutral, verhält sich der Speichertransistor ebenso. Anders ist es, wenn das Floating Gate negativ geladen ist. Die Ladung des Floating Gate schirmt dann das Feld des Steuer Gates ab. Somit kann der Transitor auch bei aktivierter Wortleitung nicht durchschalten, da das Steuer Gate kein ausreichend starkes Feld aufbauen kann. Der Speichertransistor speichert eine Null.

Die Beladung des Floating Gates geschieht durch einen elektrischen Impuls mit einer Spannung von 20 V und ca. 50 ms Dauer. Durch diesen Impuls entstehen im Kanalbereich schnelle Ladungsträger (Elektronen), die genug Energie besitzen, die Isolierschicht zwischen

1. EPROM = Electrical Programmable ROM
2. EEPROM = Electrical Erasable PROM
3. MOSFET = Metal-Oxid-Semiconductor Field Effect Transistor

Substrat und Floating Gate zu überwinden. Danach sind sie im sind gut isolierten Floating Gate „gefangen“, da sie nach Abkühlung zu wenig Energie besitzen, um die Isolierschicht nochmals zu durchdringen. Die Ladungen bleiben über sehr lange Zeiträume (mindestens zehn Jahre) erhalten.

Die Programmierung eines EEPROM erfolgt also nach dem gleichen Prinzip wie die eines EPROM. Während ein EPROM nur mit UV-Licht gelöscht werden kann (durch das Licht erhalten die Elektronen im Floating Gate Energie, um es zu verlassen), ist das Löschen eines EEPROM durch einen elektrischen Spannungsimpuls möglich.

Dazu verfügt der Speichertransistor gegenüber einem EPROM über eine Schicht mit dünnerem Tunneloxid. Diese befindet sich zwischen dem Substrat und einem Teil des Floating Gates, welcher nach unten gezogen ist. Zum Entladen des Floating Gates legt man einfach eine umgekehrte Spannung zwischen Gate und Drain an. Dadurch bekommt das Drain gegenüber dem Gate ein positives Potential und die Elektronen genug Energie, um die Isolierschicht zu überwinden und damit das Floating Gate zu verlassen. Dabei muß man aber darauf achten, daß dieser Prozeß nicht zu lange dauert. Ansonsten verlassen zu viele Elektronen das Floating Gate und es erhält ein positives Potential. Das Resultat wäre eine verschobene Transistorkennlinie, die den weiteren normalen Betrieb verhindern würde. Glücklicherweise generieren Schaltkreise wie die seriellen EEPROMs von Atmel ihre Schreib- und Löschzyklen intern, so daß sich der Programmierer nicht auch noch mit den Tücken der Physik plagen muß.

3.4 Rekonfigurationsmethoden für FPGA

Rekonfigurierbare Logik ist eine Architektur, welche den Entwurf von *flexiblen* Systemen ermöglicht, welche durch ihr anwendungsspezifisches Design in Bereichen eingesetzt werden können, die selbst den schnellsten Mikroprozessoren und Supercomputern verschlossen bleiben. Ein weiterer Vorteil ist, daß einmal entworfene Designs in späteren Projekten wiederverwendet werden können, ähnlich wie Softwarebibliotheken.

3.4.1 Compile Time Rekonfiguration (CTR)

Compile Time Rekonfiguration ist die einfachste Methode der Rekonfiguration von programmierbaren Logikschaltkreisen. Typisch für die CTR Methode ist, daß die Designdaten in das FPGA geladen werden und dann *unverändert* bleiben, während die Anwendung ausgeführt, d.h. das FPGA benutzt wird. Es existiert also nur eine systemweite Konfiguration.

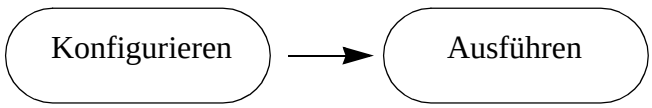


Abbildung 3.12 Compile Time Rekonfiguration([HuWi95])

Eine typische und weit verbreitete Anwendung für CTR ist die bereits im Abschnitt 2.1.5 beschriebene Emulation von ASICs. Dabei ist die Strategie der *statischen Hardware Allokation* die Eigenschaft, die das klassische ASIC Design mit der CTR Methode verbindet.

Die CTR Methode stellt keine speziellen Anforderungen an die FPGA Architektur, da es prinzipiell ausreichend ist, das FPGA einmalig konfigurieren zu können.

3.4.2 Run Time Reconfiguration (RTR)

Im Gegensatz zur CTR verfolgt die Methode der Run Time Rekonfiguration einen anderen Ansatz. Statt eine Konfiguration in das FPGA zu laden, welche die gesamte logische Funktionalität enthält, entwirft man mehrere Konfigurationen. Diese Konfigurationen werden während der Laufzeit der Anwendung nachgeladen.

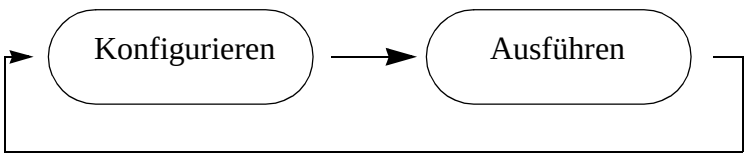


Abbildung 3.13 Run Time Rekonfiguration ([HuWi95])

RTR Applikationen verfolgen also eine Strategie der *dynamischen Hardware Allokierung*.

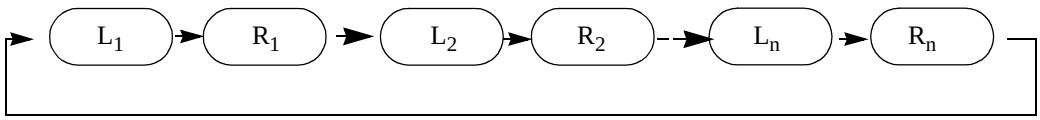
Hadley und Hutchings unterteilen in [WirHut97] die Run Time Rekonfiguration dabei in globale und lokale Run Time Rekonfiguration.

- **Globale Run Time Rekonfiguration** lädt bei jeder Rekonfiguration das *gesamte* FPGA neu.

  Diese Vorgehensweise ist natürlich nur dann sinnvoll, wenn sich die Anwendung in sogenannte *Time-Exclusive-Segments* aufteilen läßt. Zeitexklusive Segmente dürfen nicht gleichzeitig ausgeführt werden. Der Prozeß, der einen Algorithmus in diese zeitexlusiven Segmente zerlegt, wird als temporale Partitionierung bezeichnet. Ähnlich der strukturellen Partitionierung, bei der die Elemente des Schaltkreises in funktionell zusammengehörenden Gruppen angeordnet werden, hat der Prozeß der temporalen Partitionierung das Ziel, Elemente zusammenzufassen, welche gleichzeitig operieren. Ein Algorithmus wird temporal partitioniert, indem er in Phasen zerlegt wird. Jede dieser Phasen wird dann in einem eigenen FPGA Design implementiert.

  Weiterhin muß beim Design beachtet werden, daß die entstandenen Designkomponenten meist nicht unabhängig voneinander arbeiten können. Typischerweise verarbeitet jede Komponente Daten irgendwelcher Art, die an den Nachfolger weitergegeben werden, also auf irgendeine Weise gespeichert werden müssen. Eine Speicherung im FPGA selbst verbietet sich, da dieser ja bei der nächsten Rekonfiguration vollständig überschrieben wird. Somit muß der Systemdesigner andere geeignete Wege finden, um diese Inter-Configuration Kommunikation zu gewährleisten.

  Die Designs können dann in den FPGA geladen werden, wenn sie benötigt werden.



L_i .. Konfiguration i laden (Load)
R_i .. Konfiguration i benutzen (Run)

Abbildung 3.14 Globale Run-Time Rekonfiguration ([HuWi95])

  Für globale Run Time Konfiguration eignen sich prinzipiell alle FPGA, welche sich beliebig oft neu programmieren lassen, also z.B die Bausteine der XC4000 Serie. Selbstverständlich ist die Zeit, welche zum Laden benötigt wird, ein limitierender Faktor. Daher sollte die Partitionierung so erfolgen, daß die Designs eine möglichst lange Zeit im FPGA verbleiben können, bevor ein Umladen erfolgen muß.

- **Lokale Run Time Rekonfiguration** geht einen Schritt weiter. Es wird nicht mehr das geamte FPGA neu konfiguriert, sondern nur Teile der Logik nachgeladen, während der Rest des Designs unverändert bleibt.

Anstatt relativ große, globale Partitionen zu laden wird versucht, nur die benötige Funktionalität zu jedem beliebigen Zeitpunkt nachzuladen. Dadurch kann der Zeitaufwand für das Nachladen verringert werden, da sich kleinere Konfigurationen natürlich schneller in das FPGA übertragen lassen. Ebenso ist es möglich, die Allokation von Hardware effizienter zu gestalten. Die Ressourcen, welche ein FPGA bietet, lassen sich mittels lokaler RTR besser ausnutzen. So erzielten Hadley und Hutchings im RRANN2[1] Projekt eine Reduzierung der Konfigurationszeit um 25 % und eine 50 prozentigen Erhöhung der Neuronendichte. ([HuWi95])

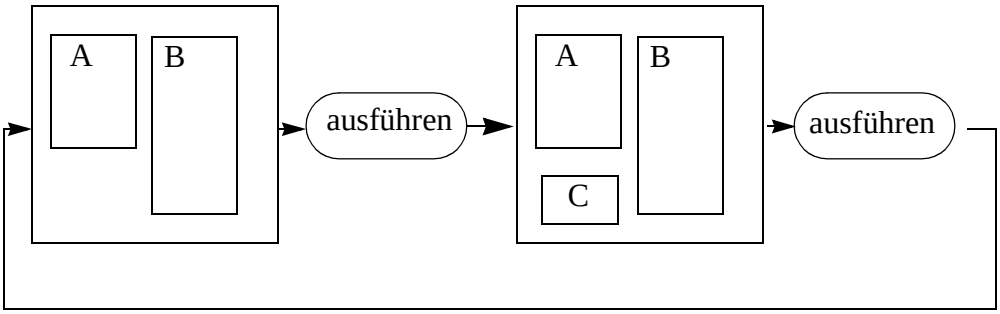


Abbildung 3.15 Lokale Run-Time Rekonfiguration ([HuWi95])

Applikationen, die sich gut mittels lokaler Run-Time Konfiguration realisieren lassen, basieren mehr auf einer funktionalen Aufteilung ihrer Komponenten als auf einer zeitlichen Partitionierung. Eine Applikation wird für die Implentierung in eine möglichst feinkörnige Menge von Operationen zerlegt. Diese Operationen müssen nicht zwingend zeitexklusiv sein. Beim Systemdesign sollte aber dennoch darauf geachtet werden, daß Komponenten, welche längere Zeit nicht benutzt worden, aus dem FPGA entfernt werden, um Ressourcen freizugeben.

Lokale Run Time Konfiguration funktioniert nur mit FPGA, welche sich partiell nachladen lassen. Darunter zählen die Vertreter der XC6000 und Virtex Serie der Firma Xilinx.

1. Run-Time Reconfiguration Artificial Neural Network

Kapitel 4

Entwicklungsumgebung

Im diesem Kapitels möchte ich die bereits vorhandenen Hard- und Softwarekomponenten vorstellen, auf denen diese Arbeit aufbaut.

Zunächst werde ich die Hardwareplattform SPYDER-CORE-P1 vorstellen. Weiterhin werden im Verlauf des Kapitels das Echtzeitbetriebsystem VxWorks sowie alternative Entwicklungsumgebungen besprochen.

4.1 Die Emulationsumgebung SPYDER-CORE-P1

SPYDER-CORE-P1 ist bereits die zweite Generation eines Emulationssystem für eingebettete Systeme, welches am Forschungszentrum Informatik in Karlsruhe entwickelt wurde.

Seine Architektur wurde so gewählt, daß sie eine schnelle und kostengünstige Entwicklung eingebetteter Systeme ermöglicht. Besonders hervorzuheben ist dabei die Möglichkeit des *parallelen* Hard- und Softwareentwurfs.

Das System besteht im wesentlichen aus zwei Komponenten: einer Emulationsumgebung für applikationsspezifische Hardware sowie einem Mikroprozessorsystem.

Ein SRAM-basiertes FPGA (XC 4010E) der Firma Xilinx emuliert digitale Schaltungen. Der I/O Bereich dieses Bausteins ist über ein ISA-Bus Interface mit dem Host PC verbunden. Somit kann das FPGA vom PC aus konfiguriert werden und die digitale Schaltung auf Korrektheit überprüft werden, indem über den ISA-BUS Testmuster an das FPGA geschickt werden. Weiterhin ist das FPGA mit einem Erweiterungsstecker verbunden, welcher den Anschluß von kundenspezifischer diskreter Hardware erlaubt. Darüber hinaus ist es auch in den Adreßraum des Mikrocontrollers eingebunden.

Zur automatischen Konfiguration des FPGA ist ein serielles EEPROM der Firma Actel vorhanden. Die Programmierung dieses Chips kann ohne ein externes Programmiergerät erfol-

gen. Ermöglicht wird dies durch das Tool *ELOAD*. Eload ist Teil der Softwareumgebung für Spyder Core und entstand im Rahmen dieser Arbeit. Eine Beschreibung findet der Leser in Kapitel 5.

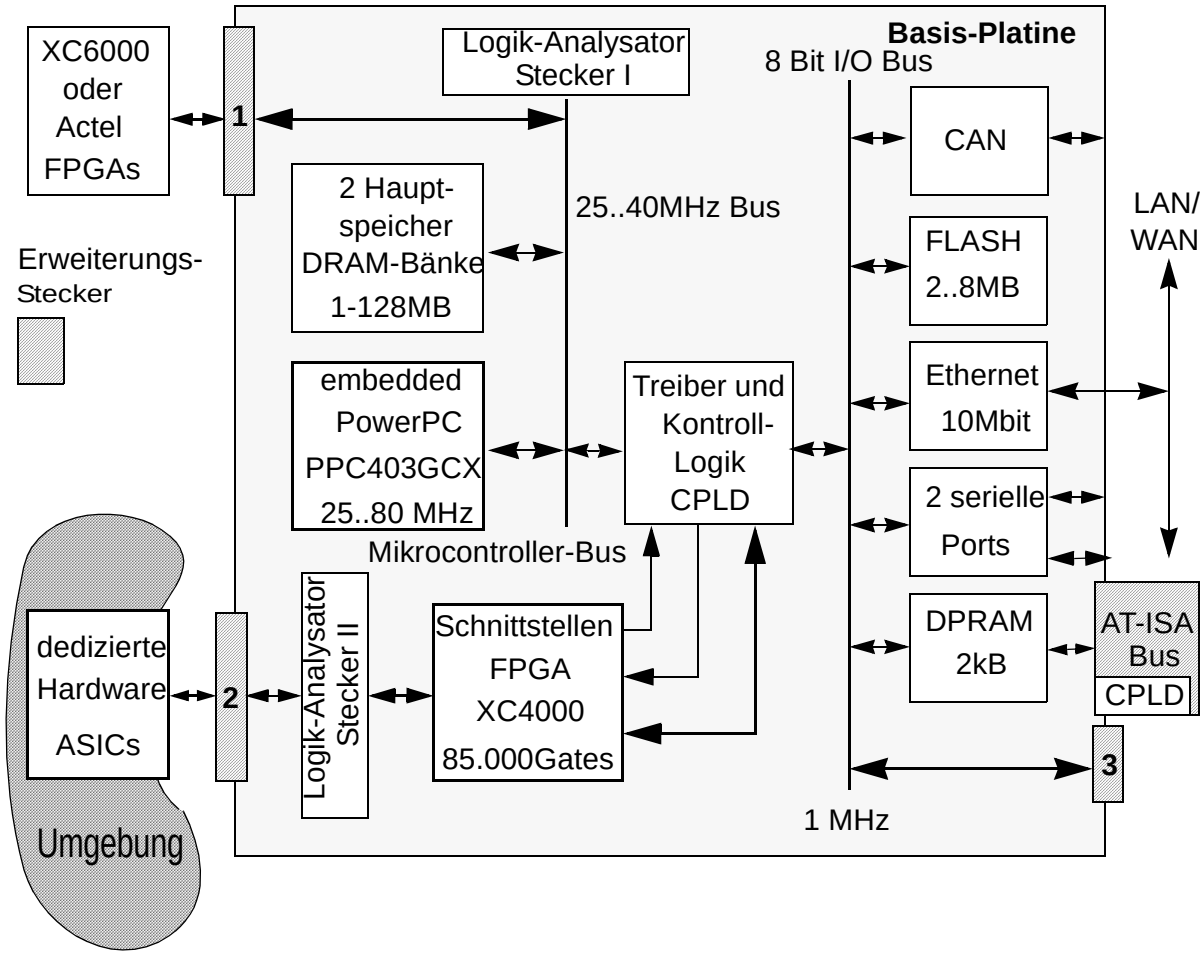


Abbildung 4.1 Blockschaltbild von SPYDER-CORE-P1 (Quelle: [KW99])

Die Mikrocontroller-Partition von SPYDER CORE baut auf einer Power PC CPU der Firma IBM auf. Zum Einsatz kommen dabei die für den Embedded-Bereich angepaßten Controller Power PC 403GA und 403GCX. Diese Chips zeichnen sich durch eine einfache Integrierbarkeit in ein eingebettetes System aus, da sie u.a. über On-Chip-Funktionalitäten wie DRAM-Controller und serieller Schnittstelle verfügen. Durch ihre Taktfrequenzen von 40 bis 80 Mhz eignen sich die Prozessoren für den Einsatz von (Echtzeit) Betriebssystemen. Ein EPROM, 8 MByte Flash-Memory und Steckplätze für Standart-PS/2-Simm gestatten eine flexible Systemkonfiguration. Die Kommunikation mit dem Entwicklungs-PC unterstützt eine Ethernet-Schnittstelle.

Der Mikrocontrollerkern von SPYDER-CORE-P1 dient vor allem dazu, Software zu testen, welche für eine bestimmte Plattform entwickelt wurde. Dies kann bereits geschehen, wenn diese Zielplattform noch gar nicht in Hardware zur Verfügung steht. Vielmehr kann Applikations-

spezifische Hardware durch das FPGA emuliert und zusammen mit der auf dem Mikrocontroller laufenden Anwendung getestet werden.

4.2 Entwicklungsumgebung SDS

Die Softwareentwicklung für eingebettete Systeme findet in der Regel auf einem Host-PC statt. Auf diesem ist ein Cross Compiler installiert, welcher Maschinencode erzeugt, der auf der Zielplattform lauffähig ist.

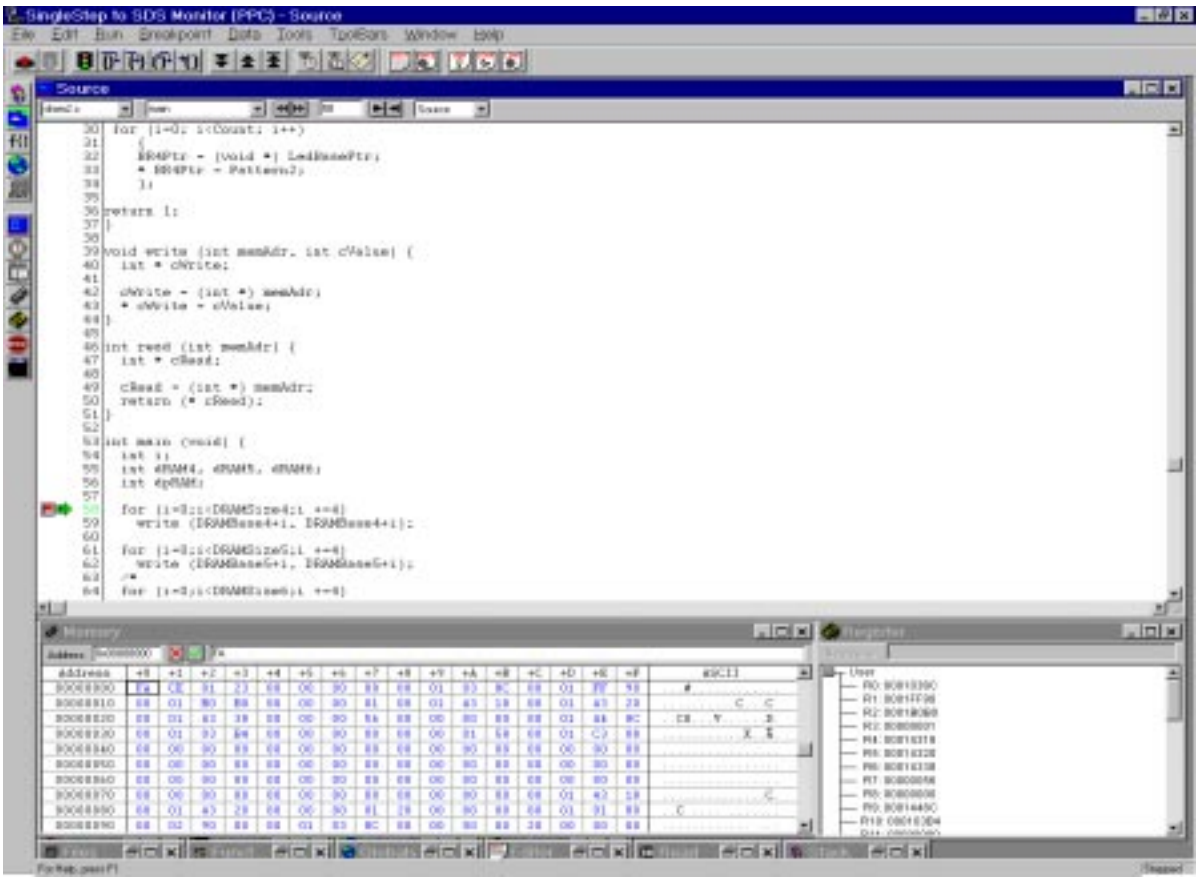


Abbildung 4.2 SDS Monitor Software

SDS ist ein Cross-Debugger. Über ein EPROM wird auf dem Target ein kleines Programm gestartet, welches mit der auf dem Host-PC laufenden Entwicklungsumgebung kommuniziert. Die Kommunikation erfolgt dabei über die seriellen Schnittstellen, über die sowohl der PC als auch die Targetplattform (hier SPYDER-CORE-P1) verfügen.

SDS ermöglicht dem Entwickler, am PC eine Anwendung zu erstellen, mittels eines Cross Compilers zu übersetzen und dann über die serielle Schnittstelle auf die Zielhardware zu laden und zu starten. Dort kann das zu testende Programm schrittweise abgearbeitet und auf Fehler analysiert werden.

SDS ist *kein* Betriebssystem und bietet auch keinerlei Kommunikationsmöglichkeiten wie die Funktion *printf* der Programmiersprache C oder gar Netzdienste. Der Programmierer muß

die gesamte Verwaltung der Hardware selbst organisieren. SDS eignet sich daher nur zum Test kleiner Applikationen.

4.3 Echtzeitbetriebsystem RTEMS[1]

Ursprünglich wurde RTEMS vom Militär entwickelt und steht heute als Open-Source-Software zur Verfügung. RTMES ist ein vollständiges Betriebssystem, welches die Verwaltung von Tasks, Semaphoren, Interrupts usw. realisiert.

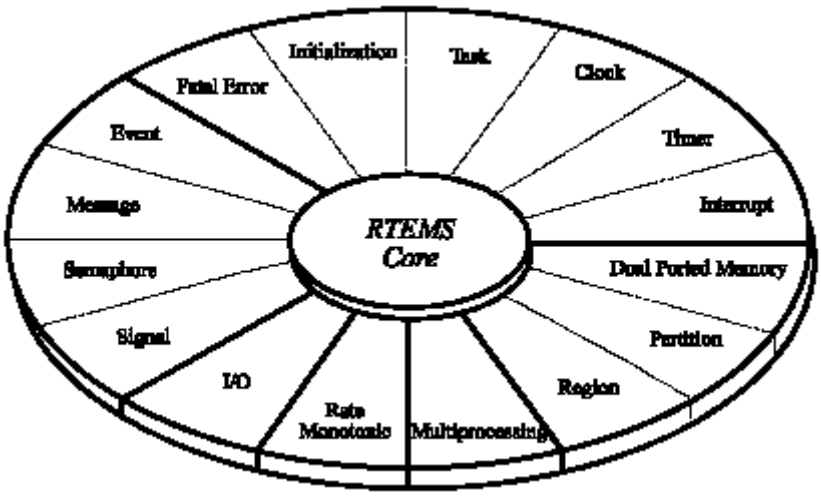


Abbildung 4.3 RTEMS Architektur (Quelle: [Army96])

Durch die Einbindung einer C-Library ist auch die Benutzerkommunikation mittels üblicher Funktionen wie *printf* möglich.

RTEMS liegt als Bibliothek vor. Eine Anwendung oder auch eine andere Library wie libc benutzt RTEMS-Funktionen, welche durch einen Linker in die Applikation eingebunden werden. Jede gelinkte RTEMS-Applikation stellt also ein monolithisches File dar, welches alle Anwendungs- und Betriebssystemfunktionen beinhaltet.

RTEMS wurde von mir im Rahmen meiner Studienarbeit auf das Vorgängermodell des Spyder Prozessorboards portiert und im Verlauf der Diplomarbeit an die Spyder Architektur angepaßt.

Die Übersetzung von RTEMS basierter Software erfolgt wieder mittels eines Cross Compilers, zur Übertragung auf das Target dient ein spezieller Loader, welcher ebenfalls von mir implementiert wurde. Dieses Prinzip ist flexibel genug, so daß auch Anwendungen erstellt und geladen werden können, die keinen Gebrauch von RTEMS Funktionen machen. Das vorhandene Paket aus Cross Compiler, Loader und RTEMS bietet sich als Low-Cost Entwicklungsumgebung an, da es vollständig auf frei verfügbarer Software basiert.

1. Real Time Executive for Multiprozessor Systems

RTEMS diente im Rahmen dieser Arbeit vor allem zur Verifikation von Laufzeitmessungen für SPYDER-CORE-P1. Eine Beschreibung der Softwaretools für diese Laufzeitmessungen sowie des Loaders folgt in Kapitel 5.

4.4 Echtzeitbetriebsystem VxWorks

VxWorks ist ein kommerzielles Softwareprodukt der Firma WindRiver. Gegenüber anderen Systemen wie RTEMS zeichnet es sich durch Features wie einen vollständigen TCP/IP Stack und vor allem durch eine sehr komfortable Entwicklungsumgebung aus.

Zur Programmentwicklung dient wiederum ein auf dem Entwicklungs-PC laufender Cross Compilers. Der Programmcode wird in Objektfiles übersetzt und zunächst *nicht* gelinkt. Diese Objektfiles können auf das Targetboard übertragen werden, indem für die Kommunikation das TCP/IP Protokoll benutzt wird. Als Übertragungsmedium dient wie bei SPYDER-CORE-P1 z.B. Ethernet, aber auch andere physikalische Medien sind möglich.

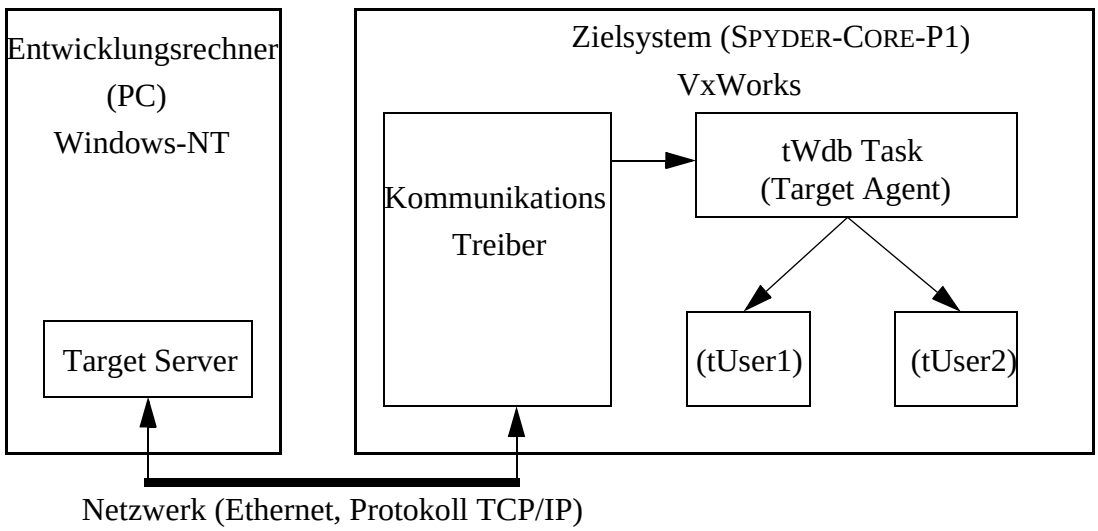


Abbildung 4.4 Entwicklungsumgebung für VxWorks (Quelle: [KW99])

Um diese Kommunikation zu realisieren, findet ein sogenannter *Target-Server* Verwendung. Dieser kommuniziert mit einem auf dem Zielsystem laufenden *Target-Agent*. Der Target-Agent läuft als spezielle Task unter VxWorks. In Abbildung 4.4 ist dieses Prinzip schematisch dargestellt.

Damit der Bootloader nach dem Einschalten zur Verfügung steht, muß er sich neben anderen wichtigen Initialisierungsroutinen auf einem nichtflüchtigen Speichermedium befinden, z.B einem EPROM oder Flash-Memory. Die Spyder-Architektur bietet beide Varianten für den initialen Start des Systems an. Target-Server und Target-Agent erlauben, das eigentliche Betriebsystem über eine Netzwerkschnittstelle zu booten. Desweiteren ist es möglich, die anfangs erwähnten Objektfiles *zur Laufzeit* in das Betriebssystem einzubinden. Ebenso ist der gezielte Aufruf von implementierten C-Funktionen durch eine Art Interpreter möglich.

Die gesamte Funktionalität wie das Nachladen von Objektfiles, Messung des Ressourcenverbrauchs der laufenden Tasks usw. ist unter einer einheitlichen Bedienoberfläche zusammen-

gefaßt. Während das eigentliche Betriebsystem den Namen VxWorks trägt, wird das gesamte Paket aus Entwicklungsumgebung und RTOS unter der Bezeichnung *Tornado* vertrieben.

Kapitel 5

Implementierung

In diesem Kapitel möchte ich die von mir entwickelte Softwareumgebung für das SPYDER-CORE-P1 Emulationsboard beschreiben.

Beginnen werde ich mit der Vorstellung eines Pakets von Diagnosetools, mit dessen Hilfe man auf der Emulationsumgebung testen kann, ob alle Hardwarekomponenten fehlerfrei arbeiten. Anschließend gehe ich auf ein Testverfahren ein, mit dessen Hilfe die Performance der eingesetzten Hard- und Softwarekomponenten meßbar gemacht wird. Es folgt die Beschreibung eines Tools für Entwickler, die das FPGA mittels Compile-Time Rekonfiguration nutzen. Weiterhin werde ich darlegen, wie sich SPYDER-CORE-P1 als Entwicklungs- und Forschungsplattform für Run Time Konfiguration verwenden läßt.

5.1 Diagnosesoftware

Das vorliegende Paket von Diagnosetools ermöglicht einen umfassenden und systematischen Test der Funktionalität eines Spyder-Core Mikroprozessorsystems. Das Paket besteht aus neun Komponenten. Die Konzeption des Testsets lehnt sich dabei an der Initialisierungs-Sequenz von VxWorks an. So ist es möglich, zunächst rudimentäre Systemfunktionen wie den Adreß- und Datenbus zu prüfen, bevor weitere Baugruppen betrachtet werden. Jedes Testtool hat die Funktionsüberprüfung *einer* bestimmten Hardwarekomponenten zum Gegenstand. Bei der Spezifikation wurde vor allem darauf Wert gelegt, daß sich diese Komponenten möglichst unabhängig voneinander testen lassen, um eine Fehlerquelle eng eingrenzen zu können. Alle Tools liegen als EPROM Images vor, sind also ohne weitere Hardwarevorraussetzungen einsetztbar. Für einige Tools sind es jedoch erforderlich, externe Meßgeräte zu benutzen.

5.1.1 Grundtest 1

Durch den Einsatz dieses Programms kann man feststellen, ob der Prozessor und der Systembus prinzipiell arbeitsfähig sind. Dazu führt es folgendes minimales Assemblerprogramm aus:

```
#####################################################################
# SIMPLE 1 ( Carsten Nitsch, Feb. 1999)
#
#  This .text region should be loaded at 0xFFFFFF00 in order to
#  map the branch instruction at 0xFFFFFFFC.
#####################################################################
  .globlrvector
  .text

  .space0xFC
rvector:
        ba rvector
```

Durch spezielle Metabefehle wird der Assembler veranlaßt, eine Branch Anweisung an der Adresse 0xFFFFFFFC abzulegen, welche einen Sprung auf genau diese Adresse ausführt. Die eben beschriebene Adresse ist die Einsprungstelle für Prozessoren der Power PC GA und GXC Serie, d.h. nach einem Power-On-Reset verarbeitet die CPU den an dieser Stelle liegenden Befehl zuerst. Der Mikrocontroller wird also auf dieser Adresse eine Endlosschleife abarbeiten.

Im folgenden werde ich erläutern, auf welche Weise dieses Programm zum Funktionstest der Hardware eingesetzt werden kann. Dazu schließt man einen Logic Analyzer an den Adreß- und Datenbus des Systems an. Beim Datenbus kann man sich auf die Auswertung der unteren acht Datenbits[1] beschränken, da das EPROM ein 8-Bit-Device darstellt. Zusätzlich muß auch das Chip-Select Signal aufgenommen werden.

Der Logic-Analyzer wird so konfiguriert, daß er die Daten- und Adreßsignale als *Bus* aufnimmt. Der Trigger wird auf die fallende Flanke des Chip-Enable Signals eingestellt, da dieses Signal low-aktiv ist.

Wird nach einem Reset der CPU am Logic-Analyzer ein Trigger-Event ausgelöst, so zeichnet dieser die Aktivität auf den Bussen auf. Im fehlerfreien Fall muß die Aufzeichnung zeigen, daß Daten- und Adreßbus nacheinander wie folgt belegt waren:

1. Beim Power PC trägt das niederwertigste Bit die Nummer 31. Das niederwertigste Datenbit ist also D_{31} und *nicht* D_0.

Takt	Adreßbus $A_{31}..A_0$	Datenbus $D_{31}..D_{24}$
1	0xFFFFFFFC	0x4B
2	0xFFFFFFFD	0xFF
3	0xFFFFFFFE	0xFF
4	0xFFFFFFFF	0xFE

Tabelle 5.1: Fetching

Das Wort 0x4BFFFFFE ist die binäre Codierung des Sprungbefehls.

Auswertung:

- Am Logic Analyzer wird ein Trigger ausgelöst und die Busaktivitäten verhalten sich wie in Tabelle 5.1 dargestellt. In diesem Fall kann man davon ausgehen, daß die CPU funktioniert und wahrscheinlich keine groben Fehler (z.B. Kurzschlüsse) vorhanden sind. Eine vollständige Funktionsprüfung für die Busse (insbesondere Treiberbausteine) ist durch dieses Tool aber noch nicht möglich.

- Am Logic Analyzer wird kein Trigger ausgelöst oder keine Busaktivitäten aufgezeichnet. In diesem Fall liegt ein ernster Fehler vor. In diesem Fall sollte an den LA[1]direkt am Steckerkranz[2] anschließen, welcher die CPU umschließt. Sind jetzt Busaktivitäten meßbar, liegt die Fehlerursache möglicherweise in einem fehlerhaften Treiberbaustein. Treiberbausteine sind Chips, welche Peripheriegeräte wie EPROMs mit der CPU verbinden.

- Die vom LA aufgezeichneten Bussignale sind falsch. Zur weiteren Fehleranalyse muß man die gemessenen Werte mit denen aus Tabelle 5.1 bitweise vergleichen. Dabei sollte man vor allem darauf achten, ob einzelne Signale permanent auf High oder Low liegen. Ein solches Verhalten gibt Hinweise auf die mögliche Fehlerursache, z.B. Kurzschlüsse zwischen den Signalen oder Leitungsunterbrechungen. Oft findet man den Fehler in Form von kalten oder unsauberen Lötstellen, die sich auch bei sorgfältiger Arbeit nie ganz ausschließen lassen. Auch ein defekter Treiberbaustein kann die Quelle solcher Bitfehler darstellen.

Dieses Testprogramm eignet sich also vor allem zum Aufspüren von Fehlern auf den Bussen. Da diese Bitfehler eine häufige Ursache für „totale“ Ausfälle sind, ist es trotzt seine minimalen Umfangs ein nützliches Tool, mit dessen Hilfe schon einige Systeme wieder funktionstüchtig gemacht werden konnten.

1. Logic Analyzer
2. Der sogenannte Steckerkranz stellt alle Signale des Daten- und Adreßbusses für Messungen zur Verfügung

5.1.2 Grundtest 2

Dieses Tool überprüft alle Adreßleitungen auf ihre Funktionalität, welche für die Funktion des EPROMS notwendig sind. Der für das EPROM nutzbare Adreßraum ist 20 Bit breit, so daß die Leitungen $A_{31}..A_{12}$ relevant sind.[1]

Das Programm springt zu einer Adresse, welche am Beginn des EPROM Speicherraums steht. Bei einem 128 kByte EPROM liegt diese Stelle auf der Adresse 0xFFFFE000 im Adreßraum des Power PC. Das Tool initialisiert das Maschinen-Status-Register (MSR) und invalidiert die Befehls- und Datencaches. Weiterhin schaltet es alle Interrupts aus.

Nach dieser Initialisierung des Mikroprozessors arbeitet das Testprogramm eine Endlosschleife ab. Dabei liest es fortlaufend den Adreßbereich des EPROM aus.

Auswertung: Um das Tool zur Analyse der Spyder-Core Hardware benutzen zu können, benötigt man wiederum einen Logic Analyzer. Um einen Überblick über den Zustand der Hardware zu erhalten genügt aber auch ein Speicheroszilloskop. Wenn man das Timing auf dem Adreßbus aufzeichnet, so ist im fehlerfreien Falle folgendes Resultat zu erwarten:

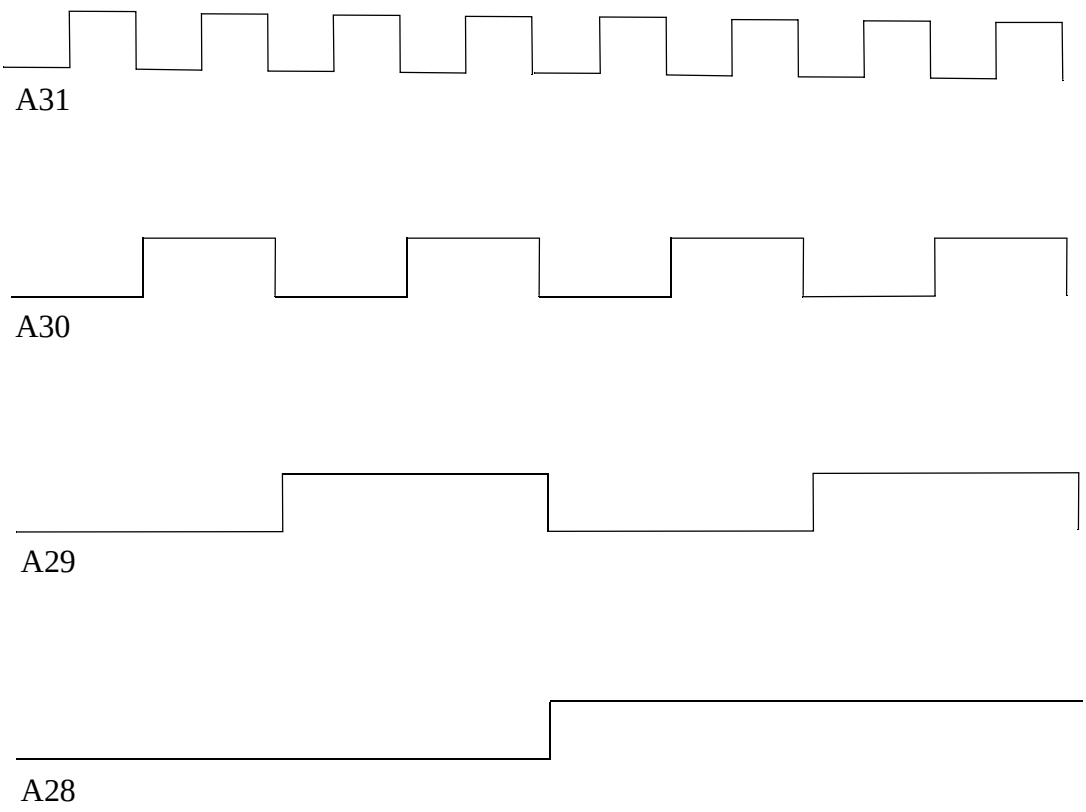


Abbildung 5.1 Busaktivitäten

1. Beim Power PC bezeichnet das Bit mit dem Index 0 das höchstwertigste Bit.

Wichtig ist, daß auf *allen* Adreßleitungen (A_{31} bis A_{12}) Pegelveränderungen sichtbar sein müssen. Zeigen einzelne Leitungen permanente Pegel, so liegt ein Fehler vor, wie er schon in Abschnitt 5.1.1 beschrieben wurde. z.B ein defekter Treiberbaustein.

Anpassung des Testprogramms:

- Der Beginn des Scanbereichs, im Programmcode mit *_reset* bezeichnet, muß am Beginn des Adreßbereichs des EPROM liegen. Dazu ist das Linkfile[1] (corep1.lnk) entsprechend zu editieren. Das folgende Beispiel zeigt die Konfiguration für ein 128 kByte EPROM:

```
MEMORY
 {

     rom:   org = 0xfffe0000, len = 0x1FF00 /* 128 kByte EPROM */
     reset: org = 0xffffff00, len = 0x100
 }

 SECTIONS
 {
     .init (TEXT) : { rvect403.o(.text) } > reset
     .vtbl (TEXT) : { init.o(.text) }     > rom
 }
```

Für andere EPROMs lauten die Parameter z.B:

```
rom:   org = 0xFFFC0000, len = 0x3FF00 /* 256 kByte */

rom:   org = 0xFFF80000, len = 0x7FF00 /* 512 kByte */
```

- IBM 403 GA / GCX CPUs unterscheiden sich u.a in der Größe ihrer Cache Lines.

  GA CPUs haben 32 Cache Lines Data Cache und 64 Lines Instruction Cache, während GCX CPUs über 256 Cache Lines Data Cache und 512 Lines Instruction Cache verfügen können.

  Die Definitionen für DATA_CACHE_LINES / INSTRUCTION_CACHE_LINES sind in init.s anzupassen.

1. Ein Linkfile enthält die Informationen darüber, in welche Adreßbereiche ein Programm bzw. Teile davon geladen werden sollen

5.1.3 Grundtest 3

Diese Programm dient darüber, sich einen schnellen Überblick über die Funktion des EPROMS und der 8 bit Bank zu verschaffen. Die sogenannte 8 Bit Bank ist ein für die Kommunikation mit peripheren Geräten konfigurierter Adreßbereich des Power PC. Die Kommunikation erfolgt elektrisch dabei über die bereits erwähnten Treiberbausteine. Über die 8-Bit Bank sind u.a. eine serielle Schnittstelle, das Ethernet Interface und 2 Leuchtdioden ansprechbar.

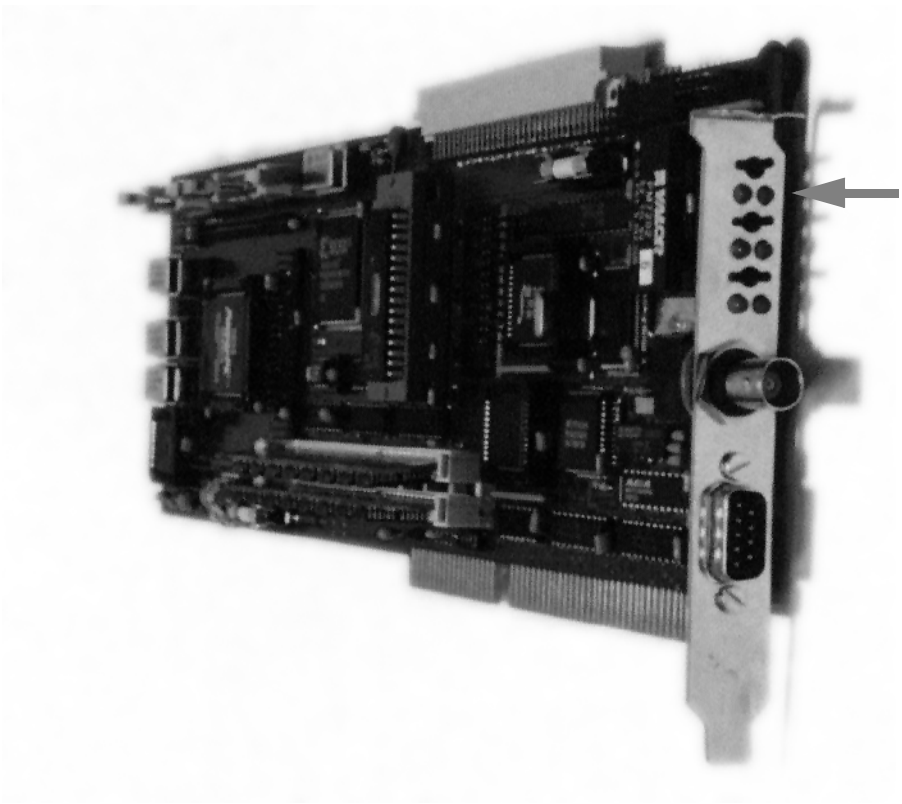


Abbildung 5.2 SPYDER-CORE-P1 mit LEDs

Das Tool initialisiert das MSR[1], invalidiert die Daten- und Befehlscaches und schaltet alle Interrupts ab. Darüber hinaus konfiguriert es einen Teil des Power PC Adreßraums, daß dieser als 8-Bit I/O-Bank benutzt werden kann.

Am Ende versucht das Programm, eine Leuchtdiode einzuschalten. Die betroffene LED ist in Abbildung 5.2 durch einen Pfeil gekennzeichnet. Zur Aktivierung der LED führt das Tool einen Schreibzugriff auf die Adresse 0xF2800000 aus und verbleibt dabei in einer Endlosschleife. Der folgende Programmausschnitt verdeutlicht das dieses Prinzip:

1. Maschinen Status Register

```
forever:

  lis r2, 0xF280 # Switch on green LED and wait for ever
  ori r2, r2, 0x0000
  lis r1, FFFF
  ori r1, r1, FFFF
  stwx r1, 0, r2
  ba forever
```

Abbildung 5.3 Grundtest 3

Ein spezieller programmierbarer Logikbaustein (Mach210A der Firma AMD) erkennt diese Zugriffe und generiert ein Signal, welches die Leuchtdiode einschaltet (LED Chip-Select).

Auswertung:

- Nach dem Einschalten oder nach einem Reset des Prozessors leuchtet die grüne LED auf. In diesem Falle kann man davon ausgehen, daß der Adreßbus und der Datenbus funktionsfähig sind sowie das EPROM und die Treiberbausteine korrekt arbeiten. Desweiteren ist kein Fehler beim Mach feststellbar.

- Bleibt die LED dunkel, so können mehrere Fehlerquellen die Ursache dafür sein. Eine Möglichkeit ist, daß ein Fehler beim EPROM, Bus bzw. den Treiberbausteinen zu finden ist. Für eine genauere Untersuchung benutzt man dann die Tools *Grundtest 1* und *Grundtest 2*.

  Funktionieren diese Komponenten, so sollte man den Mach-Baustein näher untersuchen. Zu diesem Zweck benutzt man am besten ein Oszilloskop. Dann überprüft man, ob am Pin, welcher die LED treibt, ein Low-Pegel zu messen ist (das LED Chip-Select ist low-aktiv). Ist dies der Fall, so ist entweder die LED defekt oder es besteht ein anderer Fehler, wie z.B. Kurzschluß mit einem anderen Signal bzw. eine Leitungsunterbrechung. Findet man kein korrektes Signal, liegt der Fehler wahrscheinlich beim Machbaustein selbst, der entweder defekt oder falsch programmiert ist. Auch Kontaktprobleme innerhalb seines Sockels können eine fehlerfreie Funktion verhindern.

Anpassung des Testprogramms:

Das Programm ist an die Größe des verwendeten EPROMS anzupassen. Weiterhin muß berücksichtigt werden, welche Mikroprozessor Architektur (403GA oder 403GCX) benutzt wird. Diese Konfiguration geschieht analog wie in Abschnitt 5.1.2 beschrieben.

5.1.4 DRAM Test

Diese Tool überprüft das gesamte dynamische RAM, welches sich auf dem Spyder-Core Emulationsboard befindet, auf korrekte Funktion. Im folgenden möchte ich die Arbeitsweise dieses Prüfalgorithmus kurz beschreiben

Das Programm verzweigt zunächst zum *_reset* Vektor, welcher am Beginn des Adreßbereichs des EPROMs liegt. Wird ein 128 kByte EEPROM verwendet, liegt diese Einsprungadresse bei 0xFFFE0000. Zunächst wird das Maschinenstatusregister (MSR) initialisiert, die Caches invalidiert und die Interrupts abgeschaltet. Anschließend erfolgt eine Initialisierung der 8-Bit I/O Bank (siehe auch Abschnitt 5.1.3) sowie der zu untersuchenden DRAM-Bank selbst.

Der eigentliche Test läuft wie folgt ab:

- Zuerst schreibt das Tool die Testmuster 0xaaaaaaaa und 0x55555555 an die erste Adresse des zu scannenden DRAM-Adreßraums und liest diese Werte zurück. Schlägt dieser *Patterntest* fehl, so arbeitet der Datenbus nicht korrekt.

- Danach wird ein sogenannter *Hausnummerntest* durchgeführt:

  Die Adresse einer Speicherstelle wird in selbige geschrieben

  Danach wird sie wieder ausgelesen und verglichen, ob ihr Inhalt mit dem Wert übereinstimmt, der hineingeschrieben wurde.

Auf diese Weise wird der zu testende DRAM-Adreßraum durchgescannt. Dabei werden jeweils erst eine bestimmte Anzahl von Adressen beschrieben und danach ausgelesen.

Wurde sowohl der Pattern- als auch der Hausnummerntest erfolgreich durchlaufen, wird die grüne LED eingeschaltet. Der Test beginnt anschließend von vorn und wird beliebig oft wiederholt. Somit sind auch Fehler erfaßbar, welche erst nach längerer Zeit auftreten, sei es durch thermische oder andere Ursachen.

Wird irgendwann im Verlauf des Tests ein Fehler erkannt, so wird die rote LED eingeschaltet und das Programm verweilt in dieser Stelle in einer Endlosschleife, d.h. der Testalgorithmus hält. Somit kann z.B. mittels eines Logic Analyzers festgestellt werden, an welcher Adresse der Fehler auftrat und somit Rückschlüsse auf die Ursache gezogen werden (z.B. Kurzschlüsse zwischen Adreßleitungen). Der Programmcode selbst läuft natürlich vollständig im EPROM, so daß das Programm auch bei einem fehlerhaften DRAM Baustein lauffähig bleibt.

Auswertung: Ich möchte an einem Beispiel erläutern, auf welche Weise Fehler des DRAM mit Hilfe dieses Diagnosetools erkannt werden können:

Nehmen wir an, am Datenbus des Power PC sind die Leitungen DP_8 und DP_9 kurzgeschlossen. Die Signale DP_8 und DP_9 des PPC entsprechen den Bits D_{23} und D_{22} eines Wortes der Länge von 32 Bit.

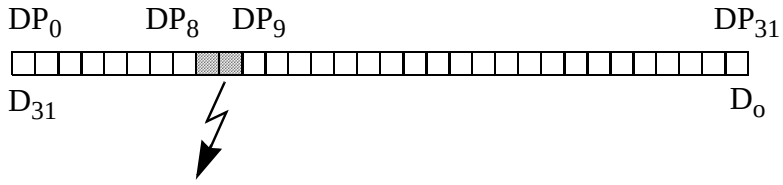


Abbildung 5.4 Fehler auf Datenbus

Während des Testprogramms wird DP_9 (D_{22}) zum ersten mal benutzt, wenn ein Zugriff auf die Adresse 0x400000 (4 MByte) erfolgt. Wegen des Kurzschlusses kommt es zu einem Fehler. Dieser wird vom Programm erkannt und die rote LED aktiviert.

Anpassung des Testprogramms:

- Die Konfiguration der Einsprungadresse *_reset* sowie der Cache Settings erfolgen wie in den vorangegangenen Abschnitten beschrieben.
- Weiterhin ist auf die richtigen Settings für die DRAM-Bank zu achten. Diese Einstellungen sowie die Einblendung der DRAM Bank in den Adreßraum des PPC wird in der Datei *bcr.inc* beschrieben:

```
; Spezifiziere Device Control Register BR7: DRAM 32 Bit
; Speicherbereich: DRAM Bank7 Simm-Module( 4 MB *32) 16 MB SIMM
; Address-Range:0x0000_0000 to 0x00FF_FFFF (Region  0) und
;  0x8000_0000 to 0x80FF_FFFF (Region 16)
;
; 31 | 30 29 28 | 27 26 25 24 23 22 21 20 | Address Lines
;  0 |  1  2  3 |  4  5  6  7  8  9 10 11 | Address Lines PPC
;  x |  0  0  0 |  0  0  0  0  0  0  0  0 | Settings
;

BAS_7           .equ     0x0  << 24     ; BAS = A4-A11
BS_7            .equ     0x4  << 21     ; BS = 16 MByte
...
SD_7            .equ     0x0  << 0      ; SD = DRAM

BR_7            .equBAS_7 | BS_7 | ... |SD_7
```

- Im Programm selbst wird der zu testende DRAM-Adreßbereich von den Konstanten *SCANSTART* und *SCANSIZE* bestimmt. Die Größe der Speicherbereichs, der am Stück beschrieben und anschließend zuammenhängend wieder gelesen wird, wird durch *BLOCKSIZE* definiert.

5.1.5 DRAM Test mit DPRAM[1]-Interface

Diese Programm hat wie der DRAM-Test den Zweck, den dynamischen Speicher auf eventuelle Fehler hin zu untersuchen. Die Arbeitsweise ist analog zum DRAM-Test, ebenso die Konfiguration des Tools.

Um die Benutzung des Tools komfortabler zu gestalten, ist in dieser Version des Speichertests ein einfaches Interface implementiert, welches die Kommunikation mit dem User ermöglicht. Die Spyder-Core Hardware bietet als Kommunikationsschnittstellen verschiedene Hardware an. So sind auf dem Board serielle Schnittstellen und ein Ethernet-Interface integriert. Beide Schnittstellen bedürfen jedoch einer Initialisierung und können selbst fehlerhaft sein. Daher habe ich mich für einen anderen Weg des Datenaustausches entschieden:

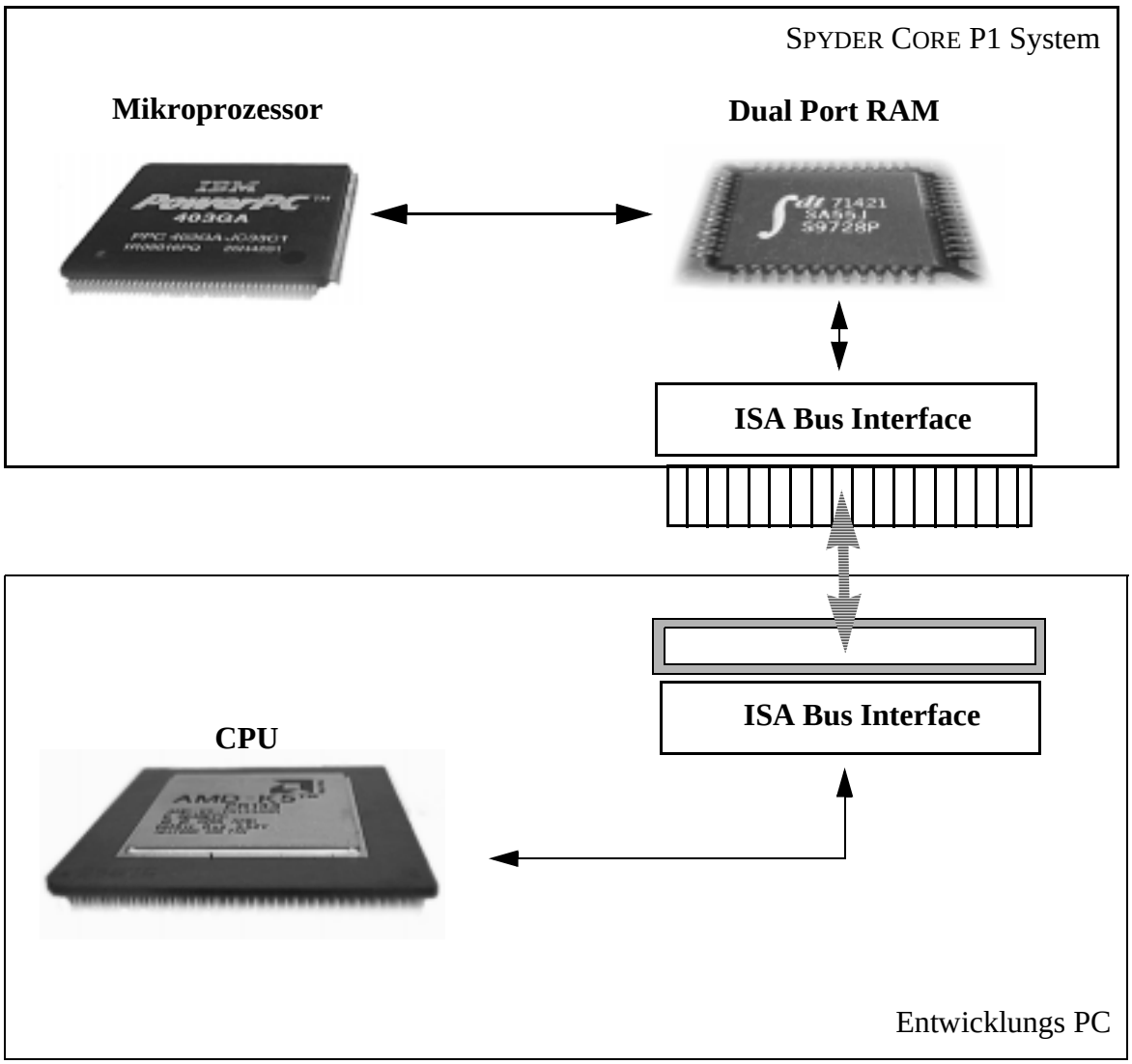


Abbildung 5.5 Dual Port RAM Interface

1. Dual Port RAM

Das Testprogramm benutzt das sogenannte Dual-Port-RAM (DPRAM) für die Kommunikation mit dem Entwicklungsrechner. Dabei hat sowohl der Mikroprozessor auf dem Spyder-Core System als auch die CPU des PC Zugriff auf einen bestimmten Speicherbereich. Dieser Bereich beginnt im Adreßraum des Power PC ab Adresse 0xF1000000. Vom PC aus ist er ab einer bestimmten Segmentadresse aus lesbar, welche mittels DIP-Schalter am Spyder System einstellbar ist. Voreingestellt ist dabei die Adresse 0xD000. Bevor der DRAM Test selbst beginnt, wird das DPRAM einer Prüfung unterzogen. Die Vorgehensweise ist dabei identisch zum Testalgorithmus des DRAM-Tests.

Tabelle 5.2 zeigt, welche Informationen vom Testprogramm über das Dual-Port-RAM bereitgestellt werden.

Nachricht	**Adresse auf Spyder-Core-P1**	**Adresse auf Entwicklungs PC[a]**	**Beschreibung**
Counter DPRAM	0xF1000000	D000:0000	Dual-Port-RAM Adresse[b], welche gerade geprüft wird
Message DPRAM	0xF1000004	D000:0004	Resultat des DPRAM-Tests als ASCII String
Counter DPRAM	0xF1000010	D000:0010	DRAM-Adresse, welche im Moment getestet wird
Message DRAM	0xF1000014	D000:0014	Ergebnis des DRAM-Tests als ASCII String

Tabelle 5.2: Dual Port ROM Interface: Adressen

a. andere Segmentadressen sind am DIP-Schalter einstellbar
b. Integerwerte wie Counter werden im besser lesbaren Big-Endian-Format gespeichert

Auf dem PC kann man das Dual-Port-RAM wie folgt einsehen:

- Befehl *isa -a*[1]eingeben
- Das DOS-Programm *debug* starten. Dort kann man sich nun den Speicherinhalt betrachten. Um einen Speicherauszug ab Adresse D000:0000 anzeigen zu lassen, muß man unter debug auf der Kommandozeile den Befehl *d d000:0000* eingeben.

Das folgende Beispiel veranschaulicht, wie die Informationen in Speicher abgelegt sind. Die Segmentadresse ist hier 0xD000. Der Speicherabzug entstand, während der DRAM-Test lief. Dabei war der Check des Dual-Port-RAM bereits abgeschlossen, was man an der Statusmeldung *DPRAM OK* erkennen kann.

1. *isa* ist ein DOS Programm, welches zum Lieferumfang des Spyder-Core Systems gehört. Es wird auf dem Entwicklungsrechner gestartet und aktiviert das ISA-Interface des Spyder Boards.

```
D000:0000  F1 00 07 FC 20 44 50 52-41 4D 20 4F 4B 20 20 20   .... DPRAM OK
D000:0010  00 3B C4 E0 F1 00 00 14-F1 00 00 18 F1 00 00 1C   .;..............
D000:0020  F1 00 00 20 F1 00 00 24-F1 00 00 28 F1 00 00 2C   ... ...$...(...,
D000:0030  F1 00 00 30 F1 00 00 34-F1 00 00 38 F1 00 00 3C   ...0...4...8...<
D000:0040  F1 00 00 40 F1 00 00 44-F1 00 00 48 F1 00 00 4C   ...@...D...H...L
```

Abbildung 5.6 Speicherauszug durch DOS-Programm Debug

Weiterhin lassen sich folgende Informationen erkennen:

- Die zuletzt getestete Speicheradresse des DPRAM war *0xF10007FC* im Adreßraum des Power PC. Im Dual-Port-RAM wird diese Information ab der Adresse D000:0000 abgelegt.
- Der DRAM-Test läuft gerade, im Moment wird die Adresse *0x0003BC4E0* überprüft, zu finden im DPRAM ab D000:0010

5.1.6 Test des On-Chip-UART

Die Prozessoren der PPC403 Serie sind speziell für den Einsatz in eingebetteten Systemen entworfen worden. Daher verfügen sie über einige auf dem Chip integrierte Schnittstellen, wie z.B. ein serielles Interface (UART). Der Test dieses UARTs ist Aufgabe des vorliegenden Diagnosetools.

Im Gegensatz zu den in den vorangegangenen Abschnitten beschriebenen Programmen ist dieses Tool nicht mehr vollständig in Assembler programmiert, sondern bedient sich der Hochsprache C. Daher sind neben den bereits mehrfach beschriebenen Initialisierungsroutinen einige weitere Konfigurationsschritte notwendig. Zu diesen zählt vor allem das Setzen des Stackpointers. Desweiteren ist es natürlich notwendig, die serielle Schnittstelle selbst zu initialisieren. Dazu werden die entsprechenden Register des PPC403 programmiert:

Die wichtigsten Parameter sind in Tabelle 5.3 festgehalten

.

Parameter	Wert
Baudrate	19200
Hardware Hand Shaking	aktiv
Startbits	1
Stopbits	8
Parity	keine

Tabelle 5.3: Parameter für Schnittstellentest (On-Chip-UART)

Im weiteren Verlauf dieses Abschnitts möchte ich die Arbeitsweise dieses Schnittstellentests kurz erläutern. Im wesentlichen schreibt das Tool in einer Endlosschleife Bytes auf die serielle Schnittstelle und liest diese wieder ein:

```
forever
{
  write x;
  read y;
  if ( x != y) then Error();
}
```

Abbildung 5.7 Schnittstellentest (Pseudocode)

Tritt während des Schreibens oder Lesens ein Timeout auf oder unterscheiden sich die geschriebenen von den zurückgelesenen Werten, so wird die rote LED auf dem Board eingeschaltet. Das Programm unterbricht danach seine weitere Abarbeitung und führt nur noch eine Endlosschleife aus.

Damit die Daten, welche vom Programm über die serielle Schnittstelle ausgegeben wurden, wieder eingelesen werden können, müssen folgende Vorraussetzungen erfüllt sein:

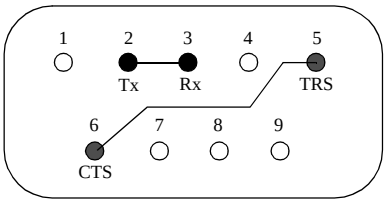


Abbildung 5.8 Loop-Back-Beschaltung einer 9-PIN Sub-D Buchse

Da kein interner Loop-Back-Modus benutzt wird, müssen die entsprechenden Pins an der Sub-D-Buchse, die sich auf SPYDER-CORE-P1 befindet, miteinander verbunden werden. Neben dem seriellen Ausgang (Tx) und Eingang (Rx) sind auch die Steuerleitungen TRS und CTS zu verbinden, damit das Hardware-Hand-Shaking fehlerfrei funktionieren kann. Abbildung 5.8 zeigt eine schematische Beschreibung der Loop-Back-Beschaltung.

Fehlersimulation: Dieses Testprogramm einschließlich der Loop-Back-Beschaltung läßt sich relativ leicht auch seine Funktionalität hin überprüfen. Zunächst startet man es. Ist die serielle Schnittstelle in Ordnung und die Loop-Back-Beschaltung einwandfrei, so wird die grüne LED aufleuchten. Stört man die Datenübertragung, z.B. durch Unterbrechen einer Loop-Back Verbindung, so wird dies durch das Aufleuchten der roten LED signalisiert. Die rote LED leuchtet auch nach Wiederherstellen der Verbindung weiter.

Anpassung des Testprogramms: Die Konfigurierung der Parameter für die EPROM-Größe und den verwendeten Power-PC geschieht wie in Abschnitt 5.1.1 beschrieben. Voreingestellt das Tool für ein 128 kByte großes EPROM. Es gibt sowohl für den PPC403GA als auch den 403GCX eine lauffähige Version.

Die Parameter für die serielle Schnittstelle bedürfen eigentlich keiner Änderung. Die Voreinstellungen sind in Tabelle 5.3 zu sehen.

5.1.7 Test der zweiten seriellen Schnittstelle

Neben der auf dem Power PC Chip integrierten seriellen Schnittstelle ist auf dem Spyder-Core Board noch ein zweites serielles Interface vorhanden. Zum Einsatz kommt dabei ein PC16550D Controller von National Semiconductor.

Der Chip ist über die 8-Bit I/O Bank in den Speicherraum des Power PC integriert, so daß seine Register über entsprechende Adressen schreib- bzw. lesbar sind. Eine ausführlichere Erklärung der I/O Bank findet sich in Abschnitt 5.1.3.

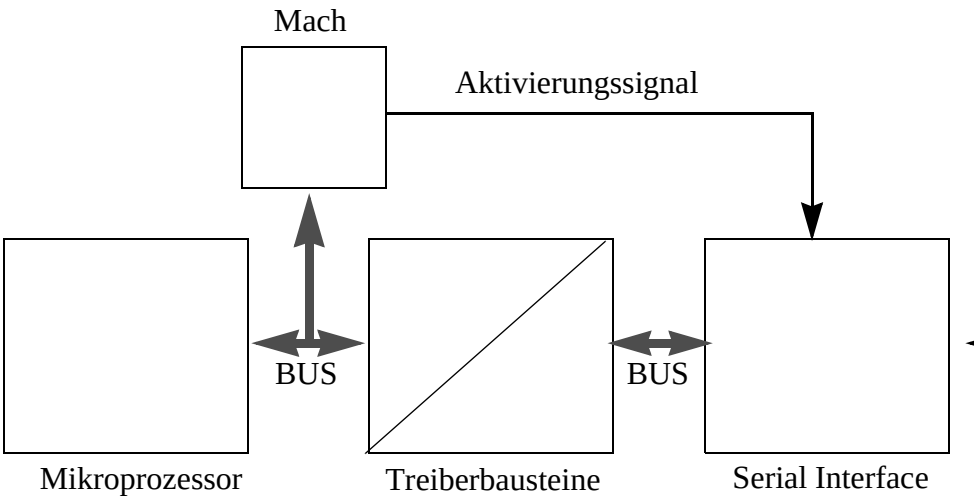


Abbildung 5.9 Externe serielle Schnittstelle als Bestandteil der I/O Bank

Der Testablauf ist identisch zu dem Algorithmus, wie er bereits im Abschnitt über den Test des On-Chip UARTs beschrieben wurde. Es werden auch hier Daten auf der seriellen Schnittstelle ausgegeben, zurückgelesen und miteinander verglichen.

Die Konfigurierung des Bausteins erfolgt analog wie in Abschnitt 5.1.6. Allerdings ist diesmal das Hardware Handshaking nicht aktiviert, wie Tabelle 5.4 zeigt.

Parameter	Wert
Baudrate	19200
Hardware Handshaking	nicht aktiv
Startbits	1
Stopbits	8
Parity	keine

Tabelle 5.4: Parameter für Schnittstellentest (externer UART)

Die notwendige Loop-Back-Beschaltung für die externe serielle Schnittstelle zeigt Abbildung 5.10. Die Pins 2 und 3 am Pfostenstecker[1] sind entsprechend zu verbinden.

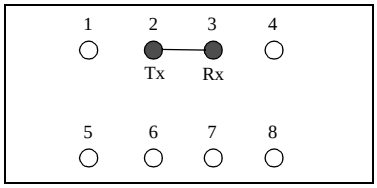
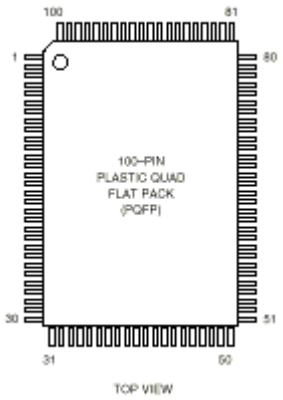


Abbildung 5.10 Loop-Back Beschaltung für den Test der externen seriellen Schnittstelle

1. Für die zweite serielle Schnittstelle ist auf dem Spyder Board keine Sub-D Buchse vorgesehen.

5.1.8 Ethernet Interface

Dieses Tool testet die Funktionsfähigkeit des Ethernet Interface, welches auf SPYDER-CORE-P1 integriert ist



. Die Ethernetfunktionalität wird dabei von einem Controllerchip vom Typ MB86960 bereitgestellt, welcher von der Firma Fujitsu hergestellt wird. Der MB86960 ist ein hochintegrierter Schaltkreis, auf welchem alle Komponenten integriert sind, die für Pufferverwaltung, Manchester Encoding/Decoding, Kollisionsverwaltung usw. notwendig sind.

Wie der externe UART ist auch der Ethernet-Chip über die 8-Bit I/O Bank in das System integriert. Zur Ankopplung an das Transportmedium, z.B. 10Base2 Koaxialkabel, dient ein weiterer Tranceiverbaustein. Gegenstand dieses Testprogramms ist aber nur der MB86960 Controller selbst, welcher in einem speziellen Testmode betrieben wird.

Abbildung 5.11 MB86960 ([Fu97])

Bevor der eigentliche Schnittstellentest beginnen kann, muß der Chip durch geeignete Parameter initialisiert werden. Die folgende Tabelle veranschaulicht dies.

Parameter	Einstellung	Betroffene Register[a]
Transmit- und Receive Interrupts	deaktiviert	xint, rint
Receive Mode	Alle pakete akzeptieren	rmode
Transmit Mode	Loop Back aktiv	xmode
Transmitter/Receiver Konfiguration	Wortlänge für Bus und Puffer ist 8 Bit	conf0

Tabelle 5.5: Konfiguration des MB86960

a. Die Registerbezeichnungen sind an den Quellcode dieses Programmes angelehnt.

Der Übersicht wegen möchte ich mich dabei auf die Beschreibung einiger ausgewählter Parameter beschränken, die für die Funktion des Testprogramm wesentlich sind. Auf eine ausführliche Beschreibung des Controllerchips und seiner Programmierung möchte ich an dieser Stelle aus Platzgründen ebenfalls verzichten und auf [Fu97] verweisen. Die vollständige Initialisierungssequenz für den hier benutzten Testmode findet sich im kommentierten Quellcode dieses Programms.

Der Testalgorithmus baut wieder auf dem Prinzip auf, Daten auszugeben, zurückzulesen und dabei zu verifizieren. Im Unterschied zu den Test für die seriellen Schnittstellen erfolgt die Datenübertragung jedoch blockorientiert. Die Länge dieser Blöcke ist konfigurierbar. Sie bestehen immer aus einer Folge von Bytes der Form 0x00, 0x01, ...

```
forever
{
  write Block_x;
  read Block_y;
  if ( Block_x != Block_y) then Error();
}
```

Abbildung 5.12 Testalgorithmus für den Test des Ethernet Interfaces

Die Verwendung von Blöcken erfolgt daher, weil die Architektur des vorhandenen Ethernet Interfaces eine blockorientierte Kommunikation vorgibt.

Für die Anwendung dieser Testsoftware sind keine weiteren Hardwaremodifikationen notwendig. Insbesondere fällt eine externe Loop-Back-Beschaltung weg. Das Zurücklesen der Daten wird im MB86960 durch einen speziellen Loop-Back-Mode ermöglicht, d.h. die gesendeten Blöcke werden intern in den Empfangspuffer umgeleitet.

Auswertung: Während des Tests blinkt die grüne LED. Stößt der Testalgorithmus während eines Lese- oder Schreibvorgangs auf einen Timeout, so wird dieser Fehler durch die permanente Aktivierung der roten LED angezeigt. Ebenso wird verfahren, falls die Daten der geschriebenen und zurückgelesenen Blöcke unterschiedlich sind.

Anpassung des Tools: Die im Programm benutzte Konfiguration bedarf eigentlich keiner Anpassung durch den Nutzer. Für die Anpassung an eine andere Zielplattform sind aber u.U. folgende Parameter von Interesse:

- Die Größe der *Datenblöcke* kann in der Datei *main.c* modifiziert werden, falls dies gewünscht wird:

  #define PACKETSIZE_LO <bit7..bit0>

  #define PACKETSIZE_HI <bit15..bit8>

- Die Basisadresse, ab welcher der Registerraum des Chips in den Adreßraum des Power PC eingeblendet ist, kann durch Modifikation der Konstanten *ETHERNET_BASE* angepaßt werden.

5.1.9 Vollständiger Systemtest

Dieses Diagnoseprogramm testet alle Komponenten, welche in den vorangegangenen Abschnitten beschrieben wurden (außer Grundtest1 bis 3). Nacheinander werden folgende Komponenten auf ihre Funktionalität hin überprüft:

- Dual Port RAM
- DRAM
- PPC403 On-Chip-UART
- externer UART (National Semiconductor PC16550D)
- Ethernetschnittstelle (Fujitsu MB 86960)

Es eignet sich vor allem dazu, einen schnellen Überblick zu bekommen, ob alle Komponenten des Spyder-Core System korrekt arbeiten. Wird ein Fehler festgestellt, versucht das Diagnoseprogramm, diesen zu lokalisieren.

Für die Kommunikation mit dem Benutzer greift das Programm auf das Dual-Port-RAM zurück. Diese Methode wurde bereits in Abschnitt 5.1.5 beschrieben. Die folgende Tabelle zeigt, welche Informationen im Dual-Port-RAM während des Tests abgelegt werden.

Nachricht	Adresse auf Spyder-Core-P1	Adresse auf Entwicklungs PC[a]	Beschreibung
Counter DPRAM	0xF1000000	D000:0000	Dual-Port-RAM Adresse[b], welche gerade geprüft wird
Message DPRAM	0xF1000004	D000:0004	Resultat des DPRAM-Tests als ASCII String
Counter DPRAM	0xF1000010	D000:0010	DRAM-Adresse, welche im Moment getestet wird
Message DRAM	0xF1000014	D000:0014	Ergebnis des DRAM-Tests als ASCII String
Message UART1	0xF1000020	D000:0020	Ergebnis des Tests der On-Chip-Uarts (ASCII-String)
Message UART2	0xF1000030	D000:0030	Ergebnis des Tests des externen UART (ASCII-String)
Message Ethernet	0xF1000040	D000:0040	Ergebnis des Test des Ethernetchips (ASCII-String)

a. andere Segmentadressen sind am DIP-Schalter einstellbar
b. Integerwerte wie Counter werden im besser lesbaren Big-Endian-Format gespeichert

Tabelle 5.6 Informationen im Dual-Port-RAM

Mit Hilfe des DOS-Programms DEBUG kann man feststellen, welche Resultate das Testprogramm geliefert hat. Wird z.B. ein Fehler im DRAM gefunden, so wird dies mit „DRAM

ERROR“ protokolliert (Message DRAM). Die Adresse, an welcher der Fehler auftrat, ist im Dual-Port-RAM unter „Counter DRAM“ abgelegt.

Wenn dies nicht eindeutig möglich ist, kommen zur detaillierten Fehleranalyse die einzelnen Testprogramme zum Einsatz. Dies ist insbesondere dann der Fall, wenn das Dual-Port-RAM selbst fehlerhaft ist und damit als Kommunikationsschnittstelle ausfällt oder das Board aufgrund eines Fehlers gar nicht mit der Programmausführung zu beginnen scheint. In diesem Fall ist dann der Einsatz von Meßequipment (Logic Analyzer) zusammen mit den Grundtest-Programmen unumgänglich.

5.2 Bewertung des Laufzeitverhaltens eines eingebetteten Systems

Wenn ein eingebettetes System harten Echtzeitbedingungen unterliegt, so ist die genaue Kenntnis seines Laufzeitverhaltens von entscheidender Bedeutung. Das System muß *stets* in der Lage sein, auf ein externes Ereignis innerhalb einer vorgegebenen Zeitspanne zu reagieren. In der Praxis sind oft Ereignisse auszuwerten, welche asynchron auftreten. So kann z.B. ein Sensor zu einem nicht vorhersagbaren Zeitpunkt einen Meßwert liefern, auf den das System mit entsprechenden Aktionen (Servicefunktionen) zu reagieren hat.

Implementiert werden die Servicefunktionen meist als Interruptroutinen. Auf diese Weise wird keine Systemleistung durch ein ständiges, periodisches Abfragen der Sensoren (Polling) verschwendet. Tritt ein Ereignis auf, so wird das gerade laufende Programm unterbrochen und die Interruptroutine aufgerufen.

In einem Multitaskingbetriebssystem laufen mehrere Routinen quasi parallel zueinander. Da aber Hardwareressourcen wie Speicher, Prozessoren usw. nur in endlicher Menge zur Verfügung stehen, müssen diese Ressourcen unter den Tasks aufgeteilt werden. Dies erzwingt in der Regel, daß eine Task nach einer bestimmten Zeit unterbrochen wird und die folgende Task zur Ausführung gelangt.

Somit ergeben sich zwei wesentliche Kriterien für die Bewertung des Echtzeitverhaltens eines Systems: Die *Interrupt-Reaction-Time* ist die Zeit die das System nach dem Auslösen des Interrupts braucht, bis es den ersten Befehl der ISR[1] ausführen kann. Weiterhin benötigt jeder Taskwechsel eine minimale Zeit (*Task-Switching-Time*). Da die Auswertung von externen Events oft auch den Aufruf entsprechender Kontrolltasks nach sich zieht, hat die Task-Switching-Time ebenfalls einen großen Einfluß auf das Zeitverhalten des Systems. Sowohl Interrupt-Reaction-Time als auch die Task-Switching-Time sind *unabhängig* vom Anwendungsprogramm. Diese Zeiten werden stets benötigt, selbst wenn die ISR oder die entsprechenden Kontroll-Tasks keinerlei Programmcode abarbeiten würden. Beide Konstanten werden sowohl von der Hardwarearchitektur als auch vom verwendeten Betriebssystem vorgegeben.

Das SPYDER-CORE-P1 System ermöglicht es, diese Konstanten meßtechnisch zu erfassen. Somit ist es dem Systementwickler möglich, exakt die Hardwarekomponenten zu wählen, mit denen das System den vorgegebenen Echtzeitbedingungen genügt, ohne durch „großzügige“ Auslegung der Performance unnötige Kosten zu verursachen.

Im Vorlauf dieser Arbeit erfolgten bereits Laufzeitmessungen mit dem Betriebsystem VxWorks sowie der Power PC 403 Serie. Um die erlangten Ergebnisse zu verifizieren, habe ich die Messungen unter dem RTOS RTEMS[2] wiederholt. RTEMS wurde während meiner Studienarbeit[3] von mir auf die Vorgängerversion von Spyder-Core portiert und im Rahmen dieser Arbeit an die Spyder-Core Architektur angepaßt. Um die Programme, welche mittels eines Cross Compilers übersetzt worden sind, auf das Target zu übertragen und dort auszuführen, habe ich weiterhin ein spezielles Ladeprogramm entwickelt, welches den ISA-Bus und das Dual-Port-RAM von Spyder-Core für die Kommunikation benutzt.

1. Interrupt Service Routine
2. Siehe auch Abschnitt 4.3
3. Eine ausführliche Dokumentation findet sich in [WSNRo99]

Im weiteren Verlauf dieses Abschnitts möchte ich die Arbeitsweise des Loaders beschreiben sowie die verwendeten Testprogramme vorstellen. Anschließend werde ich die erhaltenen Meßreihen kurz diskutieren.

5.2.1 Loader

Dieses Programm lädt Sektionen aus einem ELF-File[1] in den Hauptspeicher des Spyder-Core Systems. Dies geschieht unter Benutzung des Dual-Port-RAM von Spyder-Core. Diese spezielle Architektur erlaubt es, einen Teil des Speicherraums des Mikroprozessors, welcher sich auf dem eingebetteten System befindet, vom Entwicklungs-PC aus zugänglich zu machen.

Das Ladeprogramm nutzt diese Eingenschaft dahingehend aus, daß es zuerst den Code für ein minimales Bootprogramm in das Dual-Port-RAM schreibt. Weiterhin kopiert es 4 Byte Maschinencode auf den sogenannten Reset-Vektor des Power PC. Der Reset-Vektor ist die Adresse, die der Power PC nach einem Reset zuerst anspringt und den dort befindlichen Code abarbeitet. Die vier Byte Code stellen einen Sprungbefehl zum Beginn des Bootprogramms dar. Die Hardware von Spyder-Core stellt weiterhin ein spezielles Register bereit, mit dessen Hilfe sich ein Reset des Power-PC auslösen läßt. Dieses Register ist ebenfalls vom PC aus schreibbar. Ein programmierbarer Logikbaustein (Mach) wertet dazu die Signale aus, die über den ISA-Bus zur Spyder-Core Platine gelangen.

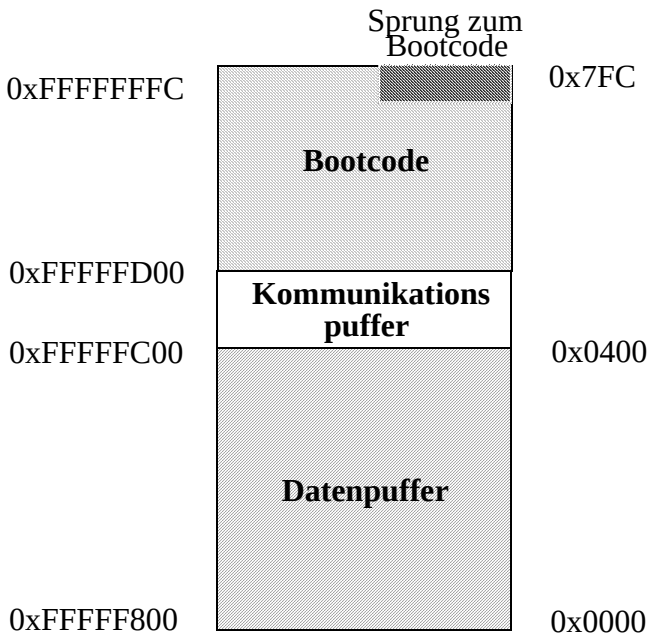


Abbildung 5.13 Nutzung des Dual-Port-RAM durch den Loader

1. Ein plattformübergreifendes Fileformat für Objektdateien und ausführbare Programme

Abbildung 5.13 zeigt, wie der Adreßraum im Dual-Port-RAM aufgeteilt wird. Auf der rechten Seite sind die Adressen eingetragen, über die der PC Zugriff auf das DPRAM hat. Diese Adressen sind dabei als Offsetadressen gegenüber einer Segmentadresse zu verstehen. Die Segmentadresse ist am Spyder Board mittels DIP-Schalter wählbar, die Voreinstellung ist 0xD000. Links sind die Adressen zu erkennen, mit denen vom Power PC aus auf die Daten zugegriffen werden kann. Dem aufmerksamen Leser wird auffallen, daß diese sich von den Adressen des DPRAM unterscheiden, welche in Abschnitt 5.1.5 beschrieben wurden. Die Ursache liegt darin begründet, daß das Spyder-Core System entsprechend modifiziert wurde, um mit dem Ladeprogramm zusammenzuarbeiten. Normalerweise ist der Adreßbereich ab 0xFFFFF800 dem EPROM oder Flash-RAM zugeordnet. Hier muß dieser Bootbereich aber im Dual-Port-RAM liegen. Da sowohl Flashspeicher als auch Dual-Port-RAM über die 8-Bit I/O Bank ansprechbar sind, genügte es, das Chip-Select-Signal des Flashspeichers dem Dual-Port-RAM zuzuordnen[1].

Nachdem am Power PC ein Reset ausgelöst wurde, beginnt er mit der Abarbeitung des Bootcodes. Im wesentlichen werden dabei die Speicherbänke initialisiert, die Cache Line invalidiert und die Caches abgeschaltet. Das Abschalten der Caches ist für die Funktion des Ladeprogramms im übrigen von essentieller Bedeutung, wie wir noch sehen werden.

Nach der Grundinitialisierung startet eine Kopierroutine auf dem Power PC, welche die Aufgabe hat, alle weiteren Segmente (Programmcode und Daten) der Applikation in das RAM von Spyder-Core zu laden. Dabei werden die zu kopierenden Daten blockweise in den Bereich des Dual-Port-RAM geschrieben, welcher in Abbildung 5.13 als *Datenpuffer* bezeichnet wird. Die Kopierroutine verschiebt sie von dort an die entsprechenden Adressen im DRAM. Der sogenannte *Kommunikationspuffer* dient dabei zur Übergabe von Argumenten. Diese Argumente spezifizieren z.B. die Größe der zu kopierenden Sektion und geben an, an welche Adresse im DRAM diese kopiert werden soll. Weiterhin dient der Kommunikationspuffer der Synchronisation zwischen der Kopierroutine auf dem Power PC und dem Ladeprogramm auf dem Entwicklungsrechner.

Sind alle Sektionen kopiert, werden die Caches aktiviert und die Bootroutine verzweigt zur Einsprungstelle der Applikation, wodurch diese automatisch gestartet wird. Der Bootcode mit Kopierroutine ist übrigens Bestandteil jeder für RTEMS gelinkten Applikation. Details dazu finden sich in [CN97].

Wenn man sich den Ladealgorithmus betrachtet, wird deutlich, weshalb die Caches während des Kopiervorganges deaktiviert sein müssen. Aus Sicht des Power PC handelt es sich bei den Maschineninstruktionen, welche von einer Speicheradresse (DPRAM) zu einer anderen (DRAM) kopiert werden, um *Daten*. Diese werden in den Data Cache geschrieben. Sollen sie dann ausgeführt werden, so findet sie die CPU folglich weder im Instruction-Cache noch im Speicher, folglich hat die CPU keinen Zugriff auf die Instruktionen. Alternativ zum Deaktivieren der Caches wäre es auch möglich, einen Cache-Flush zu erzwingen, den Inhalt der Datencaches also vor dem Start der Applikation in das RAM scheiben zu lassen.

1. Umprogrammierung des Logikbausteins MACH 210A

5.2.2 Meßumgebung

Im weiteren möchte ich die verwendeten Benchmarkprogramme kurz vorstellen, mit deren Hilfe die Interruptreaktions- und Taskumschaltzeiten gemessen wurden. Diese Programme waren bereits vorhanden und wurden für Laufzeitmessungen unter dem Echtzeitbetriebssystem VxWorks benutzt. Ich habe diese Tools für das Betriebssystem RTEMS portiert, damit die Messungen für beide Betriebssysteme unter vergleichbaren Bedingungen durchgeführt werden konnten. Weiterhin werde ich darlegen, auch welche Weise die Meßwerte gewonnen wurden.

5.2.2.1 Benchmark für Interrupt Reaktionszeit

Grundlage für dieses Benchmarkprogramm bildet die Software für den ASI-Demonstrator (Hochregallager) am FZI Karlsruhe. Wesentlich für dieses Programm ist eine spezielle Hardware, die mittels eines FPGA realisiert wird. Empfängt diese Hardware z.B. von einem ASI Sensor ein Datenpaket, so generiert das FPGA ein Signal, welches am Power PC einen Interrupt auslöst. Eine ausführliche Bescheibung des ASI-Demonstrators ist in [St99] nachzulesen.

Für diesen Interrupt wird eine Interruptserviceroutine bereitgestellt und beim Betriebssystem RTEMS registriert. Diese dient im ASI Demonstrator dazu, den Datenaustausch mit den Sensoren (ASI-Slaves) zu ermöglichen und ein Prozeßdatenabbild zu generieren. Für die Laufzeitmessungen wurde dieser Code entsprechend instrumentiert. Die erste Anweisung innerhalb der ISR schaltet eine Leuchtdiode ein, nach der Abarbeitung der Routine wird diese LED wieder deaktiviert.

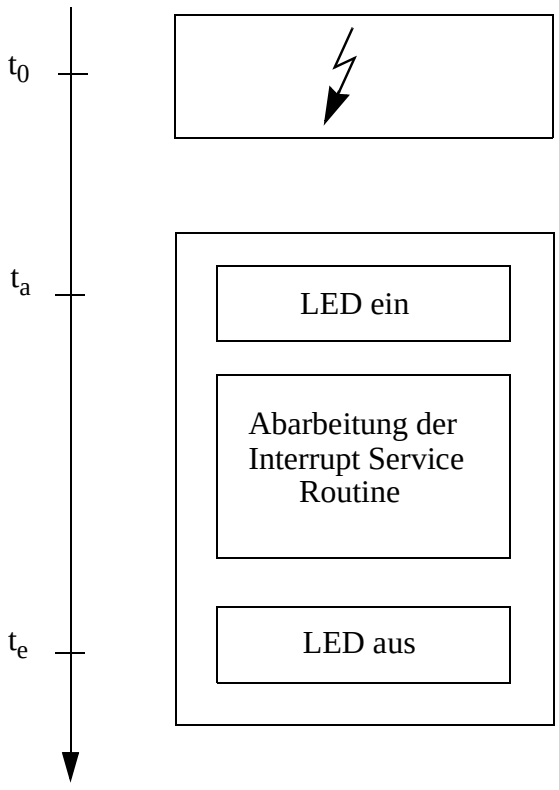


Abbildung 5.14 Ermitteln der Interrupt Reaktionszeit

Mit Hilfe eines Logic Analyzers zeichnet man die Signalverläufe am Pin des FPGAs, welcher den Interrupt generiert, und an der Leuchtdiode auf. Abbildung 5.14 zeigt, welche Informationen aus diesen Signalverläufen zu gewinnen sind.

Die **Interruptreaktionszeit** ist die Zeit, welche nach dem Auftreten des Interrupts vergeht, bis die Leuchtdiode aktiv wird. Um diese zu erhalten, ermittelt man die Differenz $\mathbf{t_a - t_0}$. Den Zeitverbrauch durch die ISR selbst kann man gewinnen, indem man die Differenz $\mathbf{t_e - t_a}$ bestimmt. Dies ist die Zeit, die zwischen dem Einschalten und der Deaktivierung der LED vergeht.

Die Messungen wurden sowohl mit Power PC 403 GA als auch mit Prozessoren der Reihe 403 GCX durchgeführt. Für jede CPU wurden die Meßwerte für vier verschiedene Cachekonfigurationen aufgenommen:

- Daten- und Befehlscache aktiv
- Nur Datencache aktiv
- Nur Befehlscache aktiv
- Beide Caches nicht aktiv

Die Konfiguration der Caches erfolgt innerhalb des Betriebssystems RTEMS, wenn die Hardware initialisiert wird.

```
#ifdef NO_INSTRUCTION_CACHE
    lis r1, 0x0000
#else
    lis r1, 0x8000
#endif

ori r1, r1, 0
mticcr r1



#ifdef NO_DATA_CACHE
    lis r1, 0x0000
#else
    lis r1, 0x8000
#endif

ori r1, r1, 0
mtdccr r1
```

Abbildung 5.15 Cachekonfiguration im Betriebsystem RTEMS[1]

Die Definitionen für die Cacheaktivierung werden dabei im Makefile des RTEMS gesetzt. Da die Betriebssystemfunktionen fest mit der Applikation gelinkt werden, ist nach einer Änderung dieser Cacheeinstellungen neben der Applikation auch das gesamte RTEMS neu zu kompilieren.

1. Auszug aus der Betriebssystemdatei flashentry.c

5.2.2.2 Benchmark für Task Umschaltzeit

In diesem Testprogramm werden 2 Tasks initialisiert, welche sich gegenseitig aktivieren. Dazu existieren zwei Semaphore. Jede der Tasks versucht, ein Semaphore zu allokieren und verbraucht anschließend etwas Rechenzeit (Dummy Code). Danach gibt die Task die jeweils andere Semaphore wieder frei.

Der Programmcode ist instrumentiert, damit er für eine Meßwerterfassung durch einen Logic Analyzer geeignet ist. Als Schnittstelle dienen wiederrum die LEDs, welche über die I/O Bank ansprechbar sind.

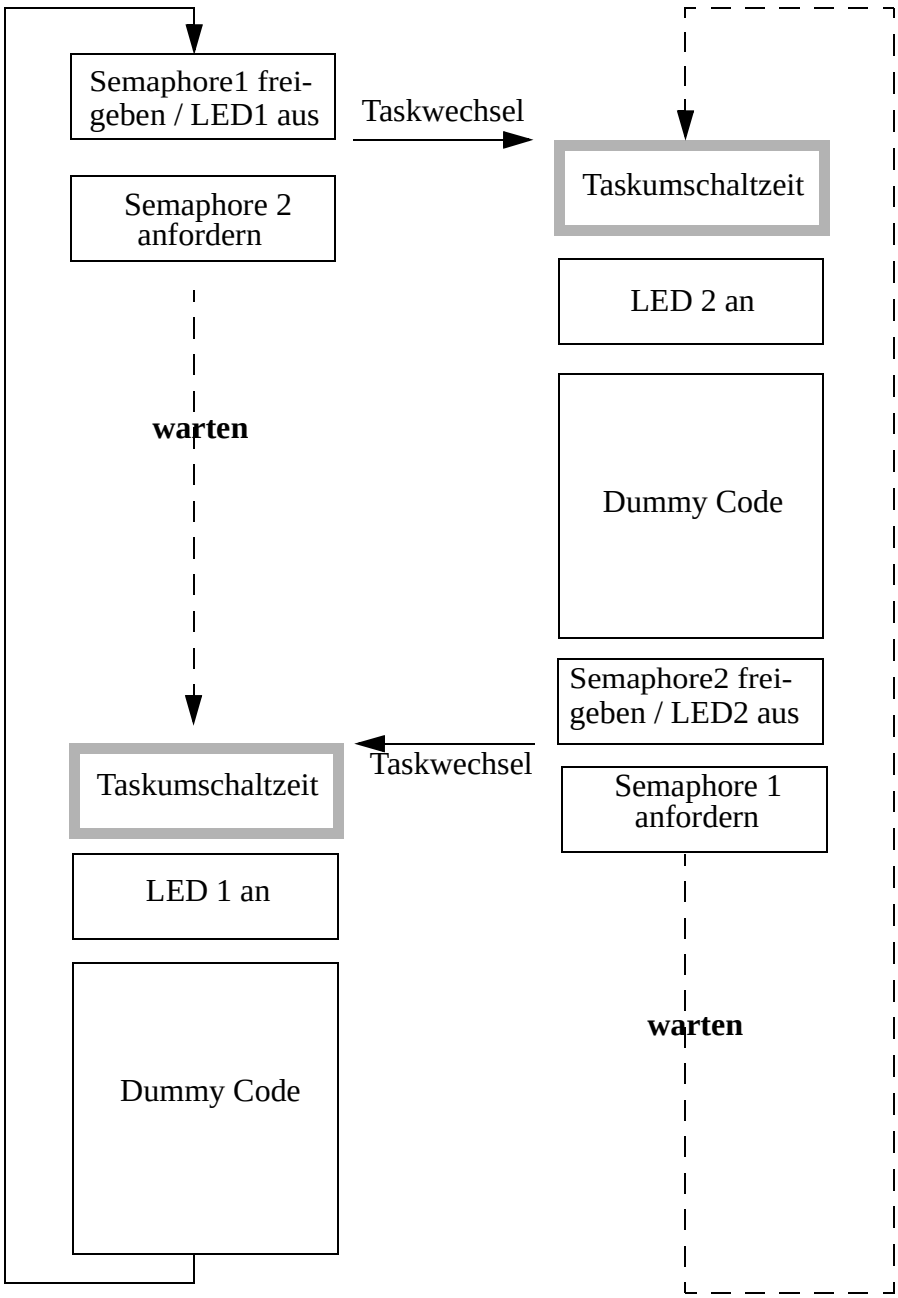


Abbildung 5.16 Messen der Taskumschaltzeiten

Mit Hilfe eines Logic Analyzers werden die Signalverläufe der beiden LED aufgezeichnet. Die abwechselnde Aktivität der beiden Task wird dabei im wechselseitigen Aktivieren und Deaktivieren der Leuchtdioden deutlich.

Wie man in Abbildung 5.16 erkennt, sind während der Taskumschaltzeit beide LED kurz deaktiviert, so daß die Task-Switching-Time meßtechnisch erfaßbar wird. Auf dem Logic Analyzer ist die Taskwechselzeit als Phasenverschiebung der beiden LED-Signale erkennbar, wie Abbildung 5.17 zeigt.

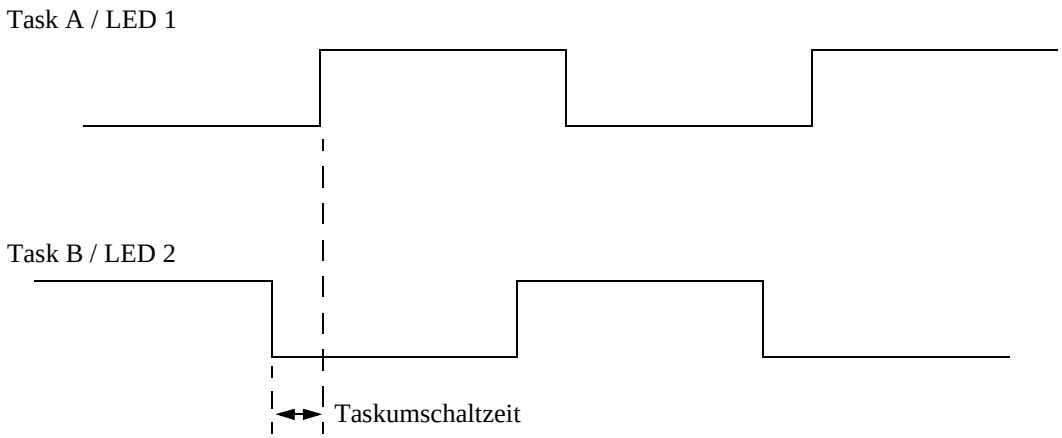


Abbildung 5.17 Messen der Taskumschaltzeiten 2

Die Taskumschaltzeiten wurden für Power PC 403GA und 403GCX für verschiedene Cachekonfigurationen:

- Daten- und Befehlscache aktiv
- Nur Datencache aktiv
- Nur Befehlscache aktiv
- Beide Caches nicht aktiv

Die dazu notwendigen Anpassungen im Betriebsystem RTEMS sind in Abschnitt 5.2.2.1 beschrieben.

5.2.3 Meßwerte

Datencache	Befehlscache	Taskumschaltzeit in µs
		72.7
✓		68.7
	✓	43.3
✓	✓	33.4

Tabelle 5.7: Taskumschaltzeiten für PPC403 GA (40 Mhz)

Datencache	Befehlscache	Taskumschaltzeit in µs
		76.0
✓		65.6
	✓	28.7
✓	✓	14.3

Tabelle 5.8: Taskumschaltzeiten für PPC 403GCX (40 Mhz)

Datencache	Befehlscache	Interruptreaktions-zeit in µs
		14.7
✓		12.3
	✓	10.1
✓	✓	7.4

Tabelle 5.9: Interruptreaktionszeiten für PPC 403GA (40 Mhz)

Datencache	Befehlscache	Interruptreaktions-zeit in µs
		14.5
✓		11.5
	✓	5.5
✓	✓	2.0

Tabelle 5.10: Interruptreaktionszeiten für PPC 403GCX (40 Mhz)

5.2.4 Zusammenfassung

Der funktionale Zusammenhang zwischen Taskumschalt- und Interruptreaktionszeiten sowie der Cachekonfiguration unter RTEMS ist vergleichbar mit den Ergebnissen[1], welche für das Echtzeitbetriebsystem VxWorks gewonnen wurden. Obwohl RTEMS besonders bei den Interruptreaktionszeiten eine bessere Performance als VxWorks zeigt, wird deutlich, daß die Echtzeitfähigkeiten eines eingebetteten System betriebsystemübergreifend von der eingesetzten Hardwarearchitektur bestimmt werden. Die für Taskwechsel und Interruptaufrufe erforderlichen Aktionen, wie z.B. das Sichern des Prozeßabbildes, werden von der Hardware vorgegeben. Die dafür benötige Zeit kann vom Betriebsystem durch geschickte Programmierung minimiert werden, stellt aber stets eine vorgegebene Schranke dar.

1. Eine ausführliche Darstellung der Untersuchungen für VxWorks findet sich in [WSNRo99]

5.3 Compile Time Rekonfiguration

SRAM basierte FPGA verlieren nach dem Abschalten der Betriebsspannung ihre Konfiguration. Sie müssen demnach nach jeder Inbetriebnahme der zugrundeliegenden Hardwareumgebung neu initialisiert werden. Dies kann zum Beispiel geschehen, indem ein elektrisch löschbares EPROM[1] die Konfigurationsdaten bereithält. Der FPGA auf dem Spyder-Board enthält intern die notwendige Logik, um diesen Datenstrom zu lesen und sich somit selbst zu initialisieren. Dieser Vorgang geschieht bei einem Power-On-Reset automatisch und bedarf keiner Unterstützung durch den Mikroprozessor. Die Initialisierung eines FPGA durch ein solches EEPROM ist ein typisches Beispiel für die in Abschnitt 3.4.1 beschriebene Compile Time Rekonfiguration. Ein Design wird einmal in das FPGA geladen und danach nicht mehr modifiziert.

Das Konfigurationsfile für das FPGA mußte ursprünglich mittels eines externen Programmiergerätes in das EEPROM geschrieben werden. Abgesehen vom Zeitaufwand unterlagen die EEPROMs und besonders die Stecksockel durch den häufigen Wechsel der Chips einem nicht vernachlässigbaren Verschleiß.

Ich habe daher das Program *ELOAD* entwickelt, welches die Programmierung des Speicherchips erlaubt, während dieser auf dem Spyder-Core System verbleibt. Unterstützt werden die seriellen EEPROMs der Firma Atmel, welche für die Konfiguration von FPGAs konzipiert wurden. Eload ermöglicht die Programmierung mit EEPROMs mit 64, 128, 256, 512 und 1024 kBit Speicherkapazität.

5.3.1 Funktionsprinzip

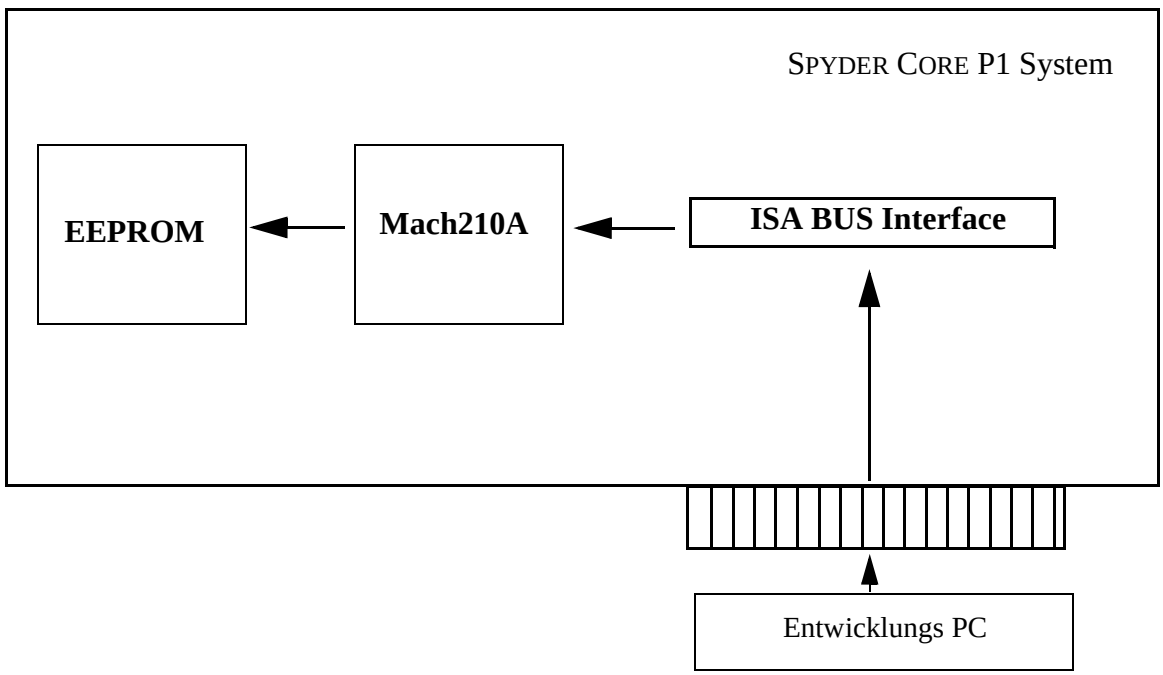


Abbildung 5.18 Datenfluß beim Laden des seriellen EEPROM

1. Zur Funktionsweise dieser EEPROM siehe Abschnitt 3.3.1

Abbildung 5.18 zeigt den Datenfluß beim Laden des seriellen EEPROMs. Vom Entwicklungs-PC aus werden Schreibzugriffe auf bestimmte Adressen (Register[1]) durchgeführt. Diese Schreibzugriffe führen zu Busaktivitäten auf dem ISA-Bus[2]. An diesem BUS ist auf dem Spyder-Core System ein programmierbarer Logikbaustein (Mach 210A von AMD) angeschlossen. Dieser wertet die Busaktivitäten aus und erzeugt die entsprechenden Signale, mit deren Hilfe das EEPROM programmiert werden kann. Das Prinzip von ELOAD beruht also darauf, das Design File als Bitstrom an das EEPROM zu senden, indem die passenden Datenbits in den Registern mit binären Nullen bzw. Einsen beschrieben werden.

5.3.2 Programmieralgorithmus für ATMEL EEPROMs

Im folgenden möchte ich die Programmierung serieller EEPROMS der Typen AT17Cx[3] beschreiben. Obwohl die Chips für verschiedene Speicherkapazitäten zwischen 64 kBit und einem MBit angeboten werden, ist der Programmieralgorithmus für alle Schaltkreise bis 256 kBit Speichergröße gleich. EEPROMs der Kapazität von 512 kbit und 1 MBit werden nach einem anderen Algorithmus programmiert.

Die Programmierung erfolgt durch einen seriellen Bitstrom. Für dessen Übertragung ist für den Programmierer nur das Data- und das Clocksignal interessant. Die Übertragung erfolgt in Blöcken. Jeder dieser Blöcke wird durch ein *Startbit* eingeleitet, es folgt der Payload in Form von *Null*- und *Einsbits*, den Abschluß bildet ein *Stopbit*.

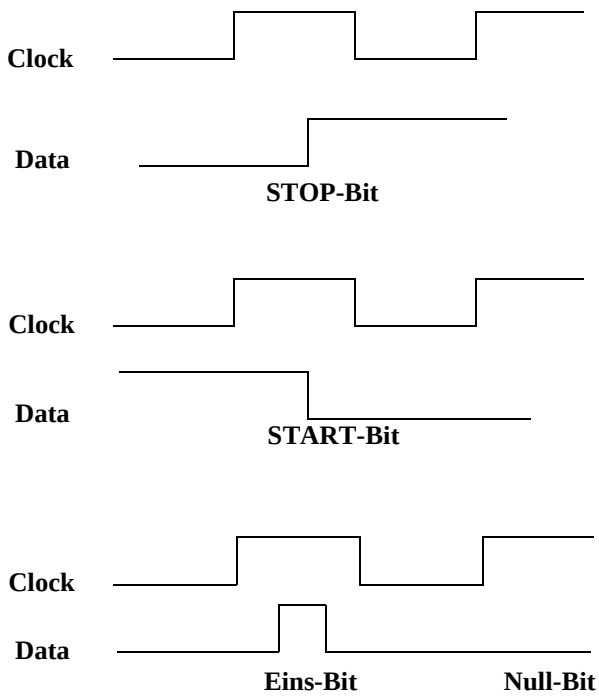


Abbildung 5.19 Bitformate

1. Als Register werden an dieser Stelle ausgewählt Adressen im Adreßraum des PC bezeichnet. Es handelt sich dabei also *nicht* um Register eines Mikroprozessors oder um Adressen im I/O Raum.
2. PC Architektur für Erweiterungssteckkarten
3. AT17C65, AT17C128, AT17C256, AT17C512 und AT17C010

Abbildung 5.19 zeigt die benutzten Bitformate. Ein Startbit ist dadurch gekennzeichnet, daß eine High-Low Flanke auftritt, während das Clocksignal Highpegel hat. Fällt eine Low-High-Flanke des Datensignals in einen Zeitraum, in welchem das Clocksignal High ist, handelt es sich um ein Stopbit. Datenbits, die eine binäre Eins darstellen, sind während aktiven Clock permanent high, Nullbits haben entsprechend einen Lowpegel.

Message Format

START BIT	DEVICE ADDRESS	1ST ADDRESS BYTE	2ND ADDRESS BYTE	3RD ADDRESS BYTE (010/512 Only)	DATA BYTE(S)	STOP BIT

Abbildung 5.20 Messageformat für die Programmierung eines AT17Cx EEPROM (Quelle: [Atmel1])

Das Beschreiben des EEPROM erfolgt blockweise. Um einen solchen Block zu programmieren, muß ein Paket an den Baustein gesendet werden, welches das in Abbildung 5.20 dargestellte Format besitzt. Device-Address und Address-Bytes sind dabei so codiert, daß das höchstwertige Bit *zuerst* übertragen wird, während bei den Datenbytes das niederwertigste Bit zuerst gesendet wird. Nach jedem empfangenen Byte generiert das EEPROM ein sogenanntes Acknowledge Bit.

Device Address Byte: Mit Hilfe des R/$\overline{W}$ Bits wird festgelegt, ob die aktuelle Transaktion eine Schreib- oder Leseoperation darstellt. Im folgenden werden nur Schreibvorgänge betrachtet. Das A_2-Bit dient dazu, daß sich 2 EEPROMS des gleichen Typs den Daten- und Adreßbus teilen können. Die Chips sind parallel an den entsprechenden Leitungen angeschlossen. Dabei wird einem Baustein die Adresse „1“, dem anderen die Adresse „0“ zugeordnet. Dies geschieht, indem ein spezieller Pins des Chips mit Vcc oder Ground verbunden wird. Über das A_2-Bit selektiert man dann beim Beschreiben entweder Chip 0 oder Chip 1.

Device Address Byte

MSB							LSB
1	0	1	0	A_2	1	1	R/$\overline{W}$
1st	2nd	3rd	4th	5th	6th	7th	8th

Where: R/$\overline{W}$ = 1 Read
= 0 Write
A2 = 1 if A2 pin is at V_{cc}
= 0 if A2 pin is at GROUND

Abbildung 5.21 Device Address Byte (Quelle: [Atmel1])

Address Bytes: Das Beschreiben des EEPROM erfolgt in sogenannten Pages. Eine solche Seite ist bei den 17C65/128/256 Chips jeweils *64 Bytes* groß, bei den 17C512 und 17V010 Bausteinen dagegen *128 Byte.* Das Beschreiben kann dabei an jeder beliebigen Adresse erfolgen, allerdings muß immer eine vollständige Seite beschrieben werden. Die Startadresse ist entweder 16 bit (also 2 Byte) lang oder 24 bit für größere EEPROMs. Die Byteorder ist Big Endian.

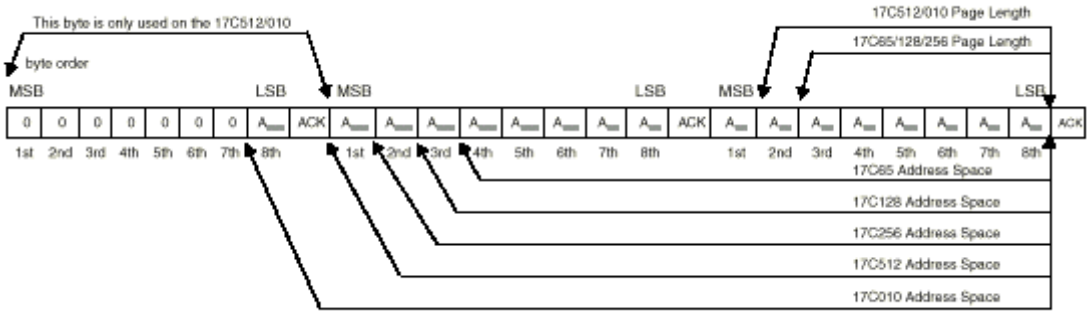


Abbildung 5.22 Address Bytes (Quelle: [Atmel1])

Data Bytes: Nach den Adreßbytes werden die 64 bzw. 128 Byte einer Page einfach hintereinander übertragen.

5.3.3 Programmierung von Atmel EEPROMS auf Spyder-Core

5.3.3.1 Dateiformate

Das Programm Eload schreibt jeweils ein FPGA Design-File in ein entsprechendes EEPROM. Während im EEPROM ein vom FPGA lesbares Binärformat abgelegt wird, generieren Softwarepakete wie z.B. die Xilinx Foundation Tools andere, plattformunabhängige Formate. Verbreitet ist z.B. das Xilinx ASCII Bitstream Format (RBT-Format). Dabei handelt es sich um eine extrem simple ASCII-Codierung einer Binärdatei.

```
Xilinx ASCII Bitstream
Created by Bitstream M1.4.12
Design name: asic-x1.ncd
Architecture: xc4000e
Part: 4010epq208
Date:  Tue Jul 28 16:25:22 1998
Bits:  178144
11111111001000000010101101111110110011111
01010111111111101111111101011111111010111111110101111111101011111110
11111110101111111010111111101011111110101111111010111111101011111111
11010111111101011111111010111111101011111110101111111010111111110101
11111101011111111011111111111
```

Abbildung 5.23 Xilinx Bitstream Format

Im RBT Format werden die binären Einsen und Nullen eines Byte einfach als ASCII Ziffern dargestellt, wobei das niederwertigste Bit zuerst notiert wird.

Ein weiteres ASCII Format stammt von Intel und stellt eine Binärdatei in der Form von Adressen und Hexadezimalwerten im Klartext dar.

```
020000020000FC
:10000000FF04D4BEF9EAFF7CFDCDB7F77BFDADF769
:100010005A7FEB9DADBFFBD6EBEFB6FFD7EEAFBF80
:10002000BFF9FFF6FB6FFFFFDEFBEFBDEFFEFBDF6F
:1000300056FFADF7DF7FDBFFF5D77FFFFFEEFFFF5A
:10004000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC0
:10005000FFFFFFFFFFFFFFFFF9BFFBFFFFEFBEF69
```

Abbildung 5.24 Intel MCS Format

Eload unterstützt beide Formate und führt vor dem Programmierung des EEPROMs eine Konvertierung der Dateiinhalte in ihre Binäräquivalente durch.

5.3.3.2 Programmierung

In den vorangegangenen Abschnitten wurde die Programmierung des EEPROM funktional beschrieben. Ist ein solcher Baustein in einem eingebetteten System integriert, sind darüber hinaus einige weitere Punkte zu beachten. So muß z.B. der Chip aktiviert und andere Komponenten abgeschaltet werden, um einen störungsfreien Betrieb zu ermöglichen. Im wesentlichen erfolgt die Programmierung in 3 Phasen:

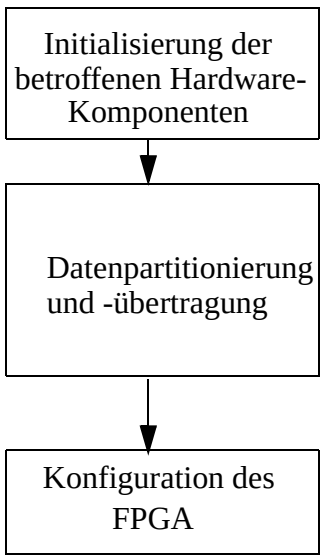


Abbildung 5.25 Programmierung eines EEPROM auf SPYDER-CORE-P1

Vor Beginn der Programmierung sind folgende Signale zu setzen:

- Signal *PROG* deaktivieren, um den FPGA vom Bus zu trennen. Diese Maßnahme ist notwendig, um Störungen während des Programmiervorgangs zu vermeiden
- Signal *SER_ENA* aktivieren, um das Beschreiben des EEPROM zu ermöglichen.
- Signal SLAVEOUT aktivieren, um das FPGA vom EEPROM zu trennen

Die Generierung der Signale erfolgt durch Schreibzugriffe des PC auf spezielle Adressen, wie in Abschnitt 5.3.1 beschrieben.

Nach diesen Initialisierungsschritten kann die eigentliche Datenübertragung zum EEPROM beginnen. Das Design File wird nach seiner Konvertierung in ein lesbares Binärformat in Blöcken der Größe von 64 bzw. 128 Byte in den Speicherchip geladen. Das Protokoll folgt dabei den Spezifikationen, wie sie in Abschnitt 5.3.2 beschrieben wurden:

- Senden eines STARTBIT
- Senden eines Devicebyte
- Senden von 2 bzw. 3 Adreßbytes
- Senden von 64 bzw. 128 Datenbytes
- Senden eines STOPBIT
- Pause einfügen, in welcher der EEPROM diese Page durch einen internen Vorgang in seinen Speicher schreibt.

Wichtig: Bei der Serialisierung der Daten muß man übrigens beachten, daß das EEPROM nach jedem empfangenen Byte ein Acknowledge-Bit generiert. Daher muß nach der Übertragung von 8 Bits eine zusätzliche Wartepause eingefügt werden, in welcher das ACK-Bit generiert werden kann. In dieser Zeit darf die Datenleitung vom Ladeprogramm nicht getrieben werden, da es ansonsten zu Fehlern im Ladeprotokoll kommt.

Ist das EEPROM programmiert, wird ein Reset am FPGA ausgelöst:

- Signal *SLAVEOUT* deaktivieren, um das FPGA an das EEPROM zu koppeln
- Signal *PROG* aktivieren, damit das FPGA seine Konfigurationsdaten laden kann

Das FPGA liest darauf hin den Inhalt des Speicherchips aus und konfiguriert sich.

5.4 Run Time Rekonfiguration

Um die Möglichkeiten der Run-Time-Rekonfiguration auf dem Spyder-Core System nutzten zu können, mußten zwei neue Features realisiert werden.

Zum einen benötigt die Applikation eine Bibliothek von Design Files, mit denen das FPGA rekonfiguriert werden kann. Diese Dateien müssen auf einem geeigneten nichtflüchtigen Datenträger abgelegt werden und vom Betriebssystem verwaltbar sein. Um dieses Problem zu lösen, habe ich mich entschieden, ein Filesystem bereitzustellen. Mit Hilfe dieses Filesystems kann sowohl auf Teile des Flash-RAM von SPYDER-CORE-P1 als auch des DRAMs zugegriffen werden.

Weiterhin muß eine Applikation die Möglichkeit haben, das FPGA zu einem beliebigen Zeitpunkt umkonfigurieren zu können. Zu diesem Zweck habe ich das Echtzeitbetriebsystem VxWorks um einen Gerätetreiber erweitert, welcher als Schnittstelle zwischen Hardware und Anwendungsprogramm dient. Wie auch das Filesystem verbirgt er die internen Hardwarestrukturen vor der Anwendung.

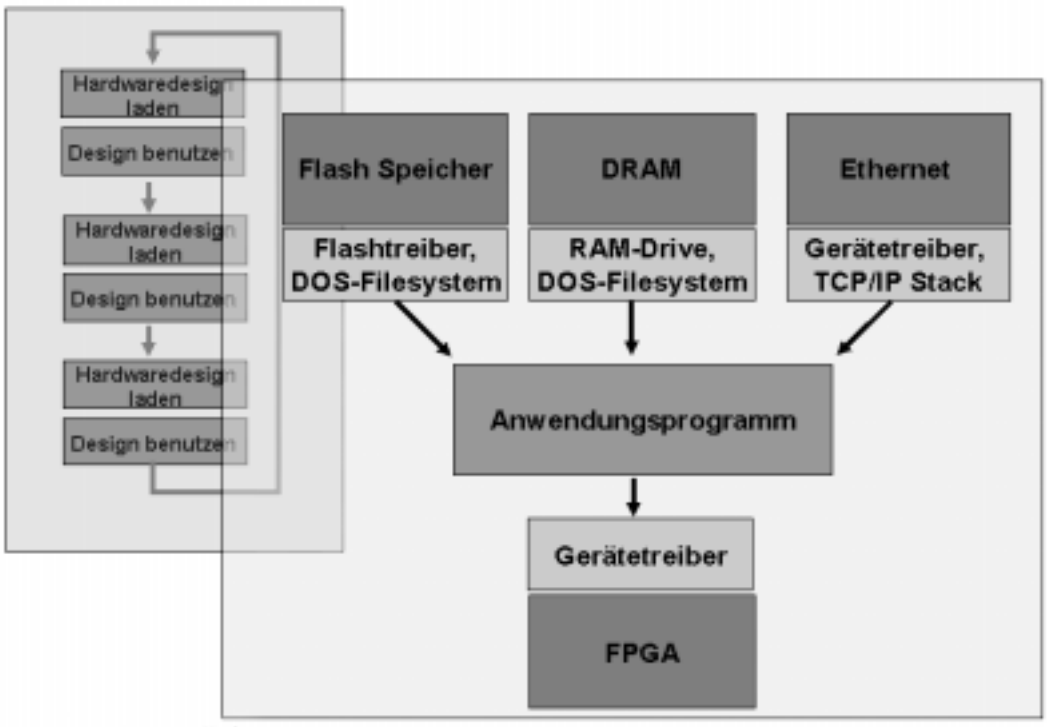


Abbildung 5.26 Run Time Rekonfiguration auf SPYDER-CORE-P1

Abbildung 5.26 zeigt das Zusammenspiel der betroffenen Komponenten. Neben den bereits erwähnten Flash- und DRAM-Speichern ist im Bild eine weitere Quelle zu erkennen, von der FPGA-Design-Files geladen werden können. Für die TCP/IP Kommunikation waren aber keine Änderungen auf Betriebssystemebene zu realisieren. Daher möchte ich die Besprechung der Möglichkeiten, welche sich durch eine Vernetzung ergeben, auf den Abschnitt 5.5 verschieben.

5.4.1 DOS Filesystem für Flash-RAM

Ein Filesystem ist eine Schnittstelle zwischen der Applikation und den Gerätetreibern, welche die Kommunikation mit der eigentlichen Hardware übernehmen. Die Anwendung muß daher über keine Kenntnisse verfügen, auf welchem Medium die Daten abgelegt sind.

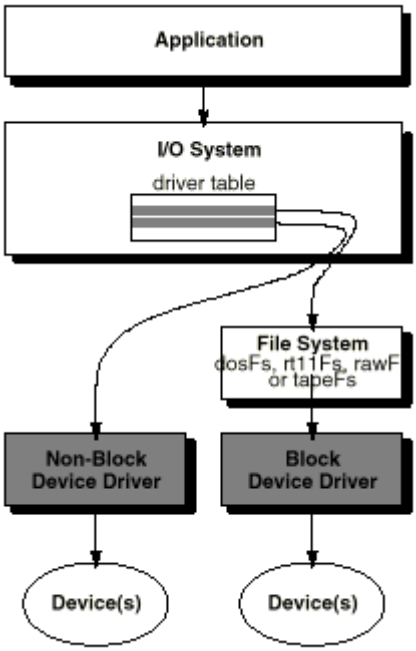


Abbildung 5.27 Non-Block Devices vs. Block Devices (Quelle:[Wind97])

Abbildung 5.27 zeigt, wie sich Filesysteme in das Schichtenmodell des Betriebssystems VxWorks einfügen. Bevor ein Filesystem benutzt werden kann, muß also zuerst der entsprechende Block Device Treiber initialisiert werden. Ist dies geschehen, kann auf dem Gerät ein Filesystem erzeugt werden. Anschließend wird das Filesystem in die Verzeichnisstruktur eingebunden. Diesen Vorgang nennt man auch „mounten“. Im folgenden möchte ich beschreiben, wie man unter VxWorks einen Treiber für ein Block Device bereitstellt und wie man danach ein Filesystem initialisiert bzw. mounted.

Block Device Treiber initialisieren: Man beschreibt eine Reihe von Parametern in Form einer Datenstruktur[1]. Diese Tabelle spezifiziert Pointer auf die Funktionen, die der Devicetreiber bereitstellt und beinhaltet darüberhinaus einige weitere Parameter. Tabelle 5.11 beschreibt diese Block Device Datenstruktur.

1. Pointer auf einen Speicherbereich

Parameter	Beschreibung
bd_blkRd	Pointer auf Treiberroutine, welche einen Block liest
bd_blkWrt	Pointer auf Treiberroutine, welche einen Block schreibt
bd_ioctl	Pointer auf Treiberroutine, welche die I/O Steuerung übernimmt
bd_reset	Pointer auf Treiberroutine, die einen Reset des Gerätes bewirkt
bd_statusChk	Pointer auf Treiberroutine, welche eine Statuscheck für das Medium ausführt
bd_removeable	*TRUE*[a], wenn das Medium wechselbar ist, z.B. eine Floppy-Disk, sonst *FALSE*
bd_nBlocks	Anzahl der Blöcke auf dem Gerät
bd_bytesPerBlk	Blockgröße in Byte
bd_blksPerTrack	Anzahl der Blöcke pro Spur
bd_nHeads	Anzahl der Schreib/Leseköpfe
bd_retry	Anzahl der Wiederholungen einer Aktion nach einem Fehler
bd_mode	Write Only, Read Only oder Read/Write
bd_readyChanged	TRUE, wenn sich der Device-Ready-Status des Gerätes geändert hat - sollte immer mit TRUE initialisiert werden, damit das Medium gemountet wird

Tabelle 5.11: Block Device Parameter unter VxWorks

a. TRUE ist eine Konstante unter VxWorks

Für die Realisierung eines Filesystems für das Flash-RAM auf Spyder-Core-P1 mußten dem Betriebsystem nur zwei Treiberroutinen bereitgestellt werden: eine Routine, welche einen Block in das Flash-RAM schreiben kann und eine weitere für die entsprechende Lesefunktion. Diese bereits vorhandenen Low-Level-Funktionen übernehmen dann die eigentliche Kommunikation mit den Flashspeicher Chips.

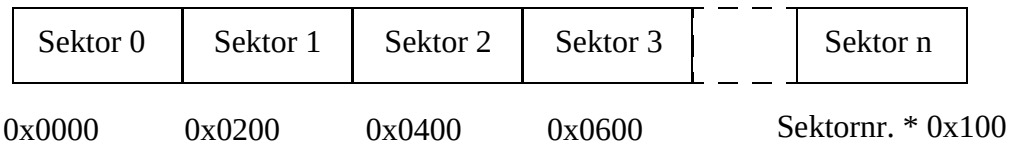


Abbildung 5.28 Flash-RAM als Laufwerk

Die Geometrieparameter wurden so gewählt, daß sich das Flash-RAM gegenüber dem System wie ein Laufwerk mit einem Kopf, einer Spur und entsprechend seiner Speicherkapazität

vielen Sektoren zu je 512 Byte verhält. Abbildung 5.28 zeigt, wie die „Sektoren“ im Adreßbereich des Flash-Speichers abgelegt werden.

Versucht z.B. das Filesystem, den Sektor 2 zu lesen, so wird die eine Treiberroutine aufgerufen, welche ab Flashadresse[1] 0x200 einen Block von 512 Byte aus dem Flash ausliest.

Im folgenden Beispiel wird eine solche Parametertabelle erzeugt:

```
BLK_DEV *flash0_DeviceCreate(int blocks, int mode)
{
BLK_DEV *dev_flash0;
dev_flash0 = (BLK_DEV*) malloc(sizeof(BLK_DEV));

dev_flash0->bd_blkRd =  (FUNCPTR) flashBlockRead;
dev_flash0->bd_blkWrt = (FUNCPTR) flashBlockWrite;
...
dev_flash0->bd_readyChanged = TRUE;

return dev_flash0;
}
```

Abbildung 5.29 Erstellen einer Parametertabelle für Block Device Treiber

Filesystem initialisieren und verfügbar machen: Auf dem durch die Parametertabelle beschriebenen Blockdevice kann nun mittels einer Betriebssystemfunktion ein DOS-Filesystem (oder ein anderes Filesystem) erzeugt werden. Dieser Vorgang wird auch als Formatieren bezeichnet.

Sowie ein gültiges Filesystem existiert, kann dieses in die Verzeichnisstruktur eingebunden (gemountet) werden. Die folgenden Bespiel zeigen, wie ein Filesystem angelegt und automatisch gemountet wird, bzw. wie man ein bereits existierendes Filesystem einbindet:

```
// Parameter für Blocktreiber initialisieren:
dev_flash0 = flash0_DeviceCreate(anzahl_blocks, ReadWrite);

// Formatieren:
volume = dosFsMkfs(“/“, dev_flash0);
```

Abbildung 5.30 Formatieren und automatisch mounten

1. Die angegebenen „Flashadressen“ sind Offsetadressen innerhalb des Adreßraums des Flash-Speichers und keine Adressen im Adreßraum des Mikrocontrollers

```
// Parameter für Blocktreiber initialisieren
dev_flash0 = flash0_DeviceCreate(anzahl_blocks, ReadWrite);

// Filesystem an das Rootverzeichnis “/“ mounten
 volume = dosFsDevInit(“/“, dev_flash0, 0);
```

Abbildung 5.31 Mounten eines existierenden Filesystems

5.4.2 DOS Filesystem für DRAM (RAM-Drive)

Für das DRAM sind die Block Device Driver bereits in VxWorks integriert. Die Initialisierung kann daher einfacher ausfallen als beim Flash-RAM. Zuerst müssen wiederum die Parameter für den Device Treiber initialisiert werden, danach kann das Filesystem erzeugt bzw. gemountet werden.

Block Device Treiber initialisieren: Die Initialisierung erfolgt analog, wie dies im vorangegangenen Abschnitt über den Flash-Speicher beschrieben wurde. Durch den Aufruf der Betriebsystemfunktion *ramDevCreate* wird die Parametertabelle für den Treiber erstellt. Die Übergabe der Parameter erfolgt in den Argumenten der Funktion:

ramDevCreate(rdev_addr, rdev_blksize, rdev_blksPerTrck, rdev_Blocks, rdev_blkSkip)

Argument	Beschreibung
rdev_addr	Beginn des RAM-Filesystems im Adreßraum des Mikroprozessors
rdev_blksize	Blockgröße in Byte
rdev_blksPerTrck	Anzahl der Blöcke pro Spur
rdev_Blocks	Anzahl der Blöcke auf dem Gerät
rdev_blksSkip	Anzahl der Blöcke, die am Beginn übersprungen werden

Tabelle 5.12: Parameter für RAM-Drive

Filesystem initialisieren und verfügbar machen: Das Formatieren und Mounten erfolgt analog, wie in Abschnitt 5.4.1 beschrieben.

Im folgenden Beispiel wird eine RAM-Disk der Größe von 200 kByte angelegt. Alle Sektoren befinden sich in einer Spur, die Sektorgröße ist 512 Byte. Das Betriebssystem sucht automatisch eine Adresse, ab der genügend freier Speicher für die Erzeugung der RAM-Disk zur Verfügung steht (rdev_addr = 0). Anschließend wird ein Filesystem erzeugt, welches an das Verzeichnis „/ramdrive“ gemountet wird.

```
RamDrive = ramDevCreate (0, 512, 400, 400, 0);
volume = dosFsMkfs(“/ramdrive“, RamDrive);
```

Abbildung 5.32 Erzeugen und Mounten einer RAM-Disk

5.4.3 Gerätetreiber für XC4000 FPGA

5.4.3.1 Treiberintegration unter VxWorks

Im Gegensatz zu einem Speichermedium mit Filesystem erfolgt die Kommunikation mit einem FPGA nicht blockorientiert, sondern zeichenweise. Um das FPGA innerhalb der Treiberarchitektur von VxWorks verfügbar zu machen, mußte ein Character- (bzw. Non-Block) Device Treiber implementiert werden.

Ein Treiber für ein solches Non-Block Device stellt insgesamt sieben Funktionen bereit: *creat(), remove(), open(), close(), read(), write()* und *ioctl()*. Die Implementierung dieser Routinen ist hardwareabhängig. Unter Umständen werden aber nicht alle Funktionen benötigt.

Ein Treiber wird installiert, indem die Betriebssystemfunktion *iosDrvInstall* aufgerufen wird. Dadurch wird der Treiber registriert, d.h. die Adressen der oben genannten I/O Funktionen in entsprechende Systemtabellen eingetragen. Das I/O System von VxWorks liefert für den soeben registrierten Treiber eine sogenannte Device Driver Number zurück.

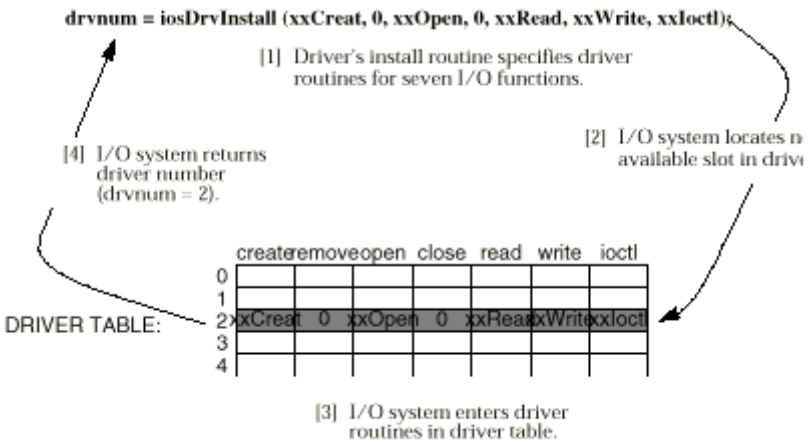


Abbildung 5.33 Treiberinitialisierung für Non-Block Devices (Quelle:[Wind97])

Es kann vorkommen, daß in einem System mehrere Geräte des gleichen Typs vorhanden sind, z.B. mehrere FPGAs. Die Registrierung des Treibers für diese Geräte erfolgt nur einmal. Um dennoch auf jedes dieser I/O Devices gezielt zugreifen zu können, wird ihnen ein Name gegeben und die Nummer des entsprechenden Gerätetreibers zugeordnet. Dies geschieht durch Aufruf der VxWorks-Funktion *iosDevAdd*.

Ab diesem Zeitfunkt werden alle I/O Funktionen, die sich auf den Namen eines Gerätes beziehen, vom System an die entsprechende Treiberroutine weitergeleitet. In Abbildung 5.34 werden z.B. zwei Geräte unter den Namen „/xx0“ und „/xx1“ registriert. Der zuständige Gerätetreiber hat die Treibernummer zwei und könnte z.B. ein Treiber für serielle Schnittstellen sein. Wird nun z.B. eine Funktion wie **fopen(“/xx0“ , “r+“)** aufgerufen, so wird dieser Aufruf durch die Routine *open()* des Schnittstellentreibers abgearbeitet. Die Argumente dev0 und dev1 dienen im übrigen dazu, weitere geräteabhängige Parameter zu spezifizieren. Im Beispiel könnten den Treiberroutinen u.a. die verschiedenen I/O Adressen der beiden seriellen Schnittstellen mitgeteilt werden.

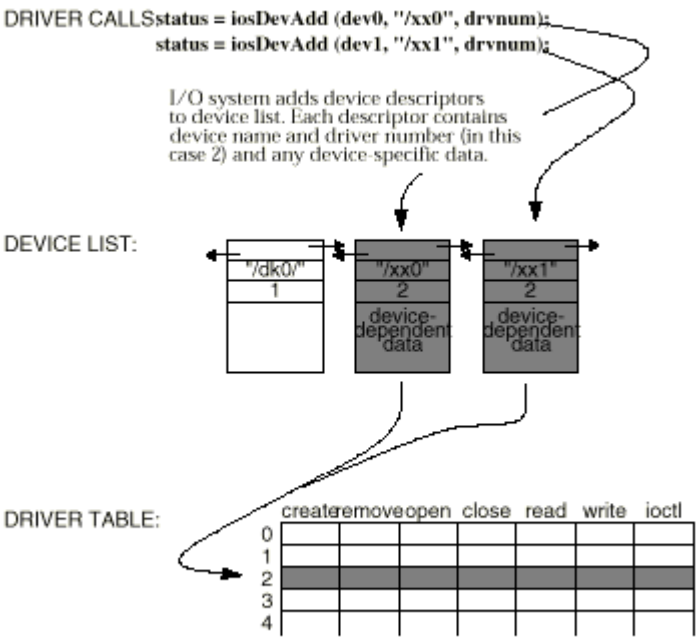


Abbildung 5.34 Hinzufügen von Geräten (Quelle:[Wind97])

5.4.3.2 Umsetzung des Konzepts für SPYDER-CORE-P1

Im folgenden möchte ich darstellen, wie ich den Devicetreiber für FPGAs der Serie XC4000 in das Betriebssystem VxWorks eingebunden habe. Um das FPGA durch eine Applikation konfigurieren zu lassen, sind nur die Funktionen zum Öffnen, Beschreiben und Schließen des Gerätes notwendig, so daß ich mich auf die Beschreibung dieser Routinen beschränken möchte.

Treiberfunktion *Open():*

Die Funktion *Open()* stellt die exklusive Verfügbarkeit des FPGA für die Task sicher, durch welche sie aufgerufen wurde. Dazu allokiert sie ein binäres Semaphore. Dieses Semaphore wird erst durch die Operation *Close()* wieder freigegeben. Versuch eine andere Task in der Zwischenzeit einen Zugriff auf das FPGA, wird dies abgeweisen und eine entsprechende Fehlermeldung generiert. Nachdem das Semaphore allokiert ist, wird das Spyder-System initialisiert, so daß daraufhin das FPGA konfiguriert werden kann. In Abbildung 5.35 ist dargestellt, wie diese Sequenz verläuft. Das Setzen der dort erwähnten Signale erfolgt wieder durch Schreibzugriffe auf spezielle Adressen im Adreßraum des Mikroprozessors. Ein Mach210A[1] analysiert die Aktivitäten auf dem Adreß- und Datenbus und generiert dann die entsprechenden Logikzustände der betroffenen Steuerleitungen. Eine Übersicht über dieses Adressen folgt am Ende dieses Abschnitts.

1. ein programmierbarer Logikchip der Firma AMD

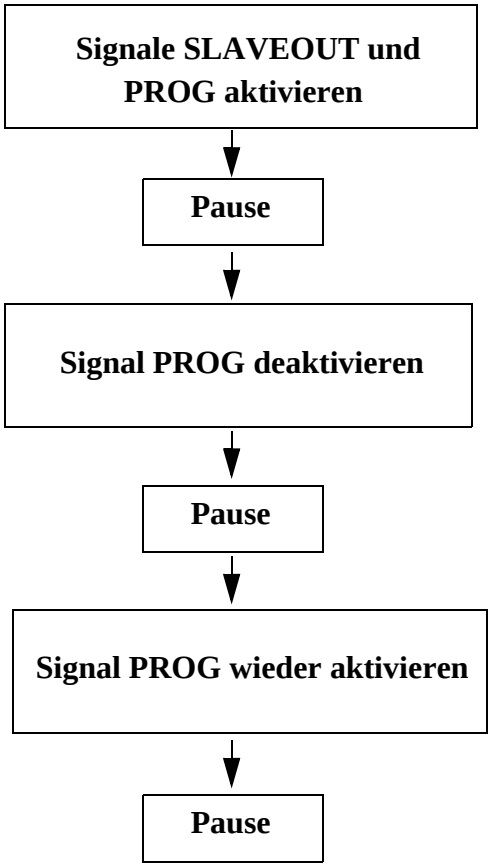


Abbildung 5.35 Hardware-Initialisierungssequenz der Funktion Open()

Treiberfunktion *Write()*:

Schreibt eine Applikation ein Zeichen auf das FPGA, so wird vom Betriebssystem die Treiberfunktion *Write()* aktiviert. Diese Funktion serialisiert das Zeichen und sendet es dann bitweise an das FPGA, wobei das niederwertigste Bit zuerst übertragen wird. Die Übertragung geschieht durch Schreibzugriffe auf das Datenregister des FPGA, welches dem Mikroprozessor über eine Adresse innerhalb seines Speicherraums zugänglich ist. Die Konfiguration ist beendet, wenn das *DONE*-Signal, welches vom FPGA kommt, aktiv wird.

```
Senden(DataByte):

for Bit=0 to 7 of DataByte
{
 if Bit = 0 then
  Write(FpgaDataPort, 0)
 else
  Write(FpgaDataPort, 1)
}
```

Abbildung 5.36 Datenübertragung

Treiberfunktion *Close:*

Diese Routine löscht das *SLAVEOUT* Signal wieder. Nach einer kurzen Wartepause gibt sie weiterhin das Semaphore wieder frei, welches von der Funktion Open() allokiert wurde. Somit steht der FPGA-Gerätetreiber wieder zur Verfügung.

Adressen:

Zum Abschluß dieser Betrachtungen möchte ich die benutzten Steuersignale kurz bescheiben. Während die oben dargestellten Algorithmen für das Öffnen, Beschreiben und Schließen einer Gerätedatei für den Zugriff auf ein XC4000 Device unter VxWorks unabhängig vom speziellen Boarddesign sind, können sich die Adressen für die Generierung der Steuersignale in zukünftigen Versionen des Spyder-Core Systems ändern.

Signal	Adresse	Databit[a]	Zugriff[b]	Beschreibung
PROG	0xf3000000	0	w	aktiviert Programmiermode des FPGA
INIT	0xf3000000	0	r	Signal kann zur Fehlererkennung beim Ladevorgang dienen, hier nicht benutzt
SLAVE-OUT	0xf3000000	1	w	verbindet das FPGA mit dem System-BUS
DONE	0xf3000000	1	r	Signal wird aktiv, wenn der Konfigurationsvorgang erfolgreich abgeschlossen wurde
DATA	0xf3400000	0	w	dient der Datenübertragung zum FPGA

Tabelle 5.13: Signale und Adresse für SPYDER-CORE-P1[c]

a. Beim Power PC ist das Datenbit 0 das höchstwertige Bit
b. w.. Verhalten bei Schreibzugriff, r.. Verhalten bei Lesezugriff
c. Stand: Spyder-Core-P1, Mai 1999

Beispiel: Wenn man das Datenbit 1 auf Adresse 0xf3000000 mit einer binären Eins beschreibt, so aktiviert man das SLAVEOUT-Signal. Führt man auf die gleiche Adresse hingegen einen Lesezugriff aus, so spiegel Bit 0 den aktuellen Zustand des DONE-Signals wider.

5.5 Demonstrator: Erweiterer FTP-Server

5.5.1 Konzept

Der vorliegende FTP Server verdeutlicht auf anschauliche Weise, welche neuen Möglichkeiten sich durch die Bereitstellung eines Filesystems sowie eines Gerätetreibers für FPGA eröffnen:

Datenverwaltung: Der FTP-Server stellt all die Funktionalität bereit, welche benötigt wird, um Daten vom Entwicklungsrechner zum Spyder-Core-Board und zurück zu übertragen. Weiterhin sind fundamentale Filesystemoperationen wie das Anlegen, Löschen sowie Umbenennen von Dateien und Verzeichnissen möglich. Als Speichermedium auf Spyder-Core kommt sowohl ein RAM-Drive als auch das Flash-RAM in Betracht. Für beide Medien steht ein funktionsfähiges Filesystem zur Verfügung (siehe Abschnitt 5.4.1 und Abschnitt 5.4.2).

Zugriff auf Geräte: Der FTP-Server läßt sich so konfigurieren, daß Geräte wie FPGAs als Verzeichnisse dargestellt werden. Schreibzugriffe auf diese Verzeichnisse bewirken einen direkten Zugriff auf einen Gerätetreiber. Somit ist es z.B. möglich, ein Design File direkt vom Entwicklungsrechner in das entsprechende FPGA zu laden. Darüber hinaus besteht die Möglichkeit, Konfigurationsfiles, welche im Flash-RAM abgelegt wurden, in das FPGA zu kopieren.

Dabei erfolgt die gesamte Steuerung mit einem Standard FTP-Client. Damit ist die Integration des Spyder-Core-System in die Benutzeroberfläche eines Entwicklungsrechners mit einem beliebigen Betriebssystem möglich.

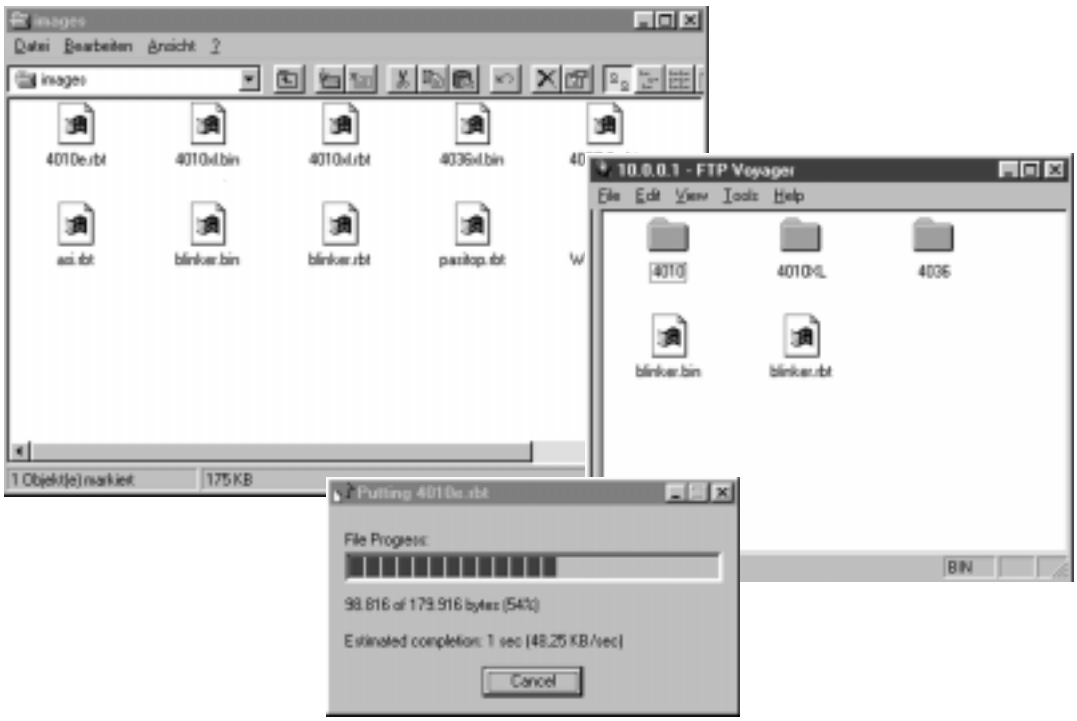


Abbildung 5.37 Integration des Spyder-Core Systems in die Benutzeroberfläche des Hostrechners

Abbildung 5.37 zeigt, wie sich das Spyder-Core-P1 System dem Benutzer eines Windows basierten Hostrechners darstellen kann. Das Fenster *Images* zeigt das Dateisystem des Hosts, während im Window des *FTP-Clients*[1] das Filesystem des Spyder Systems zu erkennen ist. Dabei ist der FTP Server so konfiguriert, daß alle Schreibzugriffe auf den Ordner *4010* an den Devicetreiber des FPGA weitergeleitet werden. Zieht man nun mit der Maus ein Design File, r.B. *4010e.rbt* auf den *Ordner 4010* im FTP Fenster, so wird automatisch ein FTP Upload gestartet und das FPGA sofort geladen.

Durch die ausschließliche Benutzung des FTP-Protokolls bestehen auch keine Einschränkungen bezüglich der Hardware des steuernden Computers. Ebenso ist es möglich, den Rechner, auf welchem die Benutzeroberfläche läuft, und das SPYDER-CORE-P1 System selbst beliebig räumlich zu trennen.

5.5.2 Realisierung

Die Einbindung der Gerätetreiber wird durch die Definition spezieller Verzeichnisse ermöglicht, denen der Name eines Gerätetreibers zugeordnet wird. Dies geschieht durch Editieren eines Headerfiles (ftpd.h):

```
#define SPECIALROOT "/fpga"
#define SPECIALDIRS "/fpga/4010:/fpga/4036:/fpga/4010XL"
#define DEVICENAMES   "/dev/fpga0: : "
```

Abbildung 5.38 Assoziation zwischen Verzeichnissen und Gerätetreibern

Beispiel (Abbildung 5.38): Die Pfade */fpga/4010*, */fpga/4036* und */fpga/4010XL* sind als Schnittstelle für Gerätetreiber reserviert. Diese Verzeichnisse werden immer vom FTP Server gelistet, auch wenn sie sich nicht innerhalb des Verzeichnisses /fpga[2] auf dem Datenträger (z.B. Flash-Memory) befinden. Existieren auf dem lokalen Filesystems des Spyderboards Verzeichnisse mit diesen Pfaden, werden sie vom FTP Server ignoriert.

In der letzten Zeile erkennt man, wie diesen Pfaden Gerätetreiber zugeordnet werden. So wird dem Verzeichnis */fpga/4010* der Gerätetreiber */dev/fpga0* zugeordnet, der z.B. für das FPGA der Serie XC4000 zuständig ist, welches sich auf Spyder-Core-P1 befindet. Den beiden anderen Pfaden ist kein Treiber zugeordnet. Schreibversuche auf diese Ordner werden mit einer entsprechenden Fehlermeldung zurückgewiesen.

Es existieren zwei grundsätzliche Möglichkeiten, ein FPGA oder ein anderes Gerät mit Hilfe des FTP-Servers zu bescheiben:

- **Upload:** Diese Variante kommt immer dann zur Ausführung, wenn eine Datei von einem entfernten Rechner, z.B. dem Entwicklungs-PC, zu einem Gerät auf Spyder-Core übertragen werden soll. Benutzt wird dabei der Befehl *STOR* des FTP Protokolls, welcher auf einigen textbasierten Clients auch über den Befehl *put* benutzt wird.

1. zu sehen ist der FTP Client „Voyager“
2. dieses Verzeichnis muß auf dem Datenträger vorliegen und dient gewissermaßen als Mountpoint

- **Umbenennen:** Liegt eine Datei schon auf einem lokalen Filesystem des Spyder-Systems, so realisiert der FTP-Server dies unter Benutzung der Befehle *RNFR* und *RNTO*, welche das FTP-Protokoll anbietet. Diese Funktionen werden auf textbasierten Clients meist unter dem Kommando *rename* zusammengefaßt. Während die Kommandos bei normalen Dateien tatsächlich nur ein Umbenennen bzw. Verschieben bewirken, ist ihre Arbeitsweise modifiziert, wenn als Ziel ein Verzeichnis gewählt wurde, hinter dem sich ein Gerätetreiber verbirgt.

In beiden Fällen untersucht der FTP-Server das Format einer Datei, bevor er sie an das entsprechende Gerät sendet. Auf diese Weise wird auch intern eine Konvertierung der verschiedenen Dateiformate auf ein gerätespezifisches Format realisiert. So konvertiert der FTP-Server z.B. Dateien, welche im Xilinx-Bitstream-ASCII Format (RBT) vorliegen, in ihr Binäräquivalent. Dateien mit unbekanntem Dateiformat werden mit einer entsprechenden Fehlermeldung abgelehnt.

Beispiel: Betrachten wir noch einmal Abbildung 5.37. Wenn der Anwender im Fenster des FTP-Clients versucht, die Datei *4010e.rbt* mit der Maus in den Ordner *4010* zu ziehen, so geschieht folgendes:

- Der Client schickt einen Befehl an den FTP-Server, die Datei *4010e.rbt* in den Ordner */fpga/4010* zu verschieben.
- Der Server erkennt, daß diesem Ordner ein Gerätetreiber zugeordnet ist
- Der FTP-Server erkennt an der Struktur der Datei, daß es ich um ein RBT-File handelt. Er konvertiert es entsprechend.
- Die ins Binärformat konvertierten Dateien werden an das FPGA gesendet

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit hatte zwei wesentliche Teile zum Inhalt. Zum einen bestand die Aufgabe, die Produktqualität des existierenden Emulationssystems Spyder-Core-P1 durch Bereitstellung spezifischer Softwarekomponenten zu erhöhen. Diese Komponenten dienen der Qualitätssicherung, Performanceanalyse und der Erhöhung des Bedienkomfort des Spyder-Core-Systems.

Die entwickelten Softwarekomponenten haben sich bewährt, so daß sie auch in zukünftigen Projekten des Forschungszentrums Informatik ihren Platz finden werden. Hervorzuheben ist dabei vor allem der Know-How-Zuwachs bei der Entwicklung eingebetteter Systeme. So unterstützt das FZI zur Zeit laufende Projekte sowohl in der Industrie als auch im universitären Bereich, wie z.B. an der Universität München[1].

Ein weiters Aufgabenfeld war, dem System neue Aufgabenfelder in der aktuellen Forschung zu erschließen. Das Spyder-Core System steht jetzt als Plattform für Untersuchungen auf dem Gebiet der Run Time Rekonfiguration bereit. Zur Zeit existiert ein Treiber für FPGAs der XC4000 Reihe von Xilinx.

Der Zeitraum, in dem diese Arbeit enstand, gestattete leider nur eine kurze Bewertung der Leistungsfähigkeit des aktuellen Spyder-Core-P1 Systems bezüglich der Anforderungen, welche durch die Run Time Rekonfiguration gestellt werden. Eine Analyse der Zeitressourcen, welche zum Umkonfigurieren nötig sind, zeigt folgendes Ergebnis:

1. Rekonfiguration von FPGA unter Windows NT

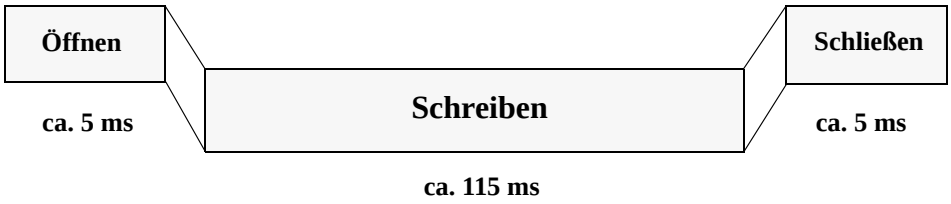


Abbildung 6.1 Performance von Spyder Core P1 bei der Rekonfiguration des FPGA

Wie in Abbildung 6.1 zu erkennen ist, liegt der Zeitbedarf für die Rekonfiguration des FPGA im oberen Millisekundenbereich. Somit eröffnen sich für Spyder-Core all diejenigen Forschungsfelder, in denen es eine komfortable, automatisierbare Rekonfigurierbarkeit der programmierbaren Logik erforderlich ist.

Ein sehr interessantes Beispiel aus diesem Bereich ist die Anwendung evolutionärer Algorithmen für den Entwurf von Hardwaredesigns, wie dies z.B. in [AT96] beschrieben wird. Im wesentlichen geht es bei diesem Konzept darum, Hardware nicht mehr auf die klassische Weise zu entwerfen, sondern vielmehr eine evolutionäre Lösung zu finden. Man wählt aus einer Reihe von anfangs zufällig generierten „Schaltungen“ diejenige aus, welche in ihrer Funktionalität dem vorgegebenen Ziel am nächsten kommt. Von dieser werden verschiedene Variationen erstellt, welche ihrerseits wiederum bewertet werden. Mit jedem Zyklus „lernt“ die Hardware hinzu. Auf diese Weise ist es unter Umständen möglich, „automatisch“ Lösungen zu finden oder vereinfacht gesprochen, ein System zu entwickeln, welches sich selbst den gestellten Aufgaben anpaßt.

Andere Aufgabenbereiche hingegen stellen wesentlich höhere Anforderungen an die Geschwindigkeit des Systems, insbesondere an die Ladezeiten des FPGA. Darunter zählen z.B. Anwendungen wie ein ATM Monitor, wie er in [KW99] beschrieben wurde.

Daher werden neue Technologien wie die Unterstützung der Virtex-Architektur von Xilinx den weiteren Entwicklungsweg des Spyder System prägen. Die komponentenorientierte Struktur (Filesystem, FPGA-Treiber, Anwendungen wie FTP-Server) des Softwarepaketes gestattet es dabei, diese neuen Hardwarekomponenten innerhalb relativ kurzer Zeit auch für die Arbeiten verfügbar zu machen, welche sich mit Problemen der Run Time Rekonfiguration beschäftigen.

Anhang A

Referenzen

[Act94] ___: *FPGA-Data Book and Design Guide.* Actel, Inc. 1994

[Army96] __: Real Time Executive for Multiprocessor Systems. Development Environment Guide. U.S. Army Missile Command, Redstone Arsenal, Alabama 1996

[AT96] Adrian Thompson: *An evolved circuit, intrinsic in silicon, entwined with physics.* COGS, University of Sussex, Brighton, BN1 9QH, UK

[Atmel1] __:*Programming Specifications for Atmel's FPGA Configuration EEPROMS AT17C65/128/256/010 AT17C65A/128A/256A/512A/010A*

[CN97] Carsten Nitsch: Power-PC-Tools, FZI Karlsruhe und Universtät Leipzig 1997

[DeMi95] G.De Micheli: Hardware/Software Co-Design: Application Domains and Design Technilogies. Proceedings of the NATO Advanced Study Institute on Hardware/ Software Co-Design, Tremezzo Italy, Kluwer Academic Publishers, 1995

[Fu97] MB86960 Network Interface Controller with Encoder/Decoder (NICE). Data Sheet, Fujitsu 1997

[HaHu95] J.D. Hadley, B.L.Hutchings: *Design Methologies for Partial Reconfigured Systems.* Dep. of Electrical and Computer Eng. Brigham Young University Provo, UT 84602, 1995

[HePa96] J. Hennessy, D.Patterson: Computer Architecture - A Quantitative Approach. Second Edition, Morgan Kaufmann Publishers, Inc. SF-CA-USA, 1996

[HM93] Hans-Peter-Messmer. *PC-Hardwarebuch.* Addison-Wesley 1993

[HuWi95] B. Hutchings, M. Wirthlin: *Implementation Approaches for Reconfigurable Logic Applications*. Proceedings of 5th International Workshop of Field-Programmable Logic and Applications, Oxford 1995

[KW99] Karlheiz Weiß: Systematische Architekturfindung und Emulation eingebetteter Systeme. Karlsruhe 1999

[MS97] Mike Schubert: *Echtzeitbetriebssysteme.* Vortrag an der HTW Dresden, 1997, http://wwwbs.informatik.htw-dresden.de/svortrag/ai94/Schubert/RTFolien.html

[St90] J. Stankovic, K.Ramamritham: *What is predictability for Real-Time Systems, 1990*

[St99] Thorsten Steckstor: *Evaluierung der Vorteile des Einsatzes eines Echtzeitbetriebssystems in einem eingebetteten System am Beispiel von ASI.* FZI Karlsruhe 1999

[Wind97] __: Tornado User´s and Programmer´s Guide. Windriver Systems,Inc. 1997

[WirHut97] M. Wirthlin, B. Hutchings: *Improving Functional Density Through Run-Time Constant Propagation.* ACM-Symposium on Field Programmable Gate Arrays, 1997.

[Wo94] W. Wolf: Hardware-Software Co-Design of Embedded Systems. Procceedings of the IEEE, Vol.82, No.7, July 1994

[WSNRo99] K. Weiß, T. Steckstor, C. Nitsch, W. Rosenstiel: *Performance Analysis of Real-Time-Operation Systems by Emulation of an Embedded System.* 10th IEEE International Workshop on Rapid System Prototyping (RSP), Clearwater, Florida, USA, 1999

[Xil99] ___: *Virtex 2,5V Field Programmable Gate Arrays.* Xilinx, Inc. Datasheet, http://www.xilinx.com, 1999

[Xil98/1] ___: *The Programmable Logic Data Book.* Xilinx, Inc. 1998

Anhang B

Loader

```
/*-------------------------------------------------------------------------

                              Programm Loader
                             =================
  Datei: loader21.c

 laedt Sektionen aus einem ELF-File in dem Speicher des Power PC

-------------------------------------------------------------------------*/



#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include „defines.h“
#include „elf.h“
#include „loader.h“

#ifdef BCC
#include <dos.h>
#endif



#define VERSION „Loader Version 2.1“

char *SectName[256];

void help(char *s)
{
   fprintf(stderr, „usage  %s [-s] [-mbig=<1|0>] <filename>\n“, s);
   fprintf(stderr, „            -s ..         reverse Byteorder \n“);
```

```
   fprintf(stderr, „            -mbig=1       target = big endian\n“);
   fprintf(stderr, „            -mbig=0       target = little endian\n“);
   fprintf(stderr, „            -V            version\n“);
   fprintf(stderr, „            -v            verbose\n“);
   fprintf(stderr, „            -c            do not initialize ppc\n“);
   fprintf(stderr, „            -r            do not start application\n“);
   fprintf(stderr, „            -d<base>      baseaddresse (hex)\n“);
   fprintf(stderr, „            -h , --help   help\n\n“);
   exit(-1);
}

long GetVal(char *arg, long i, long b)
{
   char *s;

   if (arg[2] == , ,)
      s = arg + i + 1;
   else
      s = arg + i;
   if (s)
      return strtol(s, NULL, b);
   else
      return -1;
}

void version()
{
   fprintf(stderr, VERSION);
   fprintf(stderr, „\n\n“);
   exit(0);
}

void main(int argc, char **argv)
{

   FILE *f, *g;
   long ByteOrder = -1;
   char FileName[256];
   char buf[4096], buf1[4096];
   long i, x, rsize;
   long address, size;
   long base = -1;
   long verbose = 0;
   long initppc = 1;
   long run = 1;
   Elf32_Ehdr stElfHdr;
   Elf32_Phdr(*stProgHdrs)[];
   Elf32_Phdr(*stSectHdrs)[];
   Elf32_Shdr *SecHdr;
//StrArray SectName[64];

   long found;
   char foo;

   if (argc < 2) {
      help(argv[0]);
      exit(-1);
   } else {
      for (i = 1; i < argc; i++)
    if ((char *) strstr(argv[i], „-s“) == argv[i])
       ByteOrder = REVERSE_BYTE_ORDER;
    else if ((char *) strstr(argv[i], „-mbig=“) == argv[i]) {
       printf(„DEBUG: %d\n“, GetVal(argv[i], 6, 10));
```

```
    if (GetVal(argv[i], 6, 10) == 0)
       ByteOrder = REVERSE_BYTE_ORDER;
    else
       ByteOrder = DEFAULT_BYTE_ORDER;
 } else if (((char *) strstr(argv[i], „-h“) == argv[i]) ||
       ((char *) strstr(argv[i], „--help“) == argv[i]))
    help(argv[0]);
 else if ((char *) strstr(argv[i], „-V“))
    version();
 else if ((char *) strstr(argv[i], „-d“))
    base = GetVal(argv[i], 2, 16);
 else if ((char *) strstr(argv[i], „-v“))
    verbose = 1;
 else if ((char *) strstr(argv[i], „-c“))
    initppc = 0;
 else if ((char *) strstr(argv[i], „-r“))
    run = 0;
 else
    strcpy(FileName, argv[i]);
}


if (base == -1)
   base = 0xD000;


f = fopen(FileName, „rb“);
if (f == NULL) {
   fprintf(stderr, „/Can not open %s\n“, FileName);
   perror(„“);
   exit(-1);
}
/* Wenn ByteOrder nicht per Option angegeben wurde, automatisch
   feststellen
 */

if (ByteOrder == -1) {
   fseek(f, 5, SEEK_SET);
   fread(&x, 1, 1, f);
   if (x == 1)
 ByteOrder = REVERSE_BYTE_ORDER;/* little */
   else
 ByteOrder = DEFAULT_BYTE_ORDER;/* big */
   fseek(f, 0, SEEK_SET);
}
/* Alle Headerinformationen aus ELF/File lesen */

ReadELFHeaders(f, &stElfHdr, &stProgHdrs, &stSectHdrs,
     SectName, ByteOrder);
if (verbose) {
   printf(VERSION);
   printf(„\n\n“);
   printf(„Byteorder des ELF-Files: „);
   if (ByteOrder == REVERSE_BYTE_ORDER)
 printf(„Little Endian\n)“);
   else
 printf(„Big Endian\n“);
   printf(„DualPortRAM-Adresse: 0x%x\n“, base);
   printf(„\n“);
   printf(„Analyse des ELF-Files: \n\n“);
   PrintELFHeaders(stdout, &stElfHdr, &stProgHdrs, &stSectHdrs, SectName);
}
x = 1;
```

```
if (verbose)
   printf(„Versetze PPC in reset-Zustand\n“);

/* Aktiviere ISA Bus */
printf(„ReadChunk(base, 0x807, &foo, 1);\n“);
ReadChunk(base, 0x807, &foo, 1);
printf(„done\n“);
foo &= 0xfe;
WriteChunk(base, 0x807, &foo, 1);

//WriteRegister(base, REG_GNET, &x);       /* PCC in reset */

foo = 0x00;
WriteChunk(base, REG_GNET, &foo, 1);
sleep(1);


if (initppc) {

   if (verbose)
 printf(„initialisiere PPC-Speicher\n“);
   for (i = 0; i < 0x800; i++) {
 buf[i] = 0xaa;
 buf1[i] = 0x00;
   }
   WriteChunk((short) base, 0, (char *) &buf, 0x800);
} else if (verbose)
   printf(„initialisiere PPC-Speicher nicht\n“);


//exit(-1);

/* Test, ob die Sektionen .rvector, .init_dp und .copy im File
   stehen. Falls Option -c (nur kopieren) nicht gesetzt ist,
   Abbruch mit Fehlermeldung */

found = 0;
if (stElfHdr.e_shnum)
   for (i = 0; i < stElfHdr.e_shnum; i++) {
 if (!strcmp(SectName[i], „.rvector“))
    found |= 0x01;
 if (!strcmp(SectName[i], „.entry“))
    found |= 0x02;
 if (!strcmp(SectName[i], „.copy“))
    found |= 0x04;
   }
if (initppc) {
   if (!(found & 0x01)) {
 fprintf(stderr, „Sektion .rvector nicht gefunden.\n“);
 exit(-1);
   }
   if (!(found & 0x02)) {
 fprintf(stderr, „Sektion .entry nicht gefunden.\n“);
 exit(-1);
   }
   if (!(found & 0x04)) {
 fprintf(stderr, „Sektion .copy nicht gefunden.\n“);
 exit(-1);
   }
}
/* ------------------------------------------------------------ */
```

```
/* Falls (initppc==1), so lade die Sektionen .rvector, .init_dp
   und .copy direkt
 */



if (stElfHdr.e_shnum)
   for (i = 0; i < stElfHdr.e_shnum; i++) {
 if ((!strcmp(SectName[i], „.rvector“)) ||
     (!strcmp(SectName[i], „.entry“)) ||
     (!strcmp(SectName[i], „.copy“))
     ) {
    SecHdr = (Elf32_Shdr *) (*stSectHdrs) + i;
    printf(„Lade Section %s\n“, SectName[i]);
    if (verbose) {
       printf(„Adresse:     0x%lx  (0x%lx)\n“,
         SecHdr->sh_addr, SecHdr->sh_addr - DPROM);
       printf(„Groesse:     0x%lx\n\n“, SecHdr->sh_size);
    }
    fseek(f, SecHdr->sh_offset, SEEK_SET);
    rsize = fread(buf, 1, SecHdr->sh_size, f);
    WriteChunk(base, (SecHdr->sh_addr - DPROM), &buf, SecHdr->sh_size);
 }
   }
/* ------------------------------------------------------------------- */


/* PPC starten */
if (verbose)
   printf(„PPC beginnt Codeausfuehrung ab Adresse 0xFFFFFFFC\n“);
sleep(1);
x = 0;

WriteRegister(base, MSG_PC, &x);/* MSG_PC=COM_POLL     */


/* WriteRegister(base, REG_GNET, &x); *//* PPC aus reset holen */


foo = 01;
WriteChunk(base, REG_GNET, &foo, 1);

/* Alle Sektionen laden, die ausfuehrbaren Code oder initialisierte
   Daten enthalten. Groesse muss > 0 sein.
   ( SecHdr->sh_type == SHT_PROGBITS und SecHdr->sh_flags != 0
   und SecHdr->sh_size >0 )
   Liegt die Zieladresse nicht im DualMapRAM, so wird die Copyroutine
   benutzt
 */



if (verbose) {
   printf(„Copyroutine gestartet.\n Enter fuer weiter\n“);
   getc(stdin);
}
if (stElfHdr.e_shnum)
   for (i = 0; i < stElfHdr.e_shnum; i++) {
 SecHdr = (Elf32_Shdr *) (*stSectHdrs) + i;

 if ((SecHdr->sh_type == SHT_PROGBITS) &&
     (SecHdr->sh_flags) &&
```

```
         (SecHdr->sh_size) &&
         (strcmp(SectName[i], „.rvector“)) &&
         (strcmp(SectName[i], „.init_dp“)) &&
         (strcmp(SectName[i], „.copy“))
         ) {
        printf(„Kopiere Section %s\n“, SectName[i]);
        if (verbose) {
           printf(„Adresse:     0x%lx  (0x%lx)\n“,
             SecHdr->sh_addr, SecHdr->sh_addr - DPROM);
           printf(„Groesse:     0x%lx\n\n“, SecHdr->sh_size);
        }
        CopySection(SectName[i], base, &stElfHdr, &stSectHdrs, SectName, f);
     }
        }
/* ------------------------------------------------------------------------ */



   /* Falls run==1 , copyroutine an dieser Stelle beenden                   */

   if (verbose)
      printf(„Kopierroutine beendet\n“);
   if (run) {
      x = COMM_STOP;
      WriteRegister(base, MSG_PC, &x);
   }
   fclose(f);

   /*
      sleep(3);
      f = fopen(„MEM.log“,“w“);
      ReadChunk( base, 0, &buf1, 0x0800);
      fwrite(buf1, 1, 0x0800, f);
      fclose(f);
    */

   exit(0);
}

/*-----------------------------------------------------------------------------

                              Programm Loader
                              ---------------
  Datei: copy.c

*-----------------------------------------------------------------------------/

#include „elf.h“
#include „loader.h“
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


long CopySection(char *name,
      short base,
      Elf32_Ehdr * stElfHdr,
      Elf32_Shdr(**stSectHdrs)[],
      char *SectName[],
      FILE * elf_file)
{
   long i, found;
```

```
Elf32_Shdr *SecHdr;
char buf[1024];
long to_copy, msg;
long address, rsize;


if (!stElfHdr->e_shnum) {
   printf(„Es existieren keine Sections.\n“);
   exit(-1);
}
found = 0;
for (i = 0; i < stElfHdr->e_shnum; i++) {
   if (!strcmp(name, SectName[i])) {
 SecHdr = (Elf32_Shdr *) (*stSectHdrs) + i;
 found = 1;
 i = stElfHdr->e_shnum;
   }
}

if (!found) {
   printf(„Keine Section %s gefunden\n“, name);
   exit(-1);
}
to_copy = SecHdr->sh_size;
address = SecHdr->sh_addr;
fseek(elf_file, SecHdr->sh_offset, SEEK_SET);

while (to_copy) {
   if (to_copy >= sizeof(buf))/* Daten schreiben */
 rsize = fread(buf, 1, sizeof(buf), elf_file);
   else
 rsize = fread(buf, 1, to_copy, elf_file);

   msg = COMM_POLL;
   WriteRegister((short) base, (short) MSG_PC, (char *) &msg);

   //printf („FNORD\n“);
   WriteChunk(base, 0x0000, (char *) &buf, rsize);

   /* ADDRESS schreiben */
   WriteRegister(base, (short) ADDRESS + 0, (char *) (&address) + 3);
   WriteRegister(base, (short) ADDRESS + 1, (char *) (&address) + 2);
   WriteRegister(base, (short) ADDRESS + 2, (char *) (&address) + 1);
   WriteRegister(base, (short) ADDRESS + 3, (char *) (&address) + 0);

   /* SIZE schreiben */
   WriteRegister(base, (short) SIZE + 0, (char *) (&rsize) + 1);
   WriteRegister(base, (short) SIZE + 1, (char *) (&rsize) + 0);

   msg = 5;
   WriteRegister(base, (short) MSG_PPC, (char *) &msg);
   msg = COMM_COPY;
   WriteRegister(base, (short) MSG_PC, (char *) &msg);
   do {
 ReadRegister(base, (short) MSG_PPC, (char *) &msg);
   }
   while (msg != COMM_COPY);

   to_copy -= rsize;
   address += rsize;
}

return 0;
```

```
}

/*---------------------------------------------------------------------------

                            Programm Loader
                          ------------------
  Datei: comm2.c

---------------------------------------------------------------------------*/


#include <stdlib.h>
#include <stdio.h>
#include „elf.h“
#include „loader.h“


#ifdef BCC
#include <dos.h>
#endif



/**
    Funktion WriteChunk.
    Argumente:long  base       .. Basisadresse im PC Adressraum
              long  offset
              char *chunk      .. puffer mit daten, die zu schreiben sind
              long  chunksize  .. Anzahl der zu schreibenden Bytes
    Rueckgabe: Anzahl der geschriebenen Byte
*/

#ifndef DOS
int WriteChunk(long base, long offset, char *chunk, long chunksize)
{
   FILE *mem;
   long errorcode, wsize, address;

   address = (base << 4) + offset;

   mem = fopen(„/dev/mem“, „w“);
   if (mem == NULL) {
      perror(„....Can not open /dev/mem“);
      exit(-1);
   }
   errorcode = fseek(mem, address, SEEK_SET);
   if (errorcode != 0) {
      perror(„Can not seeklong o /dev/mem“);
      exit(-1);
   }
   wsize = fwrite(chunk, 1, chunksize, mem);
   if (wsize != chunksize) {
      perror(„Can not write to /dev/mem“);
      exit(-1);
   }
   fclose(mem);
   return wsize;
}
#endif

/* Some people still use it :-)  */
#ifdef DOS
long WriteChunk(short base, short offset, char *chunk, long chunksize)
```

```
{
   unsigned char *address;
#ifdef BCC
   unsigned long i;
#endif

#ifdef GCC
   /* this will not work with virtual address space */
   address = (unsigned char *) (base << 4) + offset;
   memcpy(address, chunk, chunksize);
#endif

#ifdef   /* 8086 real mode :(                      */
   for (i = 0; i < chunksize; i++) {
      pokeb(base, offset + i, *(chunk + i));
   }
#endif

   /* fuer kompatibilitaet mit linux version */
   return chunksize;
}
#endif



/*****************************************************************************/

/**
    Funktion ReadChunk.
    Argumente:long  base       .. Basisadresse im PC Adressraum
         long  offset
          char *chunk    .. Puffer, in welchen die Daten gelesen werden
         long  chunksize  .. Anzahl der zu lesenden Bytes
    Rueckgabe: Anzahl der gelesenen Byte
*/

#ifndef DOS
long ReadChunk(long base, long offset, char *chunk, long chunksize)
{
   FILE *mem;
   long errorcode, rsize, address;

   address = (base << 4) + offset;
   printf(„DPRAM: 0x%x\n“, address);

   mem = fopen(„/dev/mem“, „r“);
   if (mem == NULL) {
      perror(„.Can not open /dev/mem“);
      exit(-1);
   }
   errorcode = fseek(mem, address, SEEK_SET);
   if (errorcode != 0) {
      perror(„Can not seeklong o /dev/mem“);
      exit(-1);
   }
   rsize = fread(chunk, 1, chunksize, mem);
   if (rsize != chunksize) {
      perror(„Can not read from /dev/mem“);
      exit(-1);
   }
   fclose(mem);
   return rsize;
}
```

```
#endif


#ifdef DOS
long ReadChunk(short base, short offset, char *chunk, long chunksize)
{
   unsigned char *address;
#ifdef BCC
   unsigned long i;
#endif

#ifdef GCC
   address = (unsigned char *) (base << 4) + offset;
   printf(„dpram adresse: 0x%x\n“, (int) address);
   memcpy(chunk, address, chunksize);
#endif

#ifdef BCC
   for (i = 0; i < chunksize; i++)
      *(chunk + i) = peekb(base, offset + i);
#endif

   return chunksize; /* for compatibility */
}
#endif

/*****************************************************************************/


#ifndef DOS
void WriteRegister(long base, long reg, char *val)
{
   FILE *mem;
   long errorcode, wsize, address;

   address = (base << 4) + reg;

   mem = fopen(„/dev/mem“, „w“);
   if (mem == NULL) {
      perror(„..Can not open /dev/mem“);
      exit(-1);
   }
   errorcode = fseek(mem, address, SEEK_SET);
   if (errorcode != 0) {
      perror(„Can not seeklong o /dev/mem“);
      exit(-1);
   }
   wsize = fwrite(val, 1, 1, mem);
   if (wsize != 1) {
      perror(„Can not write to /dev/mem“);
      exit(-1);
   }
   fclose(mem);

}
#endif


#ifdef DOS
void WriteRegister(short base, short reg, char *val)
{
   unsigned char *address;
#ifdef BCC
```

```
   unsigned long i;
#endif

#ifdef GCC
   address = (unsigned char *) (base << 4) + reg;
   *address = *val;
#endif

#ifdef BCC
   //printf(„DEBUG: 0x%02x --> 0x%04x:0x%04x\n“ , *val, base, reg);
   pokeb(base, reg, *val);
#endif

}

#endif


/*************************************************************************/

#ifndef DOS
void ReadRegister(long base, long reg, char *val)
{
   FILE *mem;
   long errorcode, rsize, address;
   char x;

   address = (base << 4) + reg;

   mem = fopen(„/dev/mem“, „r“);
   if (mem == NULL) {
      perror(„...Can not open /dev/mem“);
      exit(-1);
   }
   errorcode = fseek(mem, address, SEEK_SET);
   if (errorcode != 0) {
      perror(„Can not seeklong o /dev/mem“);
      exit(-1);
   }
   rsize = fread(val, 1, 1, mem);
   if (rsize != 1) {
      perror(„Can not read from /dev/mem“);
      exit(-1);
   }
   fclose(mem);

}
#endif


#ifdef DOS
void ReadRegister(short base, short reg, char *val)
{
   unsigned char *address;

#ifdef GCC
   address = (unsigned char *) (base << 4) + reg;
   *val = *address;
#endif

#ifdef BCC

   *val = peekb(base, reg);
```

```
   //printf(„DEBUG: 0x%02x <-- 0x%04x:0x%04x\n“ , *val, base, reg);
#endif

}

#endif

/*-------------------------------------------------------------------------

                        Programm Loader
                       ----------------
  Datei: readelf.c

-------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include „elf.h“
#include „loader.h“

#ifdef DOS
#include <dos.h>
#endif


/**  Funktion StructRead. ************************************************

    Autor: Andre Hergenhan

    Liest einen ELFHeader, ProgramHeader bzw. SectionHeader aus einem
    ELF-File aus.
    Argumente:   stream .. Filehandler auf ELF-File
                 target .. Struktur vom Typ - Elf32_Ehdr (ElfHeader) oder
                                            - Elf32_Shdr (SectionHeader) oder
                                            - Elf32_Phdr (ProgramHeader)
                 format .. String, der die Struktur des zu lesenden Headers
                           anbit. Format besteht aus den Ziffern 1,2 und 4.
                           1: Lese ein Byte
                           2: Lese ein 16 bit Wort
                           4: Lese ein 32 bit Wort

                 ByteOrder: DEFAULT_BYTE_ORDER: ByteOrder des Hosts
                            REVERSE_BYTE_ORDER: Byteorder ist der
                                                Byteorder des
                                                Hosts umgekehrt

 ************************************************************************/


int StructRead(
      FILE * stream, unsigned char *target, const char *format,
      int ByteOrder)
{
   unsigned char *t;
   const char *f;
   unsigned long xxx;

   t = (unsigned char *) target;
   f = format;
```

```
   for (;;) {
      switch (*f++) {
      case 0:
    return (NoError);
      case ‚1‘:
    *t++ = getc(stream);
    printf(„%02x „, *(t - 1));
    /* Test, ob Datei ein ELF-File ist */
    if ((!strcmp(format, STRUCT_ELFHDR_FORMAT)) &&
        ((t - (char *) target) > 3)
        ) {
       if (
        (char *) strstr((char *) target, „ELF“) !=
        ((char *) target + 1)) {
          fprintf(stderr, „This is not an ELF File.\n“);
          exit(-1);
       }
    }
    break;
      case ‚2‘:
    if (ByteOrder == DEFAULT_BYTE_ORDER) {
       t += 2;
       *--t = getc(stream);
       *--t = getc(stream);
       t += 2;
    } else {
       *t++ = getc(stream);
       *t++ = getc(stream);
    }

    break;
      case ‚4‘:
    if (ByteOrder == DEFAULT_BYTE_ORDER) {
       t += 4;
       *--t = getc(stream);
       *--t = getc(stream);
       *--t = getc(stream);
       *--t = getc(stream);
       t += 4;

    } else {
       *t++ = getc(stream);
       *t++ = getc(stream);
       *t++ = getc(stream);
       *t++ = getc(stream);
    }
    break;
      };
   };
};



/**  Funktion ReadELFHeader. ********************************************

    Liest den ELFheader, alle Program- und alle Sectionheader aus einem
    ELF-File aus.
    Argumente:   elf_file   .. Filehandler auf ELF-File
                 stElfHdr   .. Struct fuer ELF-header
                 stProgHdrs .. Enthaelt alle Programheader
                 stSectHdrs .. Enthaelt alle Sectionheader
                 SectName   .. Enthaelt Namen der Sections
```

```
                ByteOrder  .. DEFAULT_BYTE_ORDER: ByteOrder des Hosts
                              REVERSE_BYTE_ORDER: Byteorder ist der
                                                  Byteorder des
                                                  Hosts umgekehrt

 ***************************************************************************/


long ReadELFHeaders(FILE * elf_file,
         Elf32_Ehdr * stElfHdr,
         Elf32_Phdr(**stProgHdrs)[],
         Elf32_Shdr(**stSectHdrs)[],
         char *SectName[],
         long ByteOrder)
{
   long i, j, k, offset;
   void *pVoid;
   long StringTableOffset, StringTableEntryOffset;
   long StringTableSize, StringSize;
   Elf32_Shdr *SecHdr;
   char c;
   char s_temp[256], *s;

   fseek(elf_file, 0x0L, SEEK_SET);


   /* read elf header */

   printf(„DEBUG: Struct read\n“);
   StructRead(elf_file, stElfHdr, STRUCT_ELFHDR_FORMAT, ByteOrder);

   /* read n program header */
   if (stElfHdr->e_phnum) {
      pVoid = calloc(stElfHdr->e_phnum, PHDRSZ);
      *stProgHdrs = (void *) pVoid;
      fseek(elf_file, stElfHdr->e_phoff, SEEK_SET);
      for (i = 0; i < stElfHdr->e_phnum; i++) {
    offset = (long) PHDRSZ *i + (long) *stProgHdrs;

    StructRead(elf_file,
          (void *) offset,
          STRUCT_PROGHDR_FORMAT, ByteOrder);
      }
   }
   /* read n section header     */


   if (stElfHdr->e_shnum) {
      pVoid = calloc(stElfHdr->e_shnum, SHDRSZ);
      *stSectHdrs = (void *) pVoid;
      fseek(elf_file, stElfHdr->e_shoff, SEEK_SET);


      for (i = 0; i < stElfHdr->e_shnum; i++) {
    offset = (long) SHDRSZ *i + (long) *stSectHdrs;
    StructRead(
           elf_file,
           (void *) offset,
           (const char *) STRUCT_SECTHDR_FORMAT,
           ByteOrder);
      }

      for (i = 1; i < stElfHdr->e_shnum; i++) {
```

```
    k = stElfHdr->e_shstrndx;/* HeaderNo fuer */
    /* Stringtable   */
    SecHdr = (Elf32_Shdr *) (*stSectHdrs) + k;
    StringTableOffset = SecHdr->sh_offset;
    StringTableSize = SecHdr->sh_size;


    SecHdr = (Elf32_Shdr *) (*stSectHdrs) + i;
    StringTableEntryOffset = SecHdr->sh_name;

    if ((StringTableEntryOffset >= 0) &&
        (StringTableEntryOffset < StringTableSize)) {
       fseek(elf_file, StringTableOffset + StringTableEntryOffset,
        SEEK_SET);

       c = 1;
       strcpy(s_temp, „“);
       for (j = 0; c != 0; j++) {
          c = getc(elf_file);
          strncat(s_temp, &c, 1);
       }

       s = (char *) malloc(strlen(s_temp) + 1);
       strcpy(s, s_temp);
       SectName[i] = s;

    } else {
       s = (char *) malloc(7);
       strcpy(s, „[NULL]“);
       SectName[i] = s;
    }




      }


   }
   return (NoError);
}


/**  Funktion PrintELFHeader. ******************************************

    Gibt den Inhalt des ELFheader sowie  aller Program- und Sectionheader
    aus
    Argumente:   elf_logfile .. Filehandle. Ausgabe wird hier
                                gelogt.
                 stElfHdr    .. Struct fuer ELF-header
                 stProgHdrs  .. Enthaelt alle Programheader
                 stSectHdrs  .. Enthaelt alle Sectionheader
                 SectName    .. Anthaelt Namen der Sections

 ************************************************************************/



long PrintELFHeaders(FILE * elf_logfile,
          Elf32_Ehdr * stElfHdr,
          Elf32_Phdr(**stProgHdrs)[],
          Elf32_Shdr(**stSectHdrs)[],
```

```
            char *SectName[]
  )
  {
     long i;
     Elf32_Shdr *SecHdr;
     Elf32_Phdr *PrgHdr;

     fprintf(elf_logfile, „ELF-Header \n\n“);
     fprintf(elf_logfile, „    e_ident:     %c“, stElfHdr->e_ident[0]);
     fprintf(elf_logfile, „%c“, stElfHdr->e_ident[1]);
     fprintf(elf_logfile, „%c“, stElfHdr->e_ident[2]);
     fprintf(elf_logfile, „%c“, stElfHdr->e_ident[3]);
     fprintf(elf_logfile, „ %X“, stElfHdr->e_ident[4]);
     fprintf(elf_logfile, „ %X“, stElfHdr->e_ident[5]);
     fprintf(elf_logfile, „ %X“, stElfHdr->e_ident[6]);
     fprintf(elf_logfile, „ %09X\n“, stElfHdr->e_ident[7]);
     fprintf(elf_logfile, „    e_type:      0x%X \n“, stElfHdr->e_type);
     fprintf(elf_logfile, „    e_machine:   0x%X \n“, stElfHdr->e_machine);
     fprintf(elf_logfile, „    e_version:   0x%lX \n“, stElfHdr->e_version);
     fprintf(elf_logfile, „    e_entry:     0x%lX \n“, stElfHdr->e_entry);
     fprintf(elf_logfile, „    e_phoff:     0x%lX \n“, stElfHdr->e_phoff);
     fprintf(elf_logfile, „    e_shoff:     0x%lX \n“, stElfHdr->e_shoff);
     fprintf(elf_logfile, „    e_flags:     0x%lX \n“, stElfHdr->e_flags);
     fprintf(elf_logfile, „    e_ehsize:    0x%X \n“, stElfHdr->e_ehsize);
     fprintf(elf_logfile, „    e_phentsize: 0x%X \n“, stElfHdr->e_phentsize);
     fprintf(elf_logfile, „    e_phnum:     0x%X \n“, stElfHdr->e_phnum);
     fprintf(elf_logfile, „    e_shentsize: 0x%X \n“, stElfHdr->e_shentsize);
     fprintf(elf_logfile, „    e_shnum:     0x%X \n“, stElfHdr->e_shnum);
     fprintf(elf_logfile, „    e_shstrndx:  0x%X \n\n\n“, stElfHdr->e_shstrndx);


     if (stElfHdr->e_phnum) {
        for (i = 0; i < stElfHdr->e_phnum; i++) {
      PrgHdr = (Elf32_Phdr *) (*stProgHdrs) + i;
      fprintf(elf_logfile, „Program Header %d \n\n“, i);
      fprintf(elf_logfile, „   p_type:       0x%lX \n“, PrgHdr->p_type);
      fprintf(elf_logfile, „   p_offset:     0x%lX \n“, PrgHdr->p_offset);
      fprintf(elf_logfile, „   p_vaddr:      0x%lX \n“, PrgHdr->p_vaddr);
      fprintf(elf_logfile, „   p_paddr:      0x%lX \n“, PrgHdr->p_paddr);
      fprintf(elf_logfile, „   p_filesz:     0x%lX \n“, PrgHdr->p_filesz);
      fprintf(elf_logfile, „   p_memsz:      0x%lX \n“, PrgHdr->p_memsz);
      fprintf(elf_logfile,
        „p_flags:      0x%lX (4=r 2=w 1=x)\n“, PrgHdr->p_flags);
      fprintf(elf_logfile, „   p_align:      0x%lX \n\n\n“, PrgHdr->p_align);
        }
     }
     if (stElfHdr->e_shnum) {
        for (i = 0; i < stElfHdr->e_shnum; i++) {
      SecHdr = (Elf32_Shdr *) (*stSectHdrs) + i;
      fprintf(elf_logfile, „Section Header %d \n\n“, i);
      fprintf(elf_logfile, „   sh_name:      %s \n“, SectName[i]);
      fprintf(elf_logfile,
        „   sh_type:      0x%lX (1=Code, 2=symtab, 3=strtab, 8=uninitial.Daten)\n“
        ,SecHdr->sh_type);
      fprintf(elf_logfile,
            „   sh_flags:     0x%lX (4=execinstr 2=alloc  1=write)\n“,
        SecHdr->sh_flags);
      fprintf(elf_logfile, „   sh_addr:      0x%lX \n“, SecHdr->sh_addr);
      fprintf(elf_logfile, „   sh_offset:    0x%lX \n“, SecHdr->sh_offset);
      fprintf(elf_logfile, „   sh_size:      0x%lX \n“, SecHdr->sh_size);
      fprintf(elf_logfile, „   sh_link:      0x%lX \n“, SecHdr->sh_link);
      fprintf(elf_logfile, „   sh_info:      0x%lX \n“, SecHdr->sh_info);
      fprintf(elf_logfile, „   sh_addralign: 0x%lX \n“, SecHdr->sh_addralign);
```

```
    fprintf(elf_logfile, „   sh_entsize:   0x%lX \n\n\n“,
       SecHdr->sh_entsize);
       }
   }
   return (NoError);
}

/*-------------------------------------------------------------------------

                              Programm Loader
                              ---------------

       Datei: defines.h

-------------------------------------------------------------------------*/

#define BIG 0
#define LITTLE 1


/*-------------------------------------------------------------------------

                            Programm Loader
                             --------------

Dateien elf.h, elftypes.h


Headerdateien des DIAB Crosscompilers

-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------

                          Programm Loader
                           -------------

Dateien elf.h, elftypes.h


siehe Headerdateien des DIAB Crosscompilers

-------------------------------------------------------------------*/

/*-------------------------------------------------------------------

                          Programm Loader
                           -------------

Datei: Makefile
-------------------------------------------------------------------*/



######################### Makefile fuer Loader ##########################
#
# Zielplattform: Linux Box
#
#########################################################################

INC=../../include
BINDIR=../../bin/linux
BIN= loader21


all:   $(BIN) rm_obj

loader21:loader21.o readelf.o comm2.o copy.o
     if [ ! -d $(BINDIR) ] ; then mkdir $(BINDIR); fi
     $(CC) loader21.o readelf.o comm2.o copy.o -o $(BINDIR)/loader21

rm_obj:
   rm -f *.o


loader21.o: loader21.c
   $(CC) -I$(INC) -DHOST_BYTE_ORDER=$(HOST_BYTE_ORDER) -g -c loader21.c

readelf.o: readelf.c
   $(CC) -I$(INC) -DHOST_BYTE_ORDER=$(HOST_BYTE_ORDER) -g -c readelf.c


comm2.o: comm2.c
   $(CC) -I$(INC) -DHOST_BYTE_ORDER=$(HOST_BYTE_ORDER) -g -c comm2.c

copy.o: copy.c
   $(CC) -I$(INC) -DHOST_BYTE_ORDER=$(HOST_BYTE_ORDER) -g -c copy.c


clean:
   rm -f *.o
   for I in $(BIN) ; do rm -f $(BINDIR)/$$I; done;
```

Anhang C

Benchmarks

Messung der Interrupt-Umschaltzeit

```
/*------------------------------------------------------------------------

                  Messung der Interrupt-Umschaltzeit
                  ==================================

Datei: init.c

Autor:              Thorsten Steckstor (VxWorks Version)
RTEMS Portierung:   Carsten Nitsch

------------------------------------------------------------------------*/


#define TEST_INIT

#include „system.h“
#include „RegalServer.h“

unsigned int Counter = 0;

struct slave {
  char adresse;
  char daten_in;
  char daten_out;
};

struct slave SlaveListe[MAX_SLAVE];

struct sensoraktor {
  unsigned char slave_listen_nummer;
```

```
  char bit_positions_maske;
};


static char MWHighByte;
static char MWLowByte;
static char MRHighByte;
static char MRLowByte;

static char NormalBetrieb;
static unsigned char AktuelleSlaveNummer;

struct sensoraktor Lichttaster;
struct sensoraktor LichtschrankeL;
struct sensoraktor LichtschrankeR;
struct sensoraktor XPosition[10];
struct sensoraktor ZPosition[10];
struct sensoraktor YPosition[3];

struct sensoraktor MotorXRechts;
struct sensoraktor MotorXLinks;
struct sensoraktor MotorXLangsam;
struct sensoraktor MotorYVorne;
struct sensoraktor MotorYHinten;
struct sensoraktor MotorZOben;
struct sensoraktor MotorZUnten;
struct sensoraktor MotorEALinksAus;
struct sensoraktor MotorEALinksEin;
struct sensoraktor MotorEARechtsAus;
struct sensoraktor MotorEARechtsEin;


/*--------------------------------------------------------------------------*/

/* rtems stuff */

rtems_name sensorSemName;
rtems_id sensorSem;

unsigned char sysInByte( unsigned char *addr)
{
  return *addr;
}


void sysOutByte( unsigned char *addr, unsigned char val)
{
 *addr = val;
}



/*--------------------------------------------------------------------------*/

/* The ISR */
rtems_isr fpga_isr(rtems_vector_number vector)
{
  unsigned char *led = (unsigned char*) 0xf2c00000;
  unsigned char intTest;
  static char sensoren;
  register unsigned int exisr;
  register unsigned int exier;
```

```
  *led = 0xff;
  intTest = sysInByte( (unsigned char*) FPGABaseAdr + IntStatusAdr);

  if ((intTest & ChannelBit) == 0) {
    MRLowByte  = sysInByte((unsigned char*)(FPGABaseAdr + ChannelNumber*2));
    MRHighByte = sysInByte((unsigned char*)(FPGABaseAdr + ChannelNumber*2+1));
    sysOutByte( (unsigned char*) (FPGABaseAdr + IntStatusAdr) , ChannelBit);
    sensoren = SlaveListe[AktuelleSlaveNummer].daten_in;
    SlaveListe[AktuelleSlaveNummer].daten_in = MRLowByte;
    rtems_semaphore_release(sensorSem);
    if (AktuelleSlaveNummer == (MAX_SLAVE-1))
      AktuelleSlaveNummer = 0;
    else
      AktuelleSlaveNummer++;
    MWLowByte = 0x80 | SlaveListe[AktuelleSlaveNummer].adresse;
    MWHighByte = SlaveListe[AktuelleSlaveNummer].daten_out;
    sysOutByte((unsigned char*) (FPGABaseAdr + ChannelNumber*2), MWLowByte);
    sysOutByte((unsigned char*) (FPGABaseAdr + ChannelNumber*2+1),MWHighByte);
  }

   /* clear external int pin 3 */
   exisr |= 0x00000002;
   asm volatile („mtexisr %0“ :: „r“ (exisr) );

  *led = 0x00; /* switch off led */
}


/*--------------------------------------------------------------------------*/

/* RTEMS Entry Routine - the ‚main‘ function */
rtems_task Init( rtems_task_argument ignored)
{
  rtems_isr_entry previous_isr;
  rtems_status_code status;
  unsigned int i_cache_reg, d_cache_reg, pvr;
  register unsigned int exisr, exier, iocr;
  unsigned int *led = (unsigned int *) 0xf2c00000;
  unsigned char c;

  *led = 0;
   printf(„\n“);
   printf(„******************* Interrupt Test  ***********************\n“);
   printf(„\n“);


   /* create a semaphore */

   sensorSemName = rtems_build_name( ‚S‘, ‚N‘, ‚1‘, , , );
   status = rtems_semaphore_create(sensorSemName, 0,
      RTEMS_FIFO |
      RTEMS_COUNTING_SEMAPHORE |
      RTEMS_NO_INHERIT_PRIORITY |
      RTEMS_NO_PRIORITY_CEILING |
      RTEMS_LOCAL,
      1, &sensorSem);

   /* read system registers */
   asm volatile („mficcr %0“       : „=r“ (i_cache_reg) );
   asm volatile („mfdccr %0“       : „=r“ (d_cache_reg) );
   asm volatile („mfexisr %0“      : „=r“ (exisr) );
   asm volatile („mfexier %0“      : „=r“ (exier) );
   asm volatile („mfiocr %0“       : „=r“ (iocr) );
```

```
 asm volatile („mfspr %0, 0x11f“ : „=r“ ((pvr        )));

 printf(„ICCR:                          %08x\n“, i_cache_reg);
 printf(„DCCR:                          %08x\n“, d_cache_reg);
 printf(„processor version:             %08x\n“, pvr);
 printf(„External Int. Enable Register: %08x\n“, exier);
 printf(„External Int. Status Register: %08x\n“, exisr);
 printf(„I/O Control Register:          %08x\n“, iocr);
 printf(„\n“);

/* Catch a externeal Interrupt */
status = rtems_interrupt_catch(fpga_isr,  PPC_IRQ_EXTERNAL , &previous_isr);
switch( status) {
   case RTEMS_SUCCESSFUL:      printf(„ISR installed successfuly\n“); break;
   case RTEMS_INVALID_NUMBER:  printf(„illegal vector number\n“);
   case RTEMS_INVALID_ADDRESS: printf(„illegal ISr entry point\n“);
   default: printf(„Unknown error\n“);
}

/* program the I/O Control Register                                  */
iocr = 0x3c000003;             /* external Intr. Pin 3:  low level */
                               /* external Intr. Pin 1:  pos. edge */
                               /* external Intr. Pin 2:  pos  edge */

asm volatile („mtiocr %0“ :: „r“ (iocr) );


/* program the Interrupt status Register             */
exisr = 0xffffffff;  /* clear all pending interrupts */
asm volatile („mtexisr %0“ :: „r“ (exisr) );


/* read exisr and exier and print it out */
asm volatile („mfexisr %0“      : „=r“ (exisr) );
asm volatile („mfexier %0“      : „=r“ (exier) );
printf(„External Int. Enable Register: %08x\n“, exier);
printf(„External Int. Status Register: %08x\n“, exisr);

c  = sysInByte( (unsigned char*) (FPGABaseAdr + DeviceIDAdr) );
printf(„DeviceID: 0x%02x\n“, c);

sysOutByte( (unsigned char*) (FPGABaseAdr + ResRegAdr), ChannelBit);
c = sysInByte( (unsigned char*) FPGABaseAdr + IntStatusAdr);
printf(„IntStatus: 0x%02x\n“, c);

sysOutByte( (unsigned char*) (FPGABaseAdr + ChannelNumber*2), 0);
sysOutByte( (unsigned char*) (FPGABaseAdr + ChannelNumber*2+1), 0);


/* register the ISR */
status = rtems_interrupt_catch(fpga_isr,  PPC_IRQ_EXTERNAL , &previous_isr);
switch( status) {
  case RTEMS_SUCCESSFUL:      printf(„ISR installed successfuly\n“); break;
  case RTEMS_INVALID_NUMBER:  printf(„illegal vector number\n“);
  case RTEMS_INVALID_ADDRESS: printf(„illegal ISr entry point\n“);
  default: printf(„Unknown error\n“);
}

exier = 0x02;                  /* enable external int pin 3 */
asm volatile („mtexier %0“ :: „r“ (exier) );
printf(„waiting\n“);

while(1)
```

```
  {
   rtems_semaphore_obtain(sensorSem,
                          RTEMS_DEFAULT_OPTIONS,
                          RTEMS_NO_TIMEOUT);

   asm volatile („mfexisr %0“      : „=r“ (exisr) );
   asm volatile („mfexier %0“      : „=r“ (exier) );
  }


}



/*--------------------------------------------------------------------------

                  Messung der Interrupt-Umschaltzeit
                  ==================================

Datei: RegalServer.h
Autor: Thorsten Steckstor


--------------------------------------------------------------------------*/



/* RegalServer.h - header used by RegalServer */

/* defines */

#define SERVER_PORT_NUM         4321   /* server‘s port number for bind()  */
#define SERVER_WORK_PRIORITY    198    /* priority of server‘s work task   */
#define ERROR_WORK_PRIORITY     199    /* priority of server‘s work task   */
#define SERVER_STACK_SIZE       2000   /* stack size of server‘s work task */
#define SERVER_MAX_CONNECTIONS  1      /* max clients connected at a time  */

#define REQUEST_MSG_SIZE        20     /* max size of request message      */

/* Y coordinates */
#define INOUT_STATION  0xFF
#define MITTE          0x00
#define HOCHREGAL      0x01

/* Error Codes */
#define SUCESSFUL        0x00
#define ZuwenigSlaves    0x40
#define KeinYTaster      0x41
#define ZweiYTaster      0x42
#define ZweiXTaster      0x43
#define ZweiZTaster      0x44

/* ASI Error Masken */
#define GeneralError     0x80
#define FatalError       0x40
#define ASIPowerFail     0x20
#define Manchester       0x10
#define Stopbit          0x01
#define Timeout          0x02
#define MSPause 0x04
#define Parity           0x08
```

```
/* Debug Codes */
#define DEBUG  0x00
#define TEST   0x00

/* LED Adressen */
#define LED_OFF_ADR    0xF2800000
#define LED_INT_ADR    0xF2C00000

/* Wartezeiten */
#define PICResetTime    15
#define SlaveResetTime  15
#define SteuerTime      55
#define ASIErrorTime    60

/* FPGA Adressen */
#define FPGA_INT_LEVEL     30
#define FPGABaseAdr        0xF2000000
#define Pipe1LowByteAdr    0x00000000
#define Pipe1HighByteAdr   0x00000001
#define Pipe2LowByteAdr    0x00000002
#define Pipe2HighByteAdr   0x00000003
#define Pipe3LowByteAdr    0x00000004
#define Pipe3HighByteAdr   0x00000005
#define Pipe4LowByteAdr    0x00000006
#define Pipe4HighByteAdr   0x00000007
#define ResRegAdr          0x00000008
#define LedReg1Adr         0x00000009
#define PSRegAdr           0x0000000A
#define LedReg2Adr         0x0000000B
#define IntStatusAdr       0x0000000C
#define LedReg3Adr         0x0000000D
#define DeviceIDAdr        0x0000000E
#define LedReg4Adr         0x0000000F

/* ASI Configuration */
/*
#define ChannelNumber      0x00000003
#define ChannelBit         0x08
*/

#define ChannelNumber      0x00000000
#define ChannelBit         0x01
#define MAX_SLAVE          7

/* ASI-Slave Configuration */
/*
#define ID_CODE0x00
#define IO_CODE0x07
*/

#define ID_CODE0x0f
#define IO_CODE0x07




/*-------------------------------------------------------------------------

                    Messung der Interrupt-Umschaltzeit
                    ==================================
```

Datei: *system.h*

```
----------------------------------------------------------------------*/



/*  system.h
 *
 *  This include file contains information that is included in every
 *  function in the test set.
 *
 *  COPYRIGHT (c) 1989, 1990, 1991, 1992, 1993, 1994.
 *  On-Line Applications Research Corporation (OAR).
 *  All rights assigned to U.S. Government, 1994.
 *
 *  This material may be reproduced by or for the U.S. Government pursuant
 *  to the copyright license under the clause at DFARS 252.227-7013.  This
 *  notice must appear in all copies of this file and its derivatives.
 *
 *  system.h,v 1.7 1996/05/29 22:47:39 joel Exp
 */

#include <tmacros.h>


/*

                            functions
                            =========
*/

rtems_task Init(
  rtems_task_argument argument
);

rtems_task Test_task(
  rtems_task_argument argument
);



/*----------------------------------------------------------------------

                      configuration information
                      =========================

*/

#define CONFIGURE_SPTEST

// #define CONFIGURE_INIT
   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
// #define CONFIGURE_TEST_NEEDS_CLOCK_DRIVER
// #define CONFIGURE_TEST_NEEDS_TIMER_DRIVER
   #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
   #include <confdefs.h>



/*----------------------------------------------------------------------

                       global variables
```

```
                        =================

  */

  TEST_EXTERN rtems_id   Task_id[ 4 ];         /* array of task ids */
  TEST_EXTERN rtems_name Task_name[ 4 ];       /* array of task names */
```

```
/*-------------------------------------------------------------------------

                  Messung der Interrupt-Umschaltzeit
                  ==================================

Datei: Makefile (fuer Power PC 403-GCX, beide Caches inaktiv)

-------------------------------------------------------------------------*/



HOMEDIR = /root
RTEMS_GNUTOOLS = /usr/local/powerpc-eabi
RTEMS_HOME = /usr/local/emb-tools/target/rtems/GCX/ND-NI/rtems/c/ppc
LIBRARIES =  /usr/local/emb-tools/target/libs

MY_LD = $(RTEMS_GNUTOOLS)/bin/ld
MY_AS = $(RTEMS_GNUTOOLS)/bin/as
MY_CC = $(RTEMS_GNUTOOLS)/bin/gcc

MY_STDLIB = $(LIBRARIES)/stdlib.a
MY_LIBGCC = $(LIBRARIES)/libgcc2.a
MY_LIBC   = $(LIBRARIES)/libc.a


MY_LDFLAGS = -T ./system/linkcmds  \
             -o Int.GCX-ND-NI.elf \
             -u atexit            \
             -u __vectors         \
             -u download_entry    \
         -u _wrapup_reent

MY_CFLAGS = -B$(RTEMS_GNUTOOLS)/bin/ \
            -mcpu=403 \
            -mbig \
            -Wall \
            -ansi \
            -fasm \
            -O4 \
            -g  \
            -fno-keep-inline-functions \
            -DPPC_ABI=PPC_ABI_EABI \
            -DPPC_ASM=PPC_ASM_ELF  \
            -DRTEMS_NEWLIB \
            -DMALLOC_PROVIDED \
            -DPPC_VECTOR_FILE_BASE=0x0100 \
            -I$(RTEMS_HOME)/include \
            -I/usr/local/powerpc-eabi/include \
        -I../include \
           -I./ \
           -msoft-float -mcpu=403 -mnew-mnemonics -mbig \
            -c

MY_CFLAGS_ASM = -B$(RTEMS_GNUTOOLS)/bin/ \
             -mcpu=403 \
             -mbig \
             -Wall \
             -ansi \
             -fasm \
             -O4 \
```

```
            -g  \
            -fno-keep-inline-functions \
            -DPPC_ABI=PPC_ABI_EABI \
            -DPPC_ASM=PPC_ASM_ELF  \
            -DRTEMS_NEWLIB \
            -DMALLOC_PROVIDED \
            -DPPC_VECTOR_FILE_BASE=0x0100 \
            -I$(RTEMS_HOME)/include \
            -I/tools/gcc-ppc/include \
         -I../include \
           -I./ \
            -msoft-float -mcpu=403 -mnew-mnemonics -mbig \
            -S

MY_CFLAGS2 = -B$(RTEMS_GNUTOOLS)/bin/ \
            -mcpu=403 \
            -mbig \
            -Wall \
            -ansi \
            -fasm \
            -O4 \
            -g  \
            -fno-keep-inline-functions \
            -DPPC_ABI=PPC_ABI_EABI \
            -DPPC_ASM=PPC_ASM_ELF  \
            -DRTEMS_NEWLIB \
            -DMALLOC_PROVIDED \
            -DPPC_VECTOR_FILE_BASE=0x0100 \
            -I$(RTEMS_HOME)/include \
            -I/tools/gcc-ppc/include \
         -I../include \
           -I ./ \
            -msoft-float -mcpu=403 -mnew-mnemonics -mbig \
            -c


MY_CPPFLAGS = -DRTEMS_NEWLIB \
              -DMALLOC_PROVIDED  \
              -DPPC_ABI=PPC_ABI_EABI \
              -DPPC_ASM=PPC_ASM_ELF \
              -DPPC_VECTOR_FILE_BASE=0x0100 \
              -I$(RTEMS_HOME)/include \
              -I/usr/include \
             -I../include \


MY_OBJS = ./init.o  \
          $(RTEMS_HOME)/lib/no-dpmem.rel  \
          $(RTEMS_HOME)/lib/debug.rel  \
          $(RTEMS_HOME)/lib/no-event.rel  \
          $(RTEMS_HOME)/lib/no-msg.rel    \
          $(RTEMS_HOME)/lib/no-mp.rel     \
          $(RTEMS_HOME)/lib/no-part.rel   \
          $(RTEMS_HOME)/lib/no-signal.rel \
          $(RTEMS_HOME)/lib/no-timer.rel  \
          $(RTEMS_HOME)/lib/no-rtmon.rel  \
          $(RTEMS_HOME)/lib/libbsp.a      \
          $(RTEMS_HOME)/lib/libcsupport.a \
          $(RTEMS_HOME)/lib/librtems.a    \
          $(RTEMS_HOME)/lib/libmisc.a     \
          $(MY_LIBC)   \
          $(MY_STDLIB) \
          $(MY_LIBC)   \
```

```
          $(MY_LIBGCC)


all:MkDir init.o
   $(MY_LD) $(MY_LDFLAGS) $(MY_OBJS)

MkDir:
   if [ ! -d o-ppc ] ; then mkdir o-ppc ; fi


init.o: init.c
   $(MY_CC) $(MY_CFLAGS) init.c -o init.o


##

clean:
   rm -f *.o o-ppc/*
```

Messung der Task-Umschaltzeiten

```
/*------------------------------------------------------------------------

                    Messung der Task-Umschaltzeiten
                    ===============================

 Dateien:
   system.h
   Makefile

siehe „Messung der Interrupt-Umschaltzeiten“
------------------------------------------------------------------------*/


/*------------------------------------------------------------------------

                    Messung der Task-Umschaltzeiten
                    ===============================

 Datei: init.c

 Autor:             Thorsten Steckstor (VxWorks Version)
 RTEMS Portierung:  Carsten Nitsch
------------------------------------------------------------------------*/

#define LOOP 1000000
#define TEST_INIT
#include „system.h“
#include <stdio.h>

rtems_task Task1 ();
rtems_task Task2 ();

/* Global data declaration */

rtems_status_code status;
char buffer[256];
rtems_name syncSem1Name;
rtems_name syncSem2Name;
rtems_id syncSem1;
rtems_id syncSem2;



rtems_task task1( rtems_task_argument ignored)
{
 int loopy1;
 unsigned char *LED1 = (unsigned char*) 0xf2800000;
 unsigned char *dummy = (unsigned char*) 0xf2000009;

 while(1) {
  rtems_semaphore_release(syncSem1);
  *LED1 = 0x00      ;                                /* LED1 off  */
  rtems_semaphore_obtain(syncSem2,
                         RTEMS_DEFAULT_OPTIONS,
                         RTEMS_NO_TIMEOUT
                        );
  *LED1 = 0xff;                                      /* LED1 on   */
```

```
  for(loopy1 = 0; loopy1 < LOOP; loopy1++)
    dummy = 0x00;                                    /* dummy I/O */
 }
}


rtems_task task2(rtems_task_argument argument)
{
  int loopy2;
  unsigned char *LED2 = (unsigned char*) 0xf2C00000;
  unsigned char *dummy = (unsigned char*) 0xf2000009;

  while(1) {
   *LED2 = 0x00;                                     /* LED2 off  */
   rtems_semaphore_obtain(syncSem1,
                          RTEMS_DEFAULT_OPTIONS,
                          RTEMS_NO_TIMEOUT
                         );
    *LED2 = 0xff;                                    /* LED2 on   */
   for(loopy2 = 0; loopy2 < LOOP; loopy2++)
    *dummy = 0x00;
                                                     /* dummy I/O */

   rtems_semaphore_release(syncSem2);
 }
}



/*--------------------------------------------------------------------------*/

/* RTEMS ‚main‘ function */
rtems_task Init(rtems_task_argument argument)
{

   unsigned int i_cache_reg, d_cache_reg, pvr;


  /* read system registers and print it out */
   asm volatile („mficcr %0“       : „=r“ (i_cache_reg) );
   asm volatile („mfdccr %0“       : „=r“ (d_cache_reg) );
   asm volatile („mfspr %0, 0x11f“ : „=r“ ((pvr        )));
   printf(„ICCR:              %08x\n“, i_cache_reg);
   printf(„DCCR:              %08x\n“, d_cache_reg);
   printf(„processor version: %08x\n“, pvr);
   printf(„\n“);


   printf(„\n*** Task Switch Test  ***\n\n“ );

   /* create task names */
   Task_name[ 1 ] = rtems_build_name( ‚T‘, ‚W‘, ‚1‘, , , );
   Task_name[ 2 ] = rtems_build_name( ‚T‘, ‚W‘, ‚2‘, , , );


   /* create semaphore names */
   syncSem1Name = rtems_build_name( ‚S‘, ‚N‘, ‚1‘, , , );
   syncSem2Name = rtems_build_name( ‚S‘, ‚N‘, ‚2‘, , , );


  /* Create 2 Tasks */
   status = rtems_task_create(
```

```
    Task_name[ 1 ], 1, RTEMS_MINIMUM_STACK_SIZE, RTEMS_PREEMPT,
    RTEMS_DEFAULT_ATTRIBUTES, &Task_id[ 1 ]
   );

    status = rtems_task_create(
    Task_name[ 2 ], 1, RTEMS_MINIMUM_STACK_SIZE, RTEMS_PREEMPT,
    RTEMS_DEFAULT_ATTRIBUTES, &Task_id[ 2 ]
   );

   /* Start the tasks */

   printf(„Start Tasks\n“);
   status = rtems_task_start( Task_id[ 1 ], task1, 1 );
   status = rtems_task_start( Task_id[ 2 ], task2, 2 );


   /* Create 2 Semaphores */

    status = rtems_semaphore_create(syncSem1Name, 0,
    RTEMS_FIFO |
    RTEMS_COUNTING_SEMAPHORE |
    RTEMS_NO_INHERIT_PRIORITY |
    RTEMS_NO_PRIORITY_CEILING |
    RTEMS_LOCAL,
    1, &syncSem1);

    status = rtems_semaphore_create(syncSem2Name, 0,
    RTEMS_FIFO |
    RTEMS_COUNTING_SEMAPHORE |
    RTEMS_NO_INHERIT_PRIORITY |
    RTEMS_NO_PRIORITY_CEILING |
    RTEMS_LOCAL,
    1, &syncSem2);


   status = rtems_task_delete( RTEMS_SELF );
}
```

Anhang D

Ladeprogramm für EEPROM

```
/*-------------------------------------------------------------------------

                          Programm eload
                         ================


 Datei: eload.c   mit main() Routine

-------------------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <time.h>
#include <string.h>
#include „eload.h“

#define DEBUG

// Prototypen fuer Funktionen

int wait(int);
void copy2EEPROM(unsigned int, unsigned char *, unsigned long, unsigned int);
void copy2LargeEEPROM(unsigned int, unsigned char *, unsigned long,
           unsigned int);
struct Parameter *GetArgs(int, char **);
FILE *OpenOrDie(char *, char *);
void WriteAddressByte(unsigned int, unsigned char);
void WriteDataByte(unsigned int, unsigned char);
struct FPGAData *rbt2bin(FILE *, unsigned long, unsigned long);
struct FPGAData *mcs2bin(FILE *, unsigned long, unsigned long);
unsigned long GetSize(struct FPGAData *);
```

```
int AskUser(char *);
void UploadRBT(FILE *, struct Parameter *);
void UploadMCS(FILE *, struct Parameter *);
unsigned long RBTDataSize(FILE * f);
unsigned long MCSDataSize(FILE * f);


/*********************** Serieller Bitstrom *****************************
 Makros fuer die Erzeugung eines seriellen Datagramms
 fuer die Programmierung des EEPROM

 Durch Schreibzugriffe auf die Adresse <BASE>:0x800 werden
 vom Mach die Signale auf der DATA und CLOCK Leitung jeweils
 auf eins bzw. 0 gesetzt:

 Datenbit 0 steuert DATA, Datenbit 1 beeinflusst CLOCK


 Timing:
                                  __________
     Ein Bit:  72..73 usec: ______|        |_______  Data ( einsbit )

                                     _____
                               _____|    |_________  clock

                                    | |    |  |

                                 18 usec   18 usec

                                     |    |
                                    36 usec


    Ein Byte besteht aus 9 bit ( 8 data, 1 ack), nanach folgen ca 110 usec
    pause.
    Ein Byte braucht also ca. 9*72 + 110  = 760 usec ( +/- 10 usec )

    Eine Page besteht aus 64 Daten Byte, einem Initbyte und 2 Adressbyte,
    braucht also fuer die Uebertragung ca 67 * 760 usec = 51 ms (+/- 1 ms),
    danach folgen hier 20 ms pause.
    eine page braucht also insgesamt ca 70 ms.

    DOS:
    ---
                                     _________
     Ein Bit:  72..73 usec:        _______|        |_______  Data ( einsbit )

                                       _____
                                  _____|    |_________  clock

                                       | |    |  |

                                  8.25 usec    8.25 usec

                                       |    |

                                      16.5 usec

    insgesamt also ca 33 usec

    nach jedem Byte ca. 93 usec Pause

*****************************************************************************/
```

```
#define STARTBIT(BASE)    pokeb( BASE, 0x800, 0x1); \
               wait(1);                  \
               pokeb( BASE, 0x800, 0x3); \
          wait(1);                   \
               pokeb( BASE, 0x800, 0x2); \
          wait(1);                   \
          pokeb( BASE, 0x800, 0x0); \
               wait(1)


#define STOPBIT(BASE)      pokeb( BASE, 0x800, 0x0); \
                wait(1);                  \
                pokeb( BASE, 0x800, 0x2); \
           wait(1);                   \
                pokeb( BASE, 0x800, 0x3); \
                wait(1);                  \
                pokeb( BASE, 0x800, 0x1); \
           wait(1)

#define EINSBIT(BASE)      pokeb( BASE, 0x800, 0x1); \
                wait(1);                  \
                pokeb( BASE, 0x800, 0x3); \
                wait(1); wait(1);         \
           pokeb( BASE, 0x800, 0x1); \
                wait(1)


#define NULLBIT(BASE)      pokeb( BASE, 0x800, 0x0); \
                wait(1);                  \
                pokeb( BASE, 0x800, 0x2); \
                wait(1); wait(1);         \
                pokeb( BASE, 0x800, 0x0); \
                wait(1)



/************************** Programmierung des Timers ******************
 *
 * Timer-Baustein 8253 : 1/18.21/65536 s = 1/1193180 s ( 0.83 usec )
 *
 * Ein Tick dauert ca. 0.83 usec
 *
 * wait wartet fuer 3*x Ticks. Unter DOS vergehen in dieser Zeit rund
 * 3..4 usec, unter Windows duch Overhead entpreched mehr Zeit.
 *
 * Somit sollte nach Setzen der Daten/Clocksignale innerhalb von Makros wie
 * EINSBIT oder STOPBIT ein wait(1) stets eine hinreichend lange Pause
 * bewirken, damit das EEPROM die Signale verarbeiten kann.
 * Zum Schreiben einer Page (64 Byte bzw 128 Byte) ist entsprechend laenger
 * zu warten.
 ***********************************************************************/

int wait(int x)
{

   unsigned int t0, t1;
   int dt;
   unsigned int foo;
   int xx = (int) 3 * x;

   foo = inportb(0x40);
```

```
   t0 = (inportb(0x40) << 8) + foo;
   do {
      foo = inportb(0x40);
      t0 = (inportb(0x40) << 8) + foo;
      dt = t0 - t1;
      if (dt < 0)
    dt = 65535 - (t1 - t0);
   }
   while (dt < xx);

   return 0;
}



/************************** Schreibroutinen ***************************
 *
 * Ein Byte wird als serieller Datenstrom (8 bit) uebertragen
 *
 **********************************************************************/

// Schreiben eines Byte, MSB first:

void WriteAddressByte(unsigned int base, unsigned char c)
{
   unsigned char i, m[] =
   {128, 64, 32, 16, 8, 4, 2, 1};

   for (i = 0; i < 8; i++) {
      if (c & m[i]) {
    EINSBIT(base);
      } else {
    NULLBIT(base);
      }
   }
   // Datenleitung nach Ende der Uebertragung wieder 0 setzen !
   NULLBIT(base);

   wait(16);
}


/* Schreiben eines Byte, LSB first */

void WriteDataByte(unsigned int base, unsigned char c)
{
   unsigned char i, m[] =
   {1, 2, 4, 8, 16, 32, 64, 128};

   for (i = 0; i < 8; i++) {
      if (c & m[i]) {
    EINSBIT(base);
      } else {
    NULLBIT(base);
      }
   }
   NULLBIT(base);
   wait(16);
}


/***************************** Oeffnen einer Datei *******************
 *
```

```
 * Eine Datei wird geoeffnet.
 * Im Fehlerfall wird das Programm nach einer entsprechenden Meldung
 * abgebrochen.
 *
 ************************************************************************/

FILE *OpenOrDie(char *name, char *mode)
{
   FILE *f;

   f = fopen(name, mode);
   if (f == NULL) {
      fprintf(stderr, „eload> Kann %s nicht oeffnen“, name);
      perror(„“);
      exit(-1);
   }
   return f;
}


// Bestimmen der Dateilaenge

unsigned long fsize(FILE * f)
{
   unsigned long i, i0;

   i0 = ftell(f);
   fseek(f, 0, SEEK_END);
   i = ftell(f);
   fseek(f, i0, SEEK_SET);
   return i;
}


/*************************** Beschreiben des EEPROM *******************
*
* Eine Page ( 64 Byte bei 17C68, 17C128, 17C256 bzw 128 Byte bei
* 17C512 und 17C010 ) wird zum EEPROM uebertragen.
* Danach ist eine Pause einzufuegen, in welcher der EEPROM diese Page
* durch einen internen Vorgang in seinen Speicher schreibt.
*
* Das Protokoll laeuft wie folgt ab:
*
* - Senden eines STARTBIT
* - Senden eines Devicebyte
* - Senden von 2 bzw. 3 Adressbyte
* - Senden von 64 bzw. 128 Datenbyte
* - Senden eines STOPBIT
*
*  Danach ist eine Pause einzufuegen, in welcher der EEPROM diese Page
*  durch einen internen Vorgang in seinen Speicher schreibt.
*
*  Weitere Details sind dem Programming Guide fuer die Konfiguartions-EEPROM
*  zu entnehmen (http://www.atmel.com)
*
************************************************************************/

// Routine fuer 17C64 .. 17C256
//
// page ist die Nummer der Uebertragenen Page und bestimmt unmittel-
// bar die Adresse im EEPROM, ab welcher diese page abgelegt wird
//
// chip ist fuer die Unterstuetzung der Kaskadierung in spaeteren
```

```
// Versionen reserviert und sollte hier mit Null belegt werden.
//

void copy2EEPROM(unsigned int base, unsigned char *buf, unsigned long page,
      unsigned int chip)
{
   unsigned char adr_lo, adr_hi;
   unsigned int i;
   unsigned long pagesize = 64;

   adr_lo = (page * pagesize) & 0x00ff;// Adressberechnung

   adr_hi = ((page * pagesize) & 0xff00) >> 8;

   STARTBIT(base);

   if (chip)
      WriteAddressByte(base, 0xae);// DeviceByte: 1 0 1 0  x 1 1 0
   else           //                      ^     ^
      WriteAddressByte(base, 0xa6);//                     [chip] [r/w]

   WriteAddressByte(base, adr_hi);// Adresse, hier 16 bit lang

   WriteAddressByte(base, adr_lo);

   for (i = 0; i < pagesize; i++)// 64 Byte uebertragen

      WriteDataByte(base, buf[i]);

   STOPBIT(base);
   delay(20);// Pause fuer Internen Schreibzyklus

}


// Routine fuer 17C512 und 17C010
//
// page ist die Nummer der Uebertragenen Page und bestimmt unmittel-
// bar die Adresse im EEPROM, ab welcher diese page abgelegt wird
//
// chip ist fuer die Unterstuetzung der Kaskadierung in spaeteren
// Versionen reserviert und sollte hier mit Null belegt werden.
//
// Der Ablauf ist analog zur Routine Copy2EEPROM, allerdings werden
// hier 3 Adressbyte uebertragen und die Pagesize ist 128.
//


void copy2LargeEEPROM(unsigned int base, unsigned char *buf,
           unsigned long page, unsigned int chip)
{
   unsigned char adr_0, adr_1, adr_2;
   unsigned int i;
   unsigned long adr;
   unsigned long pagesize = 128;

   adr = page * pagesize;

   adr_0 = (adr & 0x0000ff);
   adr_1 = (adr & 0x00ff00) >> 8;
   adr_2 = (adr & 0xff0000) >> 16;

   STARTBIT(base);
```

```
   if (chip)
      WriteAddressByte(base, 0xae);// 1 0 1 0  x 1 1 0
   else           //          ^     ^
      WriteAddressByte(base, 0xa6);//          [chip] [r/w]

   WriteAddressByte(base, adr_2);
   WriteAddressByte(base, adr_1);
   WriteAddressByte(base, adr_0);

   for (i = 0; i < pagesize; i++)
      WriteDataByte(base, buf[i]);

   STOPBIT(base);
   delay(20);

}

/***************************** Loeschen ****************************
 *
 * Das gesamte EEPROM wird mit ‚0‘ gefuellt
 *
 ******************************************************************/

void clear_eeprom(struct Parameter *parameter)
{
   unsigned long count;
   char buf[BLOCKSIZE];
   unsigned long fsize, page, pagesize, l, l0, type;

   memset(buf, 0, BLOCKSIZE);
   type = parameter->eeprom_type;
   fsize = type * 1024 / 8;

   if (parameter->eeprom_type < 512)
      pagesize = 64;
   else
      pagesize = 128;
   fprintf(stderr, „Loesche EEPROM\n“);

   for (count = 0, page = 0; count < fsize; count += pagesize, page++) {
      if (parameter->eeprom_type < 512)
    copy2EEPROM(parameter->base, buf, page, 0);
      else
    copy2LargeEEPROM(parameter->base, buf, page, 0);

      fprintf(stderr, „%3u %%>\r“,
         (unsigned int) ((page * pagesize * 100.0) / fsize));
      fflush(stderr);
   }
   fprintf(stderr, „100 %%>\n“);
}

/********************** Laderoutine fuer Binaerformat *****************
 *
 * Die Daten werden unveraendert aus einer Datei in da EEPROM geladen
 *
 ******************************************************************/

void UploadBIN(FILE * f, struct Parameter *parameter)
{
   unsigned long page, pagesize, Size;
   unsigned char buf[BLOCKSIZE];
```

```
   if (parameter->eeprom_type < 512)
      pagesize = 64;
   else
      pagesize = 128;
   Size = fsize(f);

   page = 0;
   while (!feof(f)) {
      fread(buf, 1, pagesize, f);
      if (parameter->eeprom_type < 512)
    copy2EEPROM(parameter->base, buf, page, 0);
      else
    copy2LargeEEPROM(parameter->base, buf, page, 0);
      page++;
      fprintf(stderr, „%3u %%>\r“,
         (unsigned int) ((page * pagesize * 100.0) / Size));
      fflush(stderr);
   }
   fprintf(stderr, „\n“);
}


/******************* Laderoutine fuer das MCS Format *****************
 *
 * Die Daten aus dem MCS File werden dekodiert und in das EEPROM geladen
 *
 ********************************************************************/


void UploadMCS(FILE * f, struct Parameter *parameter)
{
   unsigned long SourceOffset;
   unsigned long page, pagesize, Size;
   struct FPGAData *data;

   if (parameter->eeprom_type < 512)
      pagesize = 64;
   else
      pagesize = 128;
   Size = MCSDataSize(f);

   SourceOffset = 0;
   page = 0;
   while ((data = mcs2bin(f, SourceOffset, pagesize))) {
      if (parameter->eeprom_type < 512)
    copy2EEPROM(parameter->base, data->block, page, 0);
      else
    copy2LargeEEPROM(parameter->base, data->block, page, 0);

      SourceOffset = data->SourceOffset;
      free(data);
      page++;
      fprintf(stderr, „%3u %%>\r“,
         (unsigned int) ((page * pagesize * 100.0) / Size));
      fflush(stderr);

   }
   fprintf(stderr, „\n“);
}
```

```
/******************** Laderoutine fuer das RBT fileformat *************
 *
 * Die Daten aus dem RBT File werden dekodiert und in das EEPROM geladen
 *
 ***************************************************************************/

void UploadRBT(FILE * f, struct Parameter *parameter)
{
   unsigned long SourceOffset;
   unsigned long page, pagesize, Size;
   struct FPGAData *data;

   if (parameter->eeprom_type < 512)
      pagesize = 64;
   else
      pagesize = 128;
   Size = RBTDataSize(f);

   SourceOffset = 0;
   page = 0;
   while ((data = rbt2bin(f, SourceOffset, pagesize))) {
      if (parameter->eeprom_type < 512)
    copy2EEPROM(parameter->base, data->block, page, 0);
      else
    copy2LargeEEPROM(parameter->base, data->block, page, 0);

      SourceOffset = data->SourceOffset;
      free(data);
      page++;
      fprintf(stderr,
             „%3u %%>\r“, (unsigned int) ((page * pagesize * 100.0) / Size));
      fflush(stderr);

   }
   fprintf(stderr, „\n“);
}




/***************************** Hauptprogramm ******************************
*
*
***************************************************************************/

int main(int argc, char **argv)
{
   FILE *SourceFile;/* File, aus dem Daten ausgelesen werden  */
   FILE *LogFile;/* File, in das geschrieben wird (.log)   */
   struct Parameter *parameter;
   time_t t;
   char *date;
   struct FPGAData *Data;

   t = time(&t);
   date = ctime(&t);

   // Kommandozeilenparameter lesen
   parameter = (struct Parameter *) GetArgs(argc, argv);
   if (parameter == NULL) {
      fprintf(stderr, „Error bei Parameteruebergabe\n“);
```

```
   exit(-1);
}
if (strcmp(parameter->LogFileName, „“))
   LogFile = OpenOrDie(parameter->LogFileName, „ab“);
else
   LogFile = NULL;// kein Logfile

if (!parameter->clear)
   SourceFile = OpenOrDie(parameter->FileName, „rb“);

if (parameter->verbose) {
   fprintf(stderr, „%s\n“, VERSION);
   fprintf(stderr, „Filename:            %s\n“, parameter->FileName);
   fprintf(stderr, „Logfile:             %s\n“, parameter->LogFileName);
   fprintf(stderr, „Basisadresse:        %X\n“, parameter->base);
   fprintf(stderr,
           „EEPROM Type:         %d kbit\n“, parameter->eeprom_type);
   switch (parameter->fileformat) {
   case BIN:
fprintf(stderr, „Fileformat:          binaer\n“);
break;
   case RBT:
fprintf(stderr, „Fileformat:          text (rbt)\n“);
break;
   case MCS:
fprintf(stderr, „Fileformat:          text (mcs)\n“);
break;
   }
   fprintf(stderr, „\n“);
}
if (LogFile) {
   fprintf(LogFile, „\n“);
   fprintf(LogFile, „*************** <eload-log> ********************\n“);
   fprintf(LogFile, „\n“);
   fprintf(LogFile, „Datum:               %s\n“, date);
   fprintf(LogFile, „Filename:            %s\n“, parameter->FileName);
   fprintf(LogFile, „\n“);
}
// Vor Beginn der Programmierung sind folgende Signale zu setzen:
//
// PROG     = 0    .. FPGA abschalten, um Stoerungen zu vermeiden
// SER_ENA  = 1    .. ermoeglicht Schreiben des EEPROM
// SLAVEOUT = 1    .. Trennt FPGA vom EEPROM
//
// (siehe auch README.hardware) im Unterverzeichnis doc
//
// Das Setzen der Signale erfolgt wieder durch Schreibzugriffe
// auf eine Adresse im PC:
//
// Offset 0x806:          BIT 0:  PROG
//                        BIT 1:  SLAVEOUT
//                        BIT 2:  SER_ENABLE
//
//

pokeb(parameter->base, 0x806, 0x6);// prog=0, ser_ena=1, slaveout=1

delay(100);
pokeb(parameter->base, 0x806, 0x2);// prog=0, ser_ena=0, slaveout=1

delay(1200); // diese relativ lange Pause ist notwendig, ansonsten
             // wird das EEPROM unvollstaendig beschrieben.
             // (fehlen der ersten 2 .. 3 bloecke)
```

```
   STOPBIT(parameter->base);
   delay(100);

   // EEPROM loeschen bzw. programmieren
   if (parameter->clear)
      clear_eeprom(parameter);
   else
      switch (parameter->fileformat) {
      case RBT:
    UploadRBT(SourceFile, parameter);
    break;
      case BIN:
    UploadBIN(SourceFile, parameter);
    break;
      case MCS:
    UploadMCS(SourceFile, parameter);
    break;
      default:
    fprintf(stderr, „Zur Zeit werden nur RBT, \
                                MCS und BIN Formate unterstuetzt\n“);
    break;
      }

   delay(100);        // war 1000 in ver 0.96

   pokeb(parameter->base, 0x806, 0x4);// prog=0, slaveout=0, ser_ena=1

   delay(100);        // war 1000 in 0.96

   pokeb(parameter->base, 0x806, 0x5);// prog=1, slaveout=0, ser_ena=1

   delay(100);        // war 1000 in 0.96

   if (LogFile) {
      fprintf(LogFile, „\n“);
      fprintf(LogFile, „*************** </eload-log> ******************\n“);
   }
   if (SourceFile)
      fclose(SourceFile);
   if (LogFile)
      fclose(LogFile);

   return 0;
}

/*---------------------------------------------------------------------------

                          Programm eload
                         ================


 Datei: parse.c  (Auswertung der Kommandozeilenparameter)

---------------------------------------------------------------------------*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
```

```
#include „eload.h“

void help(char *);
void version(void);
int AskUser(char *);
char *GetExtension(char *);

void help(char *s)
{
   fprintf(stderr, „\n“);
   fprintf(stderr, „Syntax: eload [Optionen] <Dateiname>               \n“);
   fprintf(stderr, „          -V                   Version anzeigen    \n“);
   fprintf(stderr, „          -v                   ausfuehrliche Ausgabe\n“);
   fprintf(stderr, „          -x:<eeprom_typ>      Groesse des eeproms  \n“);
   fprintf(stderr, „          -d                   loesche eeprom       \n“);
   fprintf(stderr, „          -b:<base>            Basisadresse (hex)   \n“);
   fprintf(stderr, „          -l:<logfile>         Pfade des Logfiles   \n“);
   fprintf(stderr, „          -h , --help          diese Hilfe          \n“);
   fprintf(stderr,
      „            -t:<Dateiformat>     Format der zu ladenden Datei    \n“);
   fprintf(stderr, „                                                    \n“);
   fprintf(stderr, „ Zulaessige Argumente:                              \n“);
   fprintf(stderr, „   eeprom_typ:   64, 128, 256, 512, 1024 [kbit]     \n“);
   fprintf(stderr, „   Dateiformate: RBT, MCS, BIN                      \n“);
   fprintf(stderr, „                                                    \n“);
   fprintf(stderr, „ Defaultwerte:                                      \n“);
   fprintf(stderr,
       „   eeprom_typ: 1024     Basisadresse: D000    Logfile: keins    \n“);
   fprintf(stderr, „                                                    \n“);
   fprintf(stderr, „ Beispiel:                                          \n“);
   fprintf(stderr,
      „   eload -b:D800 -t:rbt -tMeinLogFile -x:256 Datei.xzy      \n“);
   fprintf(stderr,
      „    - Datei.xyz ist im RBT Format, geladen wird in ein 256 kbit\
           grosses\n“);
   fprintf(stderr,
      „      EEPROM, die Basisadresse ist D800, das Logfile heisst \
           \“MeinLogFile\“. \n“);
   exit(-1);
}

void version()
{
   printf(„%s\n“, VERSION);
   exit(0);
}

// Konvertierung in Integer

int GetVal(char *arg, int i, int b)
{
   char *s;
   int offset;

   /* Unter DOS werden Leerzeichen in der Kommandozeile zu 0x00 Byte (?) */

#ifdef DOS
#define SPACE 0
#endif

#ifdef UNIX
#define SPACE 0
#endif
```

```
   for (offset = 0; ((arg[2 + offset] == ,:‘) || (arg[2 + offset] == SPACE));
   offset++);

   s = arg + i + offset;
   if (s)
      return strtol(s, NULL, b);
   else
      return -1;
}

// Zeichenketten (z.B bei Filenamen)

char *GetStrVal(char *arg, int i)
{
   char *s, *r;
   int offset;

   /* Unter DOS werden Leerzeichen in der Kommandozeile zu 0x00 Byte (?) */
#ifdef DOS
#define SPACE 0
#endif

#ifdef UNIX
#define SPACE 0
#endif

   for (offset = 0; ((arg[2 + offset] == ,:‘) || (arg[2 + offset] == SPACE));
   offset++);
   s = arg + i + offset;

   r = (char *) malloc(strlen(s) + 1);
   strcpy(r, s);
   return r;

}


/* entspricht der UNIX routine ‚basename‘     */
/* Bsp: basename(„/data/src/foo.c“) ergibt „foo.c“   */

char *basename(char *path)
{
#ifdef DOS
   char d[] = „\\“;
#endif

#ifdef UNIX
   char d[] = „/“;
#endif

   char *s, *r;

   for (s = path; s;)
      if
     (strstr(s, d))
    s = strstr(s, d) + 1;
      else {
    r = (char *) malloc(strlen(s) + 1);
    strcpy(r, s);
    return r;
      }
```

```
   return NULL;
}


int FoundParameter(char *s, char *p)
{
   char *x;
   x = strstr(s, p);
   if (x == NULL)
      return 0;
   else if (x == s)
      return 1;
   else if (*(x - 1) == , ,)
      return 1;
   else
      return 0;
}

struct Parameter *GetArgs(int argc, char **argv)
{

   char FileName[256];
   char *ExtensionStr;
   int i;
   int base = -1;
   int eeprom_type = -1;
   int cascade = 0;
   int verbose = 0;
   int clear = 0;
   int Extension = BIN;
   char *LogFileName = NULL;
   char *FileType = NULL;
   struct Parameter *parameter;

   strcpy(FileName, „“);

   if (argc < 2) {
      help(argv[0]);
      exit(-1);
   } else {
      for (i = 1; i < argc; i++) {
    if ((FoundParameter(argv[i], „-h“)) ||
        (FoundParameter(argv[i], „--help“)))
       help(argv[0]);
    else if (FoundParameter(argv[i], „-V“))
       version();
    else if (FoundParameter(argv[i], „-b“))
       base = GetVal(argv[i], 2, 16);
    else if (FoundParameter(argv[i], „-x“))
       eeprom_type = GetVal(argv[i], 2, 10);
    else if (FoundParameter(argv[i], „-v“))
       verbose = 1;
    else if (FoundParameter(argv[i], „-c“))
       cascade = 1;
    else if (FoundParameter(argv[i], „-d“))
       clear = 1;
    else if (FoundParameter(argv[i], „-l“))
       LogFileName = GetStrVal(argv[i], 2);
    else if (FoundParameter(argv[i], „-t“))
       FileType = GetStrVal(argv[i], 2);
    else
       strcpy(FileName, argv[i]);
      }
```

```
    }

    if (base == -1)
       base = DEFAULT_BASE;
    if (eeprom_type == -1)
       eeprom_type = DEFAULT_EEPROM_TYPE;
    if ((!strcmp(FileName, „“)) && (clear == 0))
       help(argv[0]);
    if (LogFileName == NULL) {
       LogFileName = (char *) malloc(sizeof(DEFAULT_LOGFILENAME));
       strcpy(LogFileName, DEFAULT_LOGFILENAME);
    }
    // Der Typ der zu ladenden Datei wird wie folgt bestimmt:
    //  a) zuerst wird der Parameter -t ausgewertet.  -t:RBT erzwingt z.B.
    //     RBT Format
    //  b) wurde kein Parameter angegeben, wird versucht, den Typ aus der
    //     Extension zu ermitteln
    //  c) Schlaegt a) bzw. b) fehl, so wird der Nutzer interaktiv
    //     aufgefordert, den Typ anzugeben
    //

    if (!clear) {
       if (FileType) {
     for (i = 0; i < strlen(FileType); i++)
        FileType[i] = tolower(FileType[i]);
     if (strstr(FileType, „rbt“))
        Extension = RBT;
     else if (strstr(FileType, „mcs“))
        Extension = MCS;
     else if (strstr(FileType, „bin“))
        Extension = BIN;
     else
        Extension = AskUser(FileType);
       } else {

          // Extension des zu ladenden Files
     ExtensionStr = GetExtension(FileName);

     if (!strcmp(ExtensionStr, „.rbt“))
        Extension = RBT;
     else if (!strcmp(ExtensionStr, „.mcs“))
        Extension = MCS;
     else if (!strcmp(ExtensionStr, „.bin“))
        Extension = BIN;
     else
        Extension = AskUser(ExtensionStr);
       }
       if (Extension == UNKNOWN)
     exit(-1);
    }
    parameter = (struct Parameter *) malloc(sizeof(struct Parameter));

    strcpy(parameter->FileName, FileName);
    strcpy(parameter->LogFileName, LogFileName);
    parameter->verbose = verbose;
    parameter->base = base;
    parameter->eeprom_type = eeprom_type;
    parameter->cascade = cascade;
    parameter->clear = clear;
    parameter->fileformat = Extension;

    return parameter;
}
```

```
/*--------------------------------------------------------------------

                          Programm eload
                         ================


 Datei: convert.c  (Konvertierung)

--------------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include „eload.h“


// Diese Funktion ermittelt, wieviele Bytes aus einer RBT Datei
// decodiert werden koennen

unsigned long RBTDataSize(FILE * f)
{
   char zeile[2048];
   unsigned long i, count = 0;
   unsigned long PointerPosition;
   unsigned long Size;

   // Wenn das erste Zeichen einer Zeile im RBT File eine 1 oder 0 ist,
   // so wird sie als Datenzeile angenommen, ansonsten als Kommentar
   //
   // Wird innerhalb einer Zeile eine Zeichen entdeckt, welches keine
   // 1 oder 0 ist, so wird an dieser Stelle das Ende der Zeile angenommen

   PointerPosition = ftell(f);
   fseek(f, 0, SEEK_SET);

   while (fgets(zeile, 2048, f)) {
      if ((strstr(zeile, „1“) == zeile) || (strstr(zeile, „0“) == zeile))
    for (i = 0; zeile[i] == ‚0‘ || zeile[i] == ‚1‘; i++, count++);
   }

   fseek(f, PointerPosition, SEEK_SET);
   Size = count / 8;
   return Size;  // jede ‚1‘ bzw. ‚0‘ innerhalb einer Datenzeile ist ein Bit

}


// Diese Funktion errechnet, wieviele Byte aus einer MCS Datei
// dekodiert werden koennen

unsigned long MCSDataSize(FILE * f)
{
   unsigned char zeile[2048];
   unsigned long PointerPosition;
   unsigned long Count = 0;

   PointerPosition = ftell(f);
   fseek(f, 0, SEEK_SET);
   fgets(zeile, 2048, f);// erste zeile ignorieren

   // Ein Datenbyte wird durch 2 ASCII Zeichen hexadezimal dargestellt.
   // Jede Zeile beginnt mit 9 ASCII Zeichen fuer die Adresse und endet mit
```

```
   // 2 ASCII Zeichen fuer das Checksummenbyte sowie einem Zeilenumbruch
   // Die Anzahl der Datenbyte pro Zeile ergibt sich also also der Zeilen-
   // laenge abzueglich 12 Zeichen ( 9 Adress, 2 checksumme, 1 Zeilenumbruch),
   // dividiert durch 2

   while (fgets(zeile, 2048, f))
      if (strlen(zeile) > 12)
    Count += ((strlen(zeile) - 12) / 2);

   fseek(f, PointerPosition, SEEK_SET);
   //fprintf(stderr, „%u Byte werden geladen\n“, (unsigned int) Count);
   return Count;
}


// Konvertiert MCS Dateien ins Binaerformat

struct FPGAData *mcs2bin(FILE * f, unsigned long SourceOffset,
         unsigned long pagesize)
{
   unsigned long offset = 0;
   unsigned long i, start;
   unsigned char buf[BLOCKSIZE];
   unsigned char zeile[2048];
   struct FPGAData *data;
   unsigned char HexByteStr[4];
   char **error;

   memset(buf, 0, sizeof(buf));
   fseek(f, SourceOffset, SEEK_SET);// Ab dort weiterlesen, wo man bei
                                       // Aufruf vorher aufgehoert hat

   if (SourceOffset == 0)
      fgets(zeile, 2048, f);        // erste zeile ignorieren

   offset = 0;
   while ((offset < pagesize) && fgets(zeile, 2048, f)) {
      // Adressfeld ignorieren
      if (zeile[0] == ,:‘)
    start = 9;
      else
    start = 0;

      for (i = start; (offset < pagesize) && (i < strlen(zeile) - 4); i += 2)
      {
    // 2 Ascii-Zeichen stellen ein Byte als Headezimalzahl dar
    strcpy(HexByteStr, „“);
    strncat(HexByteStr, zeile + i, 2);
    buf[offset] = (unsigned char) strtol(HexByteStr, NULL, 16);
    offset++;
      }

   }


   // 64 bzw. 128 Byte in eine Struktur FPGAData schreiben.
   // Es werden immer ganze Bloecke zu 64/128 Byte benutzt.
   // Eventuell ist der letzte Block mit Nullen aufgefuellt.

   if (offset) {
      data = (struct FPGAData *) malloc(sizeof(struct FPGAData));
      if (data == NULL) {
    perror(„Fehler bei Speicheranforderung“);
```

```
     exit(-1);
        }
        memcpy(data->block, buf, pagesize);

        // FilePointer im Sourcefile
        data->SourceOffset = (ftell(f) - strlen(zeile)) + i;

        // ist eine zeile ausgelesen, soll beim folgenden aufruf von mcs2bin
        // mit einer neuen zeile begonnen werden.
        // Die verbliebenen Bytes werden also uebersprungen.
        if (i == (strlen(zeile) - 3))
     data->SourceOffset += 3;

        return data;
     } else
        return NULL;// Sourcefile ausgelesen


  }



  // Konvertiert RBT Dateien ins Binaerformat.
  // Siehe auch eload.h fuer weitere Hinweise

  struct FPGAData *rbt2bin(FILE * f, unsigned long SourceOffset,
           unsigned long pagesize)
  {
     char c, zeile[2048];
     char maske[] =
     {128, 64, 32, 16, 8, 4, 2, 1};
     unsigned long int i;
     unsigned char byte;
     unsigned long int bit_no;
     unsigned int offset = 0;
     unsigned char buf[BLOCKSIZE];
     struct FPGAData *data;

     byte = 0;
     bit_no = 0;
     memset(buf, 0, sizeof(buf));
     fseek(f, SourceOffset, SEEK_SET);// Ab dort weiterlesen, wo man bei
                                      // Aufruf vorher aufgehoert hat

     offset = 0;
     while ((offset < pagesize) && fgets(zeile, 2048, f)) {
        if ((strstr(zeile, „1“) == zeile) || (strstr(zeile, „0“) == zeile)) {
      for (i = 0; ((offset < pagesize) && ((zeile[i] == ‚0‘) ||
                    (zeile[i] == ‚1‘))); i++) {
         if (i > 2047) {
            fprintf(stderr,
               „ERROR! Zeile im RBT-File > 2048 Zeichen !!\n“);
            exit(-1);
         }
         if (zeile[i] == ‚1‘) {
            byte |= maske[7 - (bit_no % 8)];
            bit_no++;
         }
         if (zeile[i] == ‚0‘) {
            byte &= ~maske[7 - (bit_no % 8)];
            bit_no++;
         }
```

```
        if ((bit_no % 8) == 0) {
           buf[offset] = byte;
           offset++;
        }
     }
        }
   }

   // 64 bzw. 128 Byte in eine Struktur FPGAData schreiben.
   // Es werden immer ganze Bloecke zu 64/128 Byte benutzt.
   // Eventuell ist der letzte Block mit Nullen aufgefuellt.

   if (offset) {
      data = (struct FPGAData *) malloc(sizeof(struct FPGAData));
      if (data == NULL) {
     perror(„Fehler bei Speicheranforderung“);
     exit(-1);
      }
      memcpy(data->block, buf, pagesize);
      // FilePointer im Sourcefile
      data->SourceOffset = (ftell(f) - strlen(zeile)) + i;
      return data;
   } else
      return NULL;// Sourcefile ausgelesen

}

/*---------------------------------------------------------------------------

                            Programm eload
                           ================


 Datei: ext.c  (Dateityperkennung)

---------------------------------------------------------------------------*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include „eload.h“

int AskUser(char *e)
{
   unsigned char c;
   fprintf(stderr,
      „Das Dateiformat kann aus der Extension (‚%s‘) nicht \
            bestimmt werden.\n“, e);
   fprintf(stderr, „Bitte geben Sie das Dateiformat an:\n“);


   c = 0;
   while ((c != ‚b‘) && (c != ‚B‘) && (c != ‚m‘) && (c != ‚M‘) &&
     (c != ‚r‘) && (c != ‚R‘) && (c != ‚a‘) && (c != ‚A‘)) {
      if (c != 0x0a)
     fprintf(stderr, „[B]in / [R]BT / [M]CS / [A]bbruch  „);
      fflush(stderr);
      c = getc(stdin);
   }

   if ((c == ‚b‘) || (c == ‚B‘))
```

```
      return BIN;
   else if ((c == ‚m‘) || (c == ‚M‘))
      return MCS;
   else if ((c == ‚r‘) || (c == ‚R‘))
      return RBT;
   else
      return UNKNOWN;
}

// Liefert die Extension einer Dateinamen

char *GetExtension(char *dateiname)
{
   char *s0, *a, *s;
   char *extension;
   int i;

   a = dateiname;
   s0 = NULL;
   do {
      s = strstr(a, „.“);
      if (s) {
    s0 = s;
    a = s + 1;
      }
   }
   while (s);

   extension = malloc(strlen(s0) + 1);
   strcpy(extension, s0);
   for (i = 0; i < strlen(s); i++)
      extension[i] = _tolower(extension[i]);
   return extension;
}


/*---------------------------------------------------------------------------

                          Programm eload
                         ================


 Datei: eload.h

---------------------------------------------------------------------------*/



// #define DEBUG

// #define UNIX
#define DOS

#define VERSION „eload 1.0“
#define DEFAULT_BASE        0xd000
#define DEFAULT_EEPROM_TYPE 1024
#define DEFAULT_LOGFILENAME „“// default: Kein Logfile

// unterstuetzte Dateiformate

#define UNKNOWN 0//
#define BIN     1// Binaerdump
#define RBT     2// RBT Format
```

```
#define MCS     3// MCS Format


// Defines aus eload.c

#define SER_REG_ADR     0x800
#define SER_ENABLE_ADR  0x806
#define DATA_BIT        0
#define CLOCK_BIT       1
#define BASE_ADR        DEFAULT_BASE


struct Parameter {
   char FileName[256];
   char LogFileName[256];
   int eeprom_type;// 64, 128, 256, 512, 1024 [kbit]

   int base;// Basisadresse

   int verbose;
   int cascade;// fuer spaetere Versionen

   int clear;// Fuellen des EEPROM mit ‚0‘

   int fileformat;// BIN, RBT oder MCS-Format

};


// FPGA Konfigurations Daten.
// Die Funktion RBT2BIN liefert bei einem Aufruf jeweils einen Block
// von 64 bzw. 128 Byte zurueck.
// Die Funktion wird so oft hintereinander aufgerufen, bis die RBT/MCS
// Datei vollstaendig ausgelesen ist.
// In der Struktur FPGAData werden die Daten sowie der aktuelle Offset in
// die Sourcedatei (rbt/mcs) zurueckgeliefert.
//
// Bsp: Nach dem ersten aufruf stehen in buf 128 Byte ‚decodierte‘ Daten,
//      der offset in die Sourcedatei koennte z.B 128*8 Byte betragen
//      (da in RBT jedes Bit durch ein Character dargestellt wird.)
//      Durch Headerkommentare und Zeilenumbrueche kann der Offset
//      auch entsprechend groesser ausfallen.
//

#define BLOCKSIZE 128
struct FPGAData {
   // das ist die groesste pagesize fuer die eeproms
   char block[BLOCKSIZE];

   // der offset in die Quelldatei (rbt oder mcs)
   unsigned long SourceOffset;

   unsigned long Zeilennummer;// fuer mcs files

};


/*---------------------------------------------------------------------

                          Programm eload
                         ================
```

Datei: *make.bat* (Makescript fuer Borland C Compiler)

--*/

```
@echo off

bcc - c ext.c
bcc - c Convert.c
bcc - c eload.c
bcc - c parse.c
bcc - c rbt2bin.c
bcc - c mcs2bin.c

bcc eload.obj ext.obj Convert.obj parse.obj
bcc rbt2bin.obj Convert.obj
bcc mcs2bin.obj Convert.obj

copy eload.exe.. \ bin
del eload.exe

copy rbt2bin.exe.. \ bin
del rbt2bin.exe

copy mcs2bin.exe.. \ bin
del mcs2bin.exe

del eload.obj
del parse.obj
del ext.obj
del Convert.obj
del rbt2bin.obj
del mcs2bin.obj
```

Anhang E

Testprogramme

Grundtest 1

```
/*-------------------------------------------------------------

                          Grundtest 1
                          ===========

    Datei: rvect403.s  (resetvektor)

-------------------------------------------------------------*/


   .globlrvector
   .text

   .space0xFC
rvector:
        b       rvector



/*-------------------------------------------------------------------

   Programm Grundtest 1
   ====================

File: corep1.lnk (Linkfile)
-------------------------------------------------------------------*/

MEMORY
{
```

```
    reset: org = 0xffffff00, len = 0x100
}

SECTIONS
{
    .init (TEXT) : { rvect403.o(.text) } > reset
}
```

Grundtest 2

```
/*---------------------------------------------------------------------

                              Programm Grundtest 2
                              ====================

    Datei: rvect403.s  (resetvektor)

---------------------------------------------------------------------*/

   .globlrvector
   .text

   .space0xFC
rvector:
        b       _reset



/*---------------------------------------------------------------------

                              Programm Grundtest 2
                              ====================

    Datei: init.s

---------------------------------------------------------------------*/



   .globl_reset
   .globl main
   #include bcr.inc
   .text



msr_mon:  .equ0x00000000

#for GCX:
#INSTRUCTION_CACHE_LINES:     .equ  512  # GCX: 512 Lines, GA: 64 Lines
#DATA_CACHE_LINES:            .equ  256  # GCX: 256 Lines, GA: 32 Lines

# for GA:
#
INSTRUCTION_CACHE_LINES:     .equ  64
DATA_CACHE_LINES:            .equ  32


#############################################################################

_reset:



    # Invalidate Instruction Cache
    #Invalidate Line 0 .. 512  (for GCX, GA CPUs only have 64 Lines ! )
   li r1,0    # Start: Line 0
   li r2, INSTRUCTION_CACHE_LINES
```

```
    mtctrr2
     icloop:
    .long(31<<26)|(966<<1)|(1<<11)# iccci0, r1
    addir1,r1,0x10                # invalid inst cache line
    bdnzicloop                          # advance to next line


    # Invalidate Data Cache
    # invalidate Lines 0 .. 256 ( for GCX, GA CPUs have only 32 Lines ! )
    li r1,0           # Start: Line 0
    li r2, DATA_CACHE_LINES
    mtctrr2
     dcloop:
    .long(31<<26)|(454<<1)|(1<<11)       # dccci0,r1
    addir1,r1,0x10        # invalidate data cache line
    bdnzdcloop                          # advance to next line


     # Disable Caches

    lis     r1, 0                   # disable caches
    mtspr1018,r1# mtsprDCCR,r1 ???
    mtspr1019,r1# mtspr ICCR,r1 ???

  # Clear Pending IRQs & Debug status

         addisr1,r0,0xFFFF
    orir1,r1,0xFFFF

         # mtdcrEXISR,r1
    .long31<<26|451<<1|1<<21|(0x40&0x1F)<<16|(0x40&0x3E0)<<6

         mtspr0x3F0,r1                        # mtsprDBSR,r1


     # Disable all Interrupts
      lisr1,0x0000

      # mtdcrEXIER,r1
      .long31<<26|451<<1|1<<21|(0x42&0x1F)<<16|(0x42&0x3E0)<<6


     # Initialize the MSR register.

       addisr1,r0,msr_mon@h
       orir1, r1, msr_mon@l
       sync
       mtmsrr1

       isync


  ######################## Test Loop starts here ##########################

  # Scan EPROM Addressrange

  ForEver:
   lis r30, _reset@h
   ori r30, r30, _reset@l    # address range. (for 128 kByte EPROM :
                             #                 0xffffe_0000 .. 0xffff_ffff)
   lis r29, 0xffff
   ori r29, r29, 0xffff
   lis r28, 0x0000
```

```
  ori r28, r28, 0x0000
  scan_loop:
  lwzx r27, r28, r30
  cmpl 0,0,r30,r29         # r30 == 0xffff_ffff ?
  bgt ForEver
  addi r30, r30, 0x04      # r30 += 4
  ba scan_loop:
 ba ForEver


/*---------------------------------------------------------------------
```

Programm Grundtest 2
==========

Datei: *bcr.inc* (Banksettings)
Autor: Karlheinz Weiss

```
----------------------------------------------------------------------*/

; BAS_x           .equ     << 24      - Base Address Select  8 Bits
; BS_x            .equ     << 21      - Bank Size  3 Bits
; BU_x            .equ     << 19      - Bank Usage  2 Bits
; SLF_x           .equ     << 18      - Sequential Line Fill 1 Bit
; BME_x           .equ     << 17      - Burst Mode Enable  1 Bit
; BW_x            .equ     << 15      - Bus Width  2 Bits
; RE_x            .equ     << 14      - Ready Enable  1 Bit
; FWT_x           .equ     << 10      - Transfer Wait  6 Bits
; BWT_x           .equ     << 8       - First Wait 4 Bits
; TWT_x           .equ     << 8       - Burst Wait 2 Bits
; CSN_x           .equ     << 7       - Chip Select On Timing  1 Bit
; OEN_x           .equ     << 6       - Output Enable On Timing  1 Bit
; WBN_x           .equ     << 5       - Write Byte Enable On Timing  1 Bit
; WBF_x           .equ     << 4       - Write Byte Enable Off Timing  1 Bit
; TH_x            .equ     << 1       - Transfer Hold 3 Bits
; SD_x            .equ     << 0       - SRAM/DRAM Selection for BR4-BR7
;


; Spezifiziere D evice C ontrol R egister BR1
;
; Note: This Bank is used for memory mapped I/O devices attached
;       PowerPC Address lines A6..A31 are avaliabe at the FPGA

; Address-Range: 0xF000_0000 to 0xF3FF_FFFF Instruction and
;                Data Cache disabled (Region 30)
; Note: Access to I/O-Devices must be conduct with disabled caches !!!!
; Note: Bank 1 is used for 8 Bit I/O devices at base address 0xf000_0000
;
; Speicherbereich:
; 31 | 30 29 28 | 27 26 25 24 23 22 21 20 | Address Lines
;  0 |  1  2  3 |  4  5  6  7  8  9 10 11 | Address Lines PPC
;  x |  1  1  1 |  0  0  0  0  0  0  0  0 | Settings
;                         1  0  1  0

BAS_1           .equ     0x0 << 24      ; Base Address Select  8 Bits
BS_1            .equ     0x6 << 21      ; Bank Size  3 Bits 64 MByte
BU_1            .equ     0x3 << 19      ; Bank Usage  2 Bits (R/W)
SLF_1           .equ     0x0 << 18      ; Sequential Line Fill 1 Bit
BME_1           .equ     0x0 << 17      ; Burst Mode Enable  1 Bit
BW_1            .equ     0x0 << 15      ; Bus Width  2 Bits (8 Bit)
RE_1            .equ     0x1 << 14      ; Ready Enable  1 Bit
```

```
TWT_1           .equ     0xA << 8       ; NON Burst Wait 6 Bits
                                        ; (10 Transfer Waits
CSN_1           .equ     0x1 << 7       ; Chip Select On Timing  1 Bit
OEN_1           .equ     0x1 << 6       ; Output Enable On Timing  1 Bit
WBN_1           .equ     0x1 << 5       ; Write Byte Enable On Timing  1 Bit
WBF_1           .equ     0x1 << 4       ; Write Byte Enable Off Timing  1 Bit
TH_1            .equ     0x3 << 1       ; Transfer Hold 3 Bits
SD_1            .equ     0x1 << 0       ; SRAM/DRAM Selection for BR4-BR7

BR_1            .equ    BAS_1 | BS_1 | BU_1 | SLF_1 | BME_1 | BW_1 | RE_1 \
                      | TWT_1 | CSN_1 | OEN_1 | WBN_1 | WBF_1 | TH_1 | SD_1


; Spezifiziere D evice C ontrol R egister BR7: DRAM 32Bit
;
; Speicherbereich: DRAM Bank7 Simm-Module U1A ( 4 MB *32) 16 MB SIMM
; Address-Range: 0x0000_0000 to 0x00FF_FFFF Instruction and Data Cache
;                enabled  (Region  0)
; Address-Range: 0x8000_0000 to 0x80FF_FFFF Instruction and Data Cache
;                disabled (Region 16)
; Note: This Simm-Module is default
; 31 | 30 29 28 | 27 26 25 24 23 22 21 20 | Address Lines
;  0 |  1  2  3 |  4  5  6  7  8  9 10 11 | Address Lines PPC
;  x |  0  0  0 |  0  0  0  0  0  0  0  0 | Settings
;

BAS_7           .equ     0x0  << 24     ; BAS = A4-A11
BS_7            .equ     0x4  << 21     ; BS = 16 MByte
BU_7            .equ     0x3  << 19     ; BU = Read/Write
SLF_7           .equ     0x0  << 18     ; SLF = don‘t care
ERM_7           .equ     0x0  << 17     ; early RAS
BW_7            .equ     0x2  << 15     ; BW = 32 Bit
IM_7            .equ     0x0  << 14     ;
RCT_7           .equ     0x0  << 13     ;
ARM_7           .equ     0x0  << 12     ;
PM_7            .equ     0x1  << 11     ; mit burst
FAC_7           .equ     0x1  << 9      ; first wait 3
BAC_7           .equ     0x1  << 7      ; else 2-2-2
PCC_7           .equ     0x0  << 6      ; 2 1/2 Precharge
RAR_7           .equ     0x0  << 5      ; 2 1/2 RAS aktiv during refresh
RR_7            .equ     0x8  << 1      ; 512 Zyklen Refresh
SD_7            .equ     0x0  << 0      ; SD = DRAM

BR_7            .equ    BAS_7 | BS_7 | BU_7 | SLF_7 | ERM_7 | BW_7 | IM_7 \
                      | RCT_7 | ARM_7 | PM_7 | FAC_7 | BAC_7 | PCC_7 \
                      | RAR_7 | RR_7 | SD_7
```

```
/*-------------------------------------------------------------------------

                            Programm Grundtest2
                            ===================


Datei: corep1.lnk (Linkfile)
-------------------------------------------------------------------------*/

MEMORY
{
    rom:   org = 0xfffe0000, len = 0x1FF00
    reset: org = 0xffffff00, len = 0x100
}

SECTIONS
{
    .init (TEXT) : { rvect403.o(.text) } > reset
    .vtbl (TEXT) : { init.o(.text) }     > rom
}
```

Grundtest 3

```
/*---------------------------------------------------------------------

                            Programm Grundtest 3
                            ====================


  Dateien:
         rvect403.s  (resetvektor)
         corep1.lnk (Linkfile)
         bcr.h (Banksettings)

          siehe ‚Grundtest 2‘
---------------------------------------------------------------------*/



/*---------------------------------------------------------------------

                            Programm Grundtest 3
                            ====================

    Datei: init.s
---------------------------------------------------------------------*/


    .globl_reset
    .globl main
    .include bcr.inc   /* see ‚grundtest2‘ for details */
    .text



msr_mon:  .equ0x00000000
LED_GREEN:          .equ 0xf2800000
LED_RED:            .equ 0xf2c00000
LED_ON:             .equ 0xffffffff
LED_OFF:            .equ 0x00000000

#for GCX:
#INSTRUCTION_CACHE_LINES:     .equ  512  # GCX: 512 Lines, GA: 64 Lines
#DATA_CACHE_LINES:            .equ  256  # GCX: 256 Lines, GA: 32 Lines

# for GA:
#
INSTRUCTION_CACHE_LINES:     .equ  64
DATA_CACHE_LINES:            .equ  32


#############################################################################

_reset:

    # Initialize Br1: 8-Bit IO/DPRAM/FLASHES/FPGA/LED1 und
    #                 LED2/EXTPORT/16C550/ETHRNET MB86960

    lis r1, %hi(BR_1)
    ori r1, r1, %lo(BR_1)
```

```
    mtdcr br1, r1

    # Invalidate Instruction Cache
    #Invalidate Line 0 .. 512  (for GCX, GA CPUs only have 64 Lines ! )
   li r1,0    # Start: Line 0
   li r2, INSTRUCTION_CACHE_LINES
   mtctrr2
    icloop:
   .long(31<<26)|(966<<1)|(1<<11)# iccci0, r1
   addir1,r1,0x10               # invalid inst cache line
   bdnzicloop                         # advance to next line


   # Invalidate Data Cache
   # invalidate Lines 0 .. 256 ( for GCX, GA CPUs have only 32 Lines ! )
   li r1,0            # Start: Line 0
   li r2, DATA_CACHE_LINES
   mtctrr2
    dcloop:
   .long(31<<26)|(454<<1)|(1<<11)       # dccci0,r1
   addir1,r1,0x10        # invalidate data cache line
   bdnzdcloop                         # advance to next line


    # Disable Caches

   lis     r1, 0                   # disable caches
   mtspr1018,r1# mtsprDCCR,r1 ???
   mtspr1019,r1# mtspr ICCR,r1 ???

    # Clear Pending IRQs & Debug status

        addisr1,r0,0xFFFF
   orir1,r1,0xFFFF

        # mtdcrEXISR,r1
   .long31<<26|451<<1|1<<21|(0x40&0x1F)<<16|(0x40&0x3E0)<<6

        mtspr0x3F0,r1                        # mtsprDBSR,r1


    # Disable all Interrupts
     lisr1,0x0000

     # mtdcrEXIER,r1
     .long31<<26|451<<1|1<<21|(0x42&0x1F)<<16|(0x42&0x3E0)<<6


    # Initialize the MSR register.

      addisr1,r0,msr_mon@h
      orir1, r1, msr_mon@l
      sync
      mtmsrr1

      isync

######################## Test Loop starts here ##########################

forever:

  lis r2, LED_GREEN@h       # Switch on green LED and wait for ever
  ori r2, r2, LED_GREEN@l
```

```
lis r1, LED_ON@h
ori r1, r1, LED_ON@l
stwx r1, 0, r2
ba forever
```

DRAM Test

```
/*--------------------------------------------------------------------

                            Programm DRAM Test
                            ==================


    Dateien:
          rvect403.s  (resetvektor)
          corep1.lnk (Linkfile)
          bcr.h (Banksettings)

siehe ‚Grundtest 2‘
--------------------------------------------------------------------------*/



/*--------------------------------------------------------------------

                           Programm DRAM Test
                           ==================

    Datei: init.s
--------------------------------------------------------------------------*/


    .globl_reset
    .globl main
    .include bcr.inc   /* see ‚grundtest2‘ for details */
    .text



msr_mon:  .equ0x00000000
LED_GREEN:          .equ 0xf2800000
LED_RED:            .equ 0xf2c00000
LED_ON:             .equ 0xffffffff
LED_OFF:            .equ 0x00000000
SCANSTART:          .equ 0x00000000
SCANSIZE:           .equ 0x01000000
BLOCKSIZE:          .equ 256

#for GCX:
#INSTRUCTION_CACHE_LINES:     .equ  512  # GCX: 512 Lines, GA: 64 Lines
#DATA_CACHE_LINES:            .equ  256  # GCX: 256 Lines, GA: 32 Lines

# for GA:
#
INSTRUCTION_CACHE_LINES:     .equ  64
DATA_CACHE_LINES:            .equ  32


##############################################################################

_reset:

    # Initialize Br1: 8-Bit IO/DPRAM/FLASHES/FPGA/LED1 und
    #                 LED2/EXTPORT/16C550/ETHRNET MB86960
```

```
 lis r1, %hi(BR_1)
 ori r1, r1, %lo(BR_1)
 mtdcr br1, r1


# Initialize Br7: 32Bit DRAM SIMM0, 16 MB

 lis r1, %hi(BR_7)
 ori r1, r1, %lo(BR_7)
 mtdcr br7, r1


 # Invalidate Instruction Cache
 #Invalidate Line 0 .. 512  (for GCX, GA CPUs only have 64 Lines ! )
li r1,0    # Start: Line 0
li r2, INSTRUCTION_CACHE_LINES
mtctrr2
 icloop:
.long(31<<26)|(966<<1)|(1<<11)# iccci0, r1
addir1,r1,0x10                # invalid inst cache line
bdnzicloop                          # advance to next line


# Invalidate Data Cache
# invalidate Lines 0 .. 256 ( for GCX, GA CPUs have only 32 Lines ! )
li r1,0            # Start: Line 0
li r2, DATA_CACHE_LINES
mtctrr2
 dcloop:
.long(31<<26)|(454<<1)|(1<<11)       # dccci0,r1
addir1,r1,0x10        # invalidate data cache line
bdnzdcloop                          # advance to next line


 # Disable Caches

lis     r1, 0                   # disable caches
mtspr1018,r1# mtsprDCCR,r1 ???
mtspr1019,r1# mtspr ICCR,r1 ???

 # Clear Pending IRQs & Debug status

     addisr1,r0,0xFFFF
orir1,r1,0xFFFF

     # mtdcrEXISR,r1
.long31<<26|451<<1|1<<21|(0x40&0x1F)<<16|(0x40&0x3E0)<<6

     mtspr0x3F0,r1                        # mtsprDBSR,r1


 # Disable all Interrupts
  lisr1,0x0000

  # mtdcrEXIER,r1
  .long31<<26|451<<1|1<<21|(0x42&0x1F)<<16|(0x42&0x3E0)<<6


 # Initialize the MSR register.

   addisr1,r0,msr_mon@h
   orir1, r1, msr_mon@l
```

```
      sync
      mtmsrr1

      isync


 ######################## Memory Test starts here ########################
 #                                                                       #
 #########################################################################

 main:

 # switch off red and green LED
 #
  lis r1, LED_OFF@h
  ori r1, r1, LED_OFF@l
  lis r2, LED_RED@h
  ori r2, r2, LED_RED@l
  stwx r1, 0, r2
  lis r2, LED_GREEN@h
  ori r2, r2, LED_GREEN@l
  stwx r1, 0, r2


 # Used registers:
 #
 # r2  .. SCANSTART
 # r3  .. SCANSIZE
 # r4  .. BLOCKSIZE
 # r30 .. index (blockindex)
 # r29 .. index (blockoffset)
 # r28 .. temp
 # r27 .. temp
 # r26 .. temp

forever:



 # check the data bus:
 # write 0xaaaaaaaa and 0x55555555 into a register and [SCANSTART] and read
 # it back
 # If compare fails, the data bus needs your helping hand :)

 lis r2, SCANSTART@h
 ori r2, r2, SCANSTART@l
 lis r28, 0xaaaa
 ori r28, r28, 0xaaaa
 stwx r28, 0, r2            # write
 lwzx r27, 0, r2            # read it back
 cmpl 0, 0, r28, r27
 bne scan_ERROR             # sorry ..

 # same procedure with 0x55555555 pattern
 lis r28, 0x5555
 ori r28, r28, 0x5555
 stwx r28, 0, r2
 lwzx r27, 0, r2
 cmpl 0, 0, r28, r27
 bne scan_ERROR
```

```
 lis r2, SCANSTART@h
 ori r2, r2, SCANSTART@l
 lis r3, SCANSIZE@h
 ori r3, r3, SCANSIZE@l
 lis r4, BLOCKSIZE@h
 ori r4, r4,BLOCKSIZE@l


 lis r30, 0
 ori r30, r30, 0
 scan:


 lis r29, 0
 ori r29, r29, 0
 block_write:
   add r28, r29, r30
   cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE ??
   #beq 0, scan_end
   beq 0, block_write_end
   cmpl 0, 0, r29, r4        # offset = BLOCKINDEX ?
   beq 0, block_write_end
   add  r27, r2, r28      # r27 = SCANSTART + r28
   stwx r27, r2, r28         # [SCANSTART + r28 ] = SCANSTART + r28
   addi r29, r29, 4         # Blockoffset += 4
   b block_write
 block_write_end:



 lis r29, 0
 ori r29, r29, 0
 block_read:
   add r28, r29, r30
   cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE ??
   beq 0, scan_end
   cmpl 0, 0, r29, r4        # offset = BLOCKINDEX ?
   beq 0, block_read_end
   add r27, r2, r28          # r27 = SCANSTART + r28
   lwzx r26, r2, r28         # r26 = [SCANSTART+r28]
   cmpl 0, 0, r26, r27       # r26 == SCANSTART + r28 ?
   bne scan_ERROR
   addi r29, r29, 4
   b block_read
 block_read_end:

   add r30, r30, r4          # Blockno += BLOCKSIZE
   ba scan
scan_end:


# No Errors found: switch on Green LED

  lis r2, LED_GREEN@h       # Switch on green LED and wait for ever
  ori r2, r2, LED_GREEN@l
  lis r1, LED_ON@h
  ori r1, r1, LED_ON@l
  stwx r1, 0, r2
  ba forever
```

```
#############################################################

scan_ERROR:
 lis r2, LED_RED@h          # Switch on red LED and wait for ever
 ori r2, r2, LED_RED@l
 lis r1, LED_ON@h
 ori r1, r1, LED_ON@l
 stwx r1, 0, r2
 ba scan_ERROR

scan_OK:
  lis r2, LED_GREEN@h       # Switch on green LED and wait for ever
  ori r2, r2, LED_GREEN@l
  lis r1, LED_ON@h
  ori r1, r1, LED_ON@l
  stwx r1, 0, r2
  ba scan_OK
```

DRAM Test mit DPRAM-Interface

```
/*---------------------------------------------------------------------

                        DRAM Test mit DPRAM Interface
                        =============================


    Dateien:
     rvect403.s  (resetvektor)
     bcr.inc (Banksettings)
     corep1.lnk (Linkfile)

    siehe ‚Grundtest 2‘
---------------------------------------------------------------------*/


/*---------------------------------------------------------------------

                        DRAM Test mit DPRAM Interface
                        =============================


    Datei: init.s

---------------------------------------------------------------------*/


############################  Defines ############################

 LED_GREEN:          .equ 0xf2800000
 LED_RED:            .equ 0xf2c00000
 LED_ON:             .equ 0xffffffff
 LED_OFF:            .equ 0x00000000

 DPRAM:              .equ 0xf1000000
 SCANSTART_DPRAM:    .equ 0xf1000000
 SCANSIZE_DPRAM:     .equ 0x800   # 2kByte Dual Port Ram
 BLOCKSIZE_DPRAM:    .equ 256

 SCANSTART_DRAM:     .equ 0x00000000
 SCANSIZE_DRAM:      .equ 0x01000000       # 16 MByte Dram
 #SCANSIZE_DRAM:     .equ 0x800            # 2 k Dram Dram (for debugging)
 #SCANSIZE_DRAM:     .equ 512
 BLOCKSIZE_DRAM:     .equ 0x1000

 MESSAGESIZE:         .equ 12
 COUNTERSIZE:         .equ 4

 COUNT_DPRAM:     .equ DPRAM              # dpram address counter ( 4 byte)
 MESSAGE_DPRAM:   .equ COUNT_DPRAM    +  COUNTERSIZE
 COUNT_DRAM:      .equ MESSAGE_DPRAM  +  MESSAGESIZE
 MESSAGE_DRAM:    .equ COUNT_DRAM     +  COUNTERSIZE



 msr_mon:            .equ 0x00000000

 # for GCX:
 #INSTRUCTION_CACHE_LINES:     .equ  512 #512 for GCX, 64 for GA
 #DATA_CACHE_LINES:            .equ  256 #226 for GCX, 32 for GA
 # for GA:
```

```
 INSTRUCTION_CACHE_LINES:      .equ  64
 DATA_CACHE_LINES:             .equ  32


#########################################################################


_reset:


  #  BCR-Setting fuer SPYDER-CORE P1


  ; Br1: 8Bit IO/DPRAM/FLASHES/FPGA/LED1 und LED2/EXTPORT/16C550/ETHRNET

         lis r1, %hi(BR_1)
         ori r1, r1, %lo(BR_1)
         mtdcr br1, r1


  ; Br7: 32Bit DRAM SIMM0, 16 MB

         lis r1, %hi(BR_7)
         ori r1, r1, %lo(BR_7)
         mtdcr br7, r1


    # Invalidate Instruction Cache
    #Invalidate Line 0 .. 512  (for GCX, GA CPUs only have 64 Lines ! )
   li r1,0    # Start: Line 0
   li r2, INSTRUCTION_CACHE_LINES
   mtctrr2
    icloop:

   .long(31<<26)|(966<<1)|(1<<11)   # iccci0,r1
   addir1,r1,0x10            # invalid inst cache line
   bdnzicloop                      # advance to next line


    # Invalidate Data Cache
    # invalidate Lines 0 .. 256 ( for GCX, GA CPUs hvae only 32 Lines ! )

   li r1,0        # Start: Line 0
   li r2, DATA_CACHE_LINES
   mtctrr2
    dcloop:
   .long(31<<26)|(454<<1)|(1<<11)   # dccci0,r1
   addir1,r1,0x10    # invalidate data cache line
   bdnzdcloop                      # advance to next line

    # Disable Caches

   lis     r1, 0           ; disable caches
   mtspr1018,r1# mtsprDCCR,r1 ???
   mtspr1019,r1# mtspr ICCR,r1 ???


   # Clear Pending IRQs & Debug status

    addisr1,r0,0xFFFF
   orir1,r1,0xFFFF
```

```
      # mtdcrEXISR,r1
  .long31<<26|451<<1|1<<21|(0x40&0x1F)<<16|(0x40&0x3E0)<<6
  mtspr0x3F0,r1                              # mtsprDBSR,r1


  # Disable all Interrupts
     lisr1,0x0000
      # mtdcrEXIER,r1
  .long31<<26|451<<1|1<<21|(0x42&0x1F)<<16|(0x42&0x3E0)<<6


  # Initialize the MSR register.

  addisr1,r0,msr_mon@h
  orir1, r1, msr_mon@l
  sync
  mtmsrr1

  isync



# switch off red LED

lis r1, LED_OFF@h
ori r1, r1, LED_OFF@l
lis r2, LED_RED@h
ori r2, r2, LED_RED@l
stwx r1, 0, r2

#switch on green LED

lis r1, LED_ON@h
ori r1, r1, LED_ON@l
lis r2, LED_GREEN@h
ori r2, r2, LED_GREEN@l
stwx r1, 0, r2


# Start Test

##################### DualPortRAM Test ################################

# Used registers:
#
# r2  .. SCANSTART_DPRAM
# r3  .. SCANSIZE_DPRAM
# r4  .. BLOCKSIZE_DPRAM
# r5  .. DPRAM Counter
# r6  .. DPRAM Status Message
# r7  .. messagesize
# r8  .. messagetext
# r30 .. index (blockindex)
# r29 .. index (blockoffset)
# r28 .. temp
# r27 .. temp
# r26 .. temp


# check the data bus:
# write 0xaaaaaaaa and 0x55555555 into a register and [SCANSTART] and
# read it back
```

```
# If compare fails, the data bus needs your helping hand :)

lis r2, SCANSTART_DPRAM@h
ori r2, r2, SCANSTART_DPRAM@l
lis r28, 0xaaaa
ori r28, r28, 0xaaaa
stwx r28, 0, r2            # write
lwzx r27, 0, r2            # read it back
cmpl 0, 0, r28, r27
bne scan_ERROR_dpram       # sorry ..

# same procedure with 0x55555555 pattern
lis r28, 0x5555
ori r28, r28, 0x5555
stwx r28, 0, r2
lwzx r27, 0, r2
cmpl 0, 0, r28, r27
bne scan_ERROR_dpram


# Dual Port Ram Test

lis r2, SCANSTART_DPRAM@h
ori r2, r2, SCANSTART_DPRAM@l
lis r3, SCANSIZE_DPRAM@h
ori r3, r3, SCANSIZE_DPRAM@l
lis r4, BLOCKSIZE_DPRAM@h
ori r4, r4,BLOCKSIZE_DPRAM@l
lis r5, COUNT_DPRAM@h
ori r5, r5, COUNT_DPRAM@l

lis r30, 0
ori r30, r30, 0



scan_dpram:

lis r29, 0
ori r29, r29, 0
block_write_dpram:
  add r28, r29, r30
  cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE_DPRAM ??
  #beq 0, scan_end_dpram
  beq 0, block_write_end_dpram
  cmpl 0, 0, r29, r4        # offset = BLOCKINDEX_DPRAM ?
  beq 0, block_write_end_dpram
  add  r27, r2, r28      # r27 = SCANSTART_DPRAM + r28
  #stwx r27, r2, r28        # [SCANSTART_DPRAM + r28 ] =
                            #  SCANSTART_DPRAM + r28
  stwx r27, r2, r28
  addi r29, r29, 4     # Blockoffset_DPRAM += 4
b block_write_dpram
block_write_end_dpram:



lis r29, 0
ori r29, r29, 0
block_read_dpram:
  add r28, r29, r30
  cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE_DPRAM ??
  beq 0, scan_end_dpram
```

```
    cmpl 0, 0, r29, r4        # offset = BLOCKINDEX_DPRAM ?
    beq 0, block_read_end_dpram
    add r27, r2, r28          # r27 = SCANSTART_DPRAM + r28
    stw  r27, 0(r5)           # status counter (dpram user interface)
    lwzx r26, r2, r28         # r26 = [ SCANSTART_DPRAM + r28 ]
    cmpl 0, 0, r26, r27       # r26 == SCANSTART_DPRAM + r28 ?
    bne scan_ERROR_dpram
    addi r29, r29, 4
  b block_read_dpram
  block_read_end_dpram:

   add r30, r30, r4           # Blockno += BLOCKSIZE_DPRAM
   ba scan_dpram

 scan_end_dpram:


  # write a message to dual port ram

  lis r6, MESSAGE_DPRAM@h
  ori r6, r6, MESSAGE_DPRAM@l
  lis r7, MESSAGESIZE@h
  ori r7, r7, MESSAGESIZE@l
  lis r8, DPRAM_OK@h
  ori r8, r8, DPRAM_OK@l
  lis r28, 0
  ori r28, r28,0
  message_dpram_ok:
   cmpl 0, 0, r28, r7           # r28  == MESSAGESIZE ??
   beq 0, message_dpram_ok_end
   lbzx r29, r8, r28       # r29 = [DPRAM_OK + r28] ..messagetext
   stbx r29, r6, r28
   addi r28, r28, 1
   b message_dpram_ok
  message_dpram_ok_end:




  ######################### DRAM - TEST ##############################

  # Used registers:
  #
  # r2  .. SCANSTART_DRAM
  # r3  .. SCANSIZE_DRAM
  # r4  .. BLOCKSIZE_DRAM
  # r5  .. Addresse for dram counter (DualPortRAM User interface)
  # r30 .. index (blockindex)
  # r29 .. index (blockoffset)
  # r28 .. temp
  # r27 .. temp
  # r26 .. temp

  # check the data bus:
  # write 0xaaaaaaaa and 0x55555555 into a register and [SCANSTART] and
  # read it back
  # If compare fails, the data bus needs your helping hand :)

  forever:

  lis r2, SCANSTART_DRAM@h
  ori r2, r2, SCANSTART_DRAM@l
  lis r28, 0xaaaa
```

```
ori r28, r28, 0xaaaa
stwx r28, 0, r2            # write
lwzx r27, 0, r2            # read it back
cmpl 0, 0, r28, r27
bne scan_ERROR_dram        # sorry ..

# same procedure with 0x55555555 pattern
lis r28, 0x5555
ori r28, r28, 0x5555
stwx r28, 0, r2
lwzx r27, 0, r2
cmpl 0, 0, r28, r27
bne scan_ERROR_dram



lis r2, SCANSTART_DRAM@h
ori r2, r2, SCANSTART_DRAM@l
lis r3, SCANSIZE_DRAM@h
ori r3, r3, SCANSIZE_DRAM@l
lis r4, BLOCKSIZE_DRAM@h
ori r4, r4,BLOCKSIZE_DRAM@l
lis r5, COUNT_DRAM@h
ori r5, r5, COUNT_DRAM@l

lis r30, 0
ori r30, r30, 0
scan_dram:

lis r29, 0
ori r29, r29, 0
block_write_dram:
  add r28, r29, r30
  cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE ??
  # beq 0, scan_end_dram
  beq 0, block_write_end_dram
  cmpl 0, 0, r29, r4        # offset = BLOCKINDEX ?
  beq 0, block_write_end_dram
  add  r27, r2, r28      # r27 = SCANSTART + r28
  stwx r27, r2, r28         # [SCANSTART + r28 ] = SCANSTART + r28
  addi r29, r29, 4         # Blockoffset += 4
  b block_write_dram
block_write_end_dram:

lis r29, 0
ori r29, r29, 0
block_read_dram:
  add r28, r29, r30
  cmpl 0, 0, r28, r3        # r28 + 29 == SCANSIZE ??
  beq 0, scan_end_dram
  cmpl 0, 0, r29, r4       # offset = BLOCKINDEX ?
  beq 0, block_read_end_dram
  add r27, r2, r28          # r27 = SCANSTART + r28
  stw  r27, 0(r5)
  ;sth  r27, 2(r5)
  lwzx r26, r2, r28         # r26 = [SCANSTART+r28]
  cmpl 0, 0, r26, r27      # r26 == SCANSTART + r28 ?
  bne scan_ERROR_dram
  addi r29, r29, 4
  b block_read_dram
block_read_end_dram:

  add r30, r30, r4          # Blockno += BLOCKSIZE
```

```
   ba scan_dram
scan_end_dram:


# no errors found - write a message to dual port ram:

 lis r6, MESSAGE_DRAM@h
 ori r6, r6, MESSAGE_DRAM@l
 lis r7, MESSAGESIZE@h
 ori r7, r7, MESSAGESIZE@l
 lis r8, DRAM_OK@h
 ori r8, r8, DRAM_OK@l
 lis r28, 0
 ori r28, r28,0
 message_dram_ok:
  cmpl 0, 0, r28, r7                  # r28  == MESSAGESIZE ??
  beq 0, message_dram_ok_end
  lbzx r29, r8, r28       # r29 = [DRAM_OK + r28] .. messagetext
  stbx r29, r6, r28
  addi r28, r28, 1
  b message_dram_ok
 message_dram_ok_end:


# No Errors found: switch on Green LED
#
  lis r2, LED_GREEN@h       # Switch on green LED and wait for ever
  ori r2, r2, LED_GREEN@l
  lis r1, LED_ON@h
  ori r1, r1, LED_ON@l
  stwx r1, 0, r2

  ba forever




#################### subroutines and defines ###################################



scan_ERROR_dpram:

 lis r6, MESSAGE_DPRAM@h
 ori r6, r6, MESSAGE_DPRAM@l
 lis r7, MESSAGESIZE@h
 ori r7, r7, MESSAGESIZE@l
 lis r8, DPRAM_ERROR@h
 ori r8, r8, DPRAM_ERROR@l
 lis r28, 0
 ori r28, r28,0
 message_dpram_error:
  cmpl 0, 0, r28, r7           # r28  == MESSAGESIZE ??
  beq 0, message_dpram_error_end
  lbzx r29, r8, r28       # r29 = [DPRAM_ERROR + r28] .. messagetext
  stbx r29, r6, r28
  addi r28, r28, 1
  b message_dpram_error
 message_dpram_error_end:
```

```
loop_scan_error_dpram:

 lis r2, LED_GREEN@h          # Switch off green LED
 ori r2, r2, LED_GREEN@l
 lis r1, LED_OFF@h
 ori r1, r1, LED_OFF@l
 stwx r1, 0, r2

 lis r2, LED_RED@h          # Switch on red LED and wait for ever
 ori r2, r2, LED_RED@l
 lis r1, LED_ON@h
 ori r1, r1, LED_ON@l
 stwx r1, 0, r2
 ba loop_scan_error_dpram

#############################################################################


scan_ERROR_dram:

 lis r6, MESSAGE_DRAM@h
 ori r6, r6, MESSAGE_DRAM@l
 lis r7, MESSAGESIZE@h
 ori r7, r7, MESSAGESIZE@l
 lis r8, DRAM_ERROR@h
 ori r8, r8, DRAM_ERROR@l
 lis r28, 0
 ori r28, r28,0
 message_dram_error:
  cmpl 0, 0, r28, r7           # r28  == MESSAGESIZE ??
  beq 0, message_dram_error_end
  lbzx r29, r8, r28       # r29 = [DRAM_ERROR + r28] .. messagetext
  stbx r29, r6, r28
  addi r28, r28, 1
  b message_dram_error
 message_dram_error_end:

loop_scan_error_dram:

 lis r2, LED_GREEN@h          # Switch off green LED
 ori r2, r2, LED_GREEN@l
 lis r1, LED_OFF@h
 ori r1, r1, LED_OFF@l
 stwx r1, 0, r2

 lis r2, LED_RED@h          # Switch on red LED and wait for ever
 ori r2, r2, LED_RED@l
 lis r1, LED_ON@h
 ori r1, r1, LED_ON@l
 stwx r1, 0, r2
 ba loop_scan_error_dram



############################ Messagestrings ##############################

 DPRAM_OK:            .ascii „ DPRAM OK   „
 DPRAM_ERROR:         .ascii „ DPRAM ERROR“
 DRAM_OK:             .ascii „ DRAM OK    „
 DRAM_ERROR:          .ascii „ DRAM ERROR „

#############################################################################
```

Test des On Chip UARTs

```
/*---------------------------------------------------------------------

                        Test des On Chip Uarts
                        ======================


    Dateien:
     rvect403.s  (resetvektor)
     bcr.h (banksettings)

    Siehe ‚Grundtest 2‘
---------------------------------------------------------------------*/




/*---------------------------------------------------------------------

                        Test des On Chip Uarts
                        ======================


    Datei: init.s

---------------------------------------------------------------------*/


############################  Defines ##############################

 LED_GREEN:          .equ 0xf2800000
 LED_RED:            .equ 0xf2c00000
 LED_ON:             .equ 0xffffffff
 LED_OFF:            .equ 0x00000000

 msr_mon:            .equ 0x00000000

 # for GCX:
 #INSTRUCTION_CACHE_LINES:     .equ  512 #512 for GCX, 64 for GA
 #DATA_CACHE_LINES:            .equ  256 #226 for GCX, 32 for GA
 # for GA:
 INSTRUCTION_CACHE_LINES:      .equ  64
 DATA_CACHE_LINES:             .equ  32



#######################################################################


_reset:


  #  BCR-Setting fuer SPYDER-CORE P1


  ; Br1: 8Bit IO/DPRAM/FLASHES/FPGA/LED1 und LED2/EXTPORT/16C550/ETHRNET

         lis r1, %hi(BR_1)
```

```
       ori r1, r1, %lo(BR_1)
       mtdcr br1, r1


; Br7: 32Bit DRAM SIMM0, 16 MB

       lis r1, %hi(BR_7)
       ori r1, r1, %lo(BR_7)
       mtdcr br7, r1


  # Invalidate Instruction Cache
  #Invalidate Line 0 .. 512  (for GCX, GA CPUs only have 64 Lines ! )
li r1,0    # Start: Line 0
li r2, INSTRUCTION_CACHE_LINES
mtctrr2
  icloop:

.long(31<<26)|(966<<1)|(1<<11)   # iccci0,r1
addir1,r1,0x10            # invalid inst cache line
bdnzicloop                      # advance to next line


  # Invalidate Data Cache
  # invalidate Lines 0 .. 256 ( for GCX, GA CPUs hvae only 32 Lines ! )

li r1,0        # Start: Line 0
li r2, DATA_CACHE_LINES
mtctrr2
  dcloop:
.long(31<<26)|(454<<1)|(1<<11)   # dccci0,r1
addir1,r1,0x10    # invalidate data cache line
bdnzdcloop                      # advance to next line

  # Disable Caches

lis     r1, 0           ; disable caches
mtspr1018,r1# mtsprDCCR,r1 ???
mtspr1019,r1# mtspr ICCR,r1 ???


# Clear Pending IRQs & Debug status

      addisr1,r0,0xFFFF
orir1,r1,0xFFFF
      # mtdcrEXISR,r1
.long31<<26|451<<1|1<<21|(0x40&0x1F)<<16|(0x40&0x3E0)<<6
mtspr0x3F0,r1                             # mtsprDBSR,r1


# Disable all Interrupts
      lisr1,0x0000
      # mtdcrEXIER,r1
.long31<<26|451<<1|1<<21|(0x42&0x1F)<<16|(0x42&0x3E0)<<6


# Initialize the MSR register.

addisr1,r0,msr_mon@h
orir1, r1, msr_mon@l
sync
mtmsrr1
```

```
  isync



# switch off red LED

lis r1, LED_OFF@h
ori r1, r1, LED_OFF@l
lis r2, LED_RED@h
ori r2, r2, LED_RED@l
stwx r1, 0, r2

#switch on green LED

lis r1, LED_ON@h
ori r1, r1, LED_ON@l
lis r2, LED_GREEN@h
ori r2, r2, LED_GREEN@l
stwx r1, 0, r2

 # Initialize the IOCTR

 # Get the current IOCR register
 .long31<<26|323<<1|5<<21|(0xA0&0x1F)<<16|(0xA0&0x3E0)<<6 # mfdcr r5,IOCR

 # Sys Clock
 li r6, 0x0003
 andc r5, r5, r6
 ori  r5, r5, 0x3

 # External Clock, DTR, DSR
 .long31<<26|451<<1|5<<21|(0xA0&0x1F)<<16|(0xA0&0x3E0)<<6   # mtdcr IOCR,r5

 # Initialize Stack and Jump to __init_main()

 lisr1, __SP_INIT@h     # ADDR(ram) + SIZEOF(ram) + 4
 orir1, r1, __SP_INIT@l     # defined in corep1.lnk (Linkfile)

 lisr13, _SDA_BASE_@h
 orir13, r13, _SDA_BASE_@l

 lisr2, _SDA2_BASE_@h
 orir2, r2,_SDA2_BASE_@l

lir0, 0
stwur0, -64(r1)

ba __init_main



/*---------------------------------------------------------------------

                        Test des On Chip Uarts
                        ======================



 File: initc.c

---------------------------------------------------------------------*/
```

```
extern  char __DATA_ROM[], __DATA_RAM[], __DATA_END[], __BSS_START[];
extern  char  __BSS_END[];

/*

  defined in corep1.lnk (Linkfile)

 __DATA_ROM  =  ADDR(.sdata2) + SIZEOF(.sdata2)
 __DATA_RAM  =  ADDR(.data)
 __DATA_END  =  ADDR(.sdata) + SIZEOF(.sdata)
 __BSS_START =  ADDR(.sbss)
 __BSS_END   =  ADDR(.bss) + SIZEOF(.bss)

*/

unsigned long __init_main()
{

   /* move initialized data from ROM to RAM if defined at different locations
   */
   if (__DATA_ROM != __DATA_RAM) {
     memcpy(__DATA_RAM,__DATA_ROM,__DATA_END-__DATA_RAM);
   }

  /* clear uninitialized data */
  if (__BSS_START != __BSS_END) {
     memset(__BSS_START,0,__BSS_END-__BSS_START);
  }

 /* call the main function */
 return main();
}

/*-------------------------------------------------------------------------

                        Test des On Chip Uarts
                        ======================



 File: main.c

-------------------------------------------------------------------------*/

#define LED_GREEN 0xf2800000
#define LED_RED   0xf2c00000
#define LED_ON    0xffffffff
#define LED_OFF   0x00000000

#define UART    0x40000000
#define UART2               0xf1800000
#define CLOCKRATE    7372800
#define BAUDRATE            19200

// OnChipUart Registers
#define REG_LINE_STAT       UART + 0
#define REG_HANDSHAKE_STAT  UART + 2
#define REG_BAUDRATE_HI     UART + 4
#define REG_BAUDRATE_LO    UART + 5
#define REG_CTL             UART + 6
#define REG_RECEIVER    UART + 7
#define REG_TRANSMITTER    UART + 8
#define REG_BUFFER          UART + 9
```

```
// Program the built-in serial port to communicate at 19200 baud,
// configured for DTR/RTS, 8 data bits, 1 stop bit, no parity.


void HaltOnError()
{
  unsigned int *led = (unsigned int*) LED_RED;

  *led = 0xffffffff;
  while(1);
}


void InitOnChipUart()
{
  unsigned char *reg_baudrate_hi = (unsigned char *) REG_BAUDRATE_HI;
  unsigned char *reg_baudrate_lo = (unsigned char *) REG_BAUDRATE_LO;
  unsigned char *reg_transmitter = (unsigned char *) REG_TRANSMITTER;
  unsigned char *reg_receiver    = (unsigned char *) REG_RECEIVER;
  unsigned char *reg_line_stat   = (unsigned char *) REG_LINE_STAT;
  unsigned char *reg_ctl         = (unsigned char *) REG_CTL;
  unsigned int divider;


  // Don‘t forget to configure the IOCR !!
  // The Diab Compiler doesn‘t understand asm volatile directives,
  // so the IOCR Configuration must be done in init.s :--(

  //asm volatile („mfdcr %0, 0xa0“ : „=r“ (tmp));               // IOCR
  //tmp &= ~3;                       // clear bit 1 and 0
  //tmp |= 0;                      // use internal clock
  //asm volatile („mtdcr 0xa0, %0“ : „=r“ (tmp) : „0“ (tmp));   // IOCR

 *reg_baudrate_hi = 0; *reg_baudrate_lo = 0;    // clear the baudrate divisor
 *reg_transmitter = 0; *reg_receiver = 0;       // disable transmitter and
                                                // receiver
 *reg_transmitter = 0;                          // Disable transmitter
 *reg_receiver = 0;// Disable receiver

 *reg_line_stat = 0x78;                         // Linetstatus:
                                                // Set the following Status
                                                // bits to 0:
                                  //    Framing Error, Overrun Error, Parity
Error,
                                  //    LineBreak

 divider = CLOCKRATE / BAUDRATE;
 divider = ((divider + 8) >> 4) - 1;
 *reg_baudrate_lo = divider & 0xff;             // set baudrate divider
 *reg_baudrate_hi = divider >> 8;



 *reg_ctl = 0x38;// CTR/RTS, 8bit, 1stop,
                                                // no parity

 //*reg_ctl = 0x08;// no CTR/RTS, 8bit, 1stop,
                                                // no parity

 *reg_receiver = 0x88;                        // Enable Receiver
                                                // ( RTS , No IRQs )
```

```
 *reg_transmitter = 0x84;        // Enable Transmitter
                                                // ( CTS,  No IRQs )

 //*reg_receiver = 0x80;                // Enable Receiver
                                                // ( No RTS , No IRQs )

 //*reg_transmitter = 0x80;                    // Enable Transmitter
                                                // ( No CTS,  No IRQs )

}



// Send a character using OnChip UART
 void putchar( unsigned char c)
 {
   unsigned char *reg_handshake_stat = (unsigned char*) REG_HANDSHAKE_STAT;
   unsigned char *reg_buffer         = (unsigned char*) REG_BUFFER;
   unsigned char *reg_line_stat      = (unsigned char*) REG_LINE_STAT;
   unsigned int fnord;

   // reset DIS and CS bit
   fnord = 0xff;

   *reg_handshake_stat = 0xc0;

   while( fnord && *reg_handshake_stat) {
    fnord--;
    *reg_handshake_stat = 0xc0;
   }
   if (fnord==0) HaltOnError();  // timeout


   *reg_buffer = c;             // transmit a character

   //wait until transmitbuffer is empty and ready

   for (fnord=0xff; fnord  && *reg_handshake_stat &&
        ((*reg_line_stat & 0x04)==0) ; fnord--)
          *reg_handshake_stat = 0xc0;

   if (fnord==0) HaltOnError();

 }



 // Receive a character using OnChip UART
 unsigned char getchar()
 {
   unsigned char *reg_line_stat = (unsigned char*) REG_LINE_STAT;
   unsigned char *reg_buffer = (unsigned char*) REG_BUFFER;
   unsigned int *led_red = (unsigned int*) LED_RED;
   unsigned int fnord;


   // wait for a character
   for( fnord = 0xff; fnord && ( (*reg_line_stat & 0x80 ) == 0 ); fnord--);
   if (fnord==0) HaltOnError();  // timeout

   return *reg_buffer;
 }
```

```
// ---------------------------------------------------------------------



void echohex( unsigned short i)
{
 char s[4];
 char m[] = { ,0‘,‘1‘,‘2‘,‘3‘,‘4‘,‘5‘,‘6‘,‘7‘,‘8‘,‘9‘,
              ,a‘,‘b‘,‘c‘,‘d‘,‘e‘,‘f‘ };

 s[0] = m[((i & 0xf000) >> 12)];
 s[1] = m[((i & 0x0f00) >>  8)];
 s[2] = m[((i & 0x00f0) >>  4)];
 s[3] = m[((i & 0x000f) >>  0)];

 putchar( s[0] ); putchar( s[1]) ;
 putchar( s[2] ); putchar( s[3] );
}

void echo( char *s)
{
 int i;
 for(i=0; i<strlen(s); i++) putchar(s[i]);
}


// UART Test starts here.
// The OnChipUart will be initialized ( 19200 baud, 8 databit, 1 stopbit,
// no parity, no handshaking
// The application receives and writes back characters.
//
// On Error, switch on red Led an wait forever

main()
{

  unsigned long i,j,k;
  unsigned int *led_green = (unsigned int*) LED_GREEN;
  unsigned int *led_red = (unsigned int*) LED_RED;
  unsigned char out = 0xaa;
  unsigned char in;
  unsigned char *dummy = (unsigned char*) 0xf2000009;

  *led_red = LED_OFF;

  InitOnChipUart();

  out = 0x1;
  while (1)
  {
    putchar(out);
   in = getchar();
   if ( in != out) HaltOnError();
    out = (out+1) % 256;
  }



 }
```

```
/*-------------------------------------------------------------------

                        Test des On Chip Uarts
                        ======================



 File: corep1.lnk (Linkfile)

-------------------------------------------------------------------*/


MEMORY
{
    ram:   org = 0x00000000, len = 0x2000
    rom:   org = 0xFFFE0000, len = 0x1FF00
    reset: org = 0xFFFFFF00, len = 0x100
}

SECTIONS
{
    .init (TEXT) : { rvect403.o(.text) } > reset
    .vtbl (TEXT) : { init.o(.text) }  > rom

    GROUP : {
   .text (TEXT) : {
       *(.text) *(.init) *(.fini) *(.eini)
       . = (.+15) & ~15;
   }
   .sdata2 (TEXT) : {}
    } > rom

    GROUP :
    {
   .data  (DATA) LOAD(ADDR(.sdata2) + SIZEOF(.sdata2)) : {}
   .sdata (DATA) LOAD(ADDR(.sdata2) + SIZEOF(.sdata2) + SIZEOF(.data)):{}
   .sbss  (BSS) : { *(.sbss) }
   .bss   (BSS) : {}
    } > ram
}

__HEAP_START    = ADDR(.bss)+SIZEOF(.bss);
__SP_INIT       = ADDR(ram)+SIZEOF(ram)-4;  /* save 4 bytes  */
__HEAP_END      = __SP_INIT-0x800;
__SP_END        = __HEAP_END;
__DATA_ROM      = ADDR(.sdata2)+SIZEOF(.sdata2);
__DATA_RAM      = ADDR(.data);
__DATA_END      = ADDR(.sdata)+SIZEOF(.sdata);
__BSS_START     = ADDR(.sbss);
__BSS_END       = ADDR(.bss)+SIZEOF(.bss);

/* special equates for monitor memory/stack overflow checking */
memtest2 = __SP_END;
memtest3 = __SP_END+4;
memtest4 = __SP_INIT;

__TEXT_START= ADDR(.text);
__TEXT_END= ADDR(.text)+SIZEOF(.text);
```

Test der zweiten seriellen Schnittstelle

```
/*---------------------------------------------------------------------------

                    Test der zweiten seriellen Schnittstelle
                    ========================================


    Dateien: rvect403.s  (resetvektor)
             init.s
             initc.c
             corep1.lnk (Linkfile)

    siehe ‚Test des On Chip Uarts‘ und ‚Grundtest 2‘
---------------------------------------------------------------------------*/



/*---------------------------------------------------------------------------

                      Test der 2. seriellen Schnittstelle
                      ===================================

Datei: main.c
---------------------------------------------------------------------------*/

#define LED_GREEN 0xf2800000
#define LED_RED   0xf2c00000
#define LED_ON    0xffffffff
#define LED_OFF   0x00000000

#define UART2               0xf1800000
#define CLOCKRATE    7372800
#define BAUDRATE            19200

// Second UART Registers
#define RECEIVE_BUFFER      UART2 + 0
#define XMITT_BUFFER    UART2 + 0
#define DIVISOR_LO    UART2 + 0
#define DIVISOR_HI    UART2 + 1
#define INTERRUPT_ENABLE    UART2 + 1
#define FIFO_CONTROL    UART2 + 2
#define LINE_CONTROL    UART2 + 3
#define MODEM_CONTROL    UART2 + 4
#define LINE_STATUS         UART2 + 5

void HaltOnError()
{
  unsigned int *led = (unsigned int*) LED_RED;

  *led = 0xffffffff;
  while(1);
}



/*************************** 2 nd Uart ***********************************/

// initialize the second UART 16550D
// Use 8-1-N polled Xmitter/Receiver Mode

void InitUart2()
```

```
{
  unsigned char *line_control       = (unsigned char*) LINE_CONTROL;
  unsigned char *fifo_control       = (unsigned char*) FIFO_CONTROL;
  unsigned char *interrupt_enable   = (unsigned char*) INTERRUPT_ENABLE;
  unsigned char *modem_control      = (unsigned char*) MODEM_CONTROL;
  unsigned char *divisor_lo         = (unsigned char*) DIVISOR_LO;
  unsigned char *divisor_hi         = (unsigned char*) DIVISOR_HI;
  unsigned short divisor;


  *fifo_control  =  0x00;           // bit 0 = 0 .. disable read and write
                                    //              fifo
                                    // bit 1..7  .. not used, because fifos
                                    //              are disabled

  *interrupt_enable =  0x00;        // bit 0 = 0 .. Receive Data Available
                                    //              Intr. disabled
                                    // bit 1 = 0 .. Transmit Holding Reg
                                    //              empty, Intr. disabled
                                // bit 2 = 0 .. Receiver Line Status Interrupt
disabled

  *modem_control =  0x00;           // bit 0 = 0    DTR_ output is forced to
                                    //              a logic 0
                                    // bit 1 = 0    RTS_ output is forced to
                                    //              a logic 0
                                // bit 2 = 0    OUT1_ output is forced to a logic 1
                                // bit 3 = 0    OUT2_ output is forced to a logic 1
                                    // bit 4 = 0    Loopback disabled

  *line_control =  0x83;            //  bit 0,1 = 11 .. 8 Bit
                                    //  bit 2   = 0  .. 1 Stopbit
              //  bit 3   = 0  .. no parit
                                    //  bit 7   = 1  .. access divisor latch
                                    //                  during next r/w
                                    //                  operation
                                // //                 next read/write operation

   divisor = CLOCKRATE / ( BAUDRATE << 4 );
  *divisor_lo = (unsigned char)  divisor & 0x00ff;
  *divisor_hi = (unsigned char) (divisor & 0xff00) >> 8;

  // reset bit7 of the linecontrol reg to access Xmitt / Receive registers
    *line_control &= 0x7f;
 }


// Send a character using the second UART

void putchar2(unsigned char c)
{

  unsigned char *line_status = (unsigned char*) LINE_STATUS;
  unsigned char *xmitt_buffer = (unsigned char*) XMITT_BUFFER;
  unsigned char *line_control = (unsigned char*) LINE_CONTROL;
  unsigned int *led_red = (unsigned int*) LED_RED;
  unsigned int fnord;

  // wait until the UART is ready to accept a new character for transmission
  // ( wait until bit5 and bit 6 of the linestatus reg = 1 )

  for( fnord=0xff; fnord && ((*line_status & 0x60) == 0); fnord--);
  if (fnord==0) HaltOnError();  // timeout
```

```
  *xmitt_buffer = c; // send the character
 }

unsigned char getchar2()
{
  unsigned char *line_status = (unsigned char*) LINE_STATUS;
  unsigned char *receive_buffer = (unsigned char*) RECEIVE_BUFFER;
  unsigned int *led_red = (unsigned int*) LED_RED;
  unsigned int fnord;

  // wait until a character has been received
  for( fnord=0xff; fnord &&  (( *line_status & 0x01 )== 0); fnord --);
  if (fnord==0) HaltOnError();  // timeout

  return *receive_buffer;

}



/*********************************************************************/

void echohex( unsigned short i)
{
 char s[4];
 char m[] = { ,0‘,‘1‘,‘2‘,‘3‘,‘4‘,‘5‘,‘6‘,‘7‘,‘8‘,‘9‘,
              ,a‘,‘b‘,‘c‘,‘d‘,‘e‘,‘f‘ };

 s[0] = m[((i & 0xf000) >> 12)];
 s[1] = m[((i & 0x0f00) >>  8)];
 s[2] = m[((i & 0x00f0) >>  4)];
 s[3] = m[((i & 0x000f) >>  0)];

 putchar2( s[0] ); putchar2( s[1]) ;
 putchar2( s[2] ); putchar2( s[3] );
}

void echo( char *s)
{
 int i;
 for(i=0; i<strlen(s); i++) putchar2(s[i]);
}


// UART Test starts here.
// The  Uart will be initialized ( 19200 baud, 8 databit, 1 stopbit,
// no parity, no
// handshaking
// The application receives and writes back characters.
//
// On Error, switch on red Led an wait forever

main()
{

  unsigned long i,j,k;
  unsigned int *led_green = (unsigned int*) LED_GREEN;
  unsigned int *led_red = (unsigned int*) LED_RED;
  unsigned char out = 0xaa;
  unsigned char in;
  unsigned char *dummy = (unsigned char*) 0xf2000009;

  *led_red = LED_OFF;
```

```
 InitUart2();

 out = 0x1;
 while (1)
 {
   putchar2(out);
  in = getchar2();
  if ( in != out) HaltOnError();
    out = (out+1) % 256;
 }


}
```

Test der Ethernetschnittstelle

```
/*------------------------------------------------------------------------

                    Test der Ethernetschnittstelle
                    ==============================


    Dateien: rvect403.s  (resetvektor)
             init.s
             initc.c
             corep1.lnk (Linkfile)

    siehe ‚Test des On Chip Uarts‘ und ‚Grundtest 2‘
------------------------------------------------------------------------*/



/*------------------------------------------------------------------------

                     Test der Ethernetschnittstelle
                     ==============================

Datei: main.c

------------------------------------------------------------------------*/

// Test for the SpyderCore P1 ethernet interface  ( Fujitsu MB86960 )


#define ETHERNET_BASE       0xf3800000

// 2 k buffer
#define PACKETSIZE_LO       0xff
#define PACKETSIZE_HI       0x07   // this seems to be the maximum

#define ETHERNET_XSTAT      ETHERNET_BASE + 0x00  /* transmit status    */
#define ETHERNET_RSTAT      ETHERNET_BASE + 0x01  /* receive status     */
#define ETHERNET_XINT       ETHERNET_BASE + 0x02  /* transmit interrupt */
#define ETHERNET_RINT       ETHERNET_BASE + 0x03  /* reveive interrupt  */
#define ETHERNET_XMODE      ETHERNET_BASE + 0x04  /* transmit mode      */
#define ETHERNET_RMODE      ETHERNET_BASE + 0x05  /* receive mode       */
#define ETHERNET_CONF0      ETHERNET_BASE + 0x06  /* config reg. 0      */
#define ETHERNET_CONF1      ETHERNET_BASE + 0x07  /* config reg. 1      */
#define ETHERNET_MAC        ETHERNET_BASE + 0x08  /* mac address        */
#define ETHERNET_BUF        ETHERNET_BASE + 0x08  /* buffer             */
#define ETHERNET_XSTART     ETHERNET_BASE + 0x0a  /* send packet        */
#define LED_RED             0xf2c00000
#define LED_GREEN           0xf2800000

void HaltOnError()
{
 unsigned int *led = (unsigned int*) LED_RED;

 *led = 0xffffffff;
 while(1);
}



// Initialize the MB86960 Ethernet Controller
```

```
void InitEthernet()
{
  unsigned char *xstat  = (unsigned char*) ETHERNET_XSTAT;
  unsigned char *rstat  = (unsigned char*) ETHERNET_RSTAT;
  unsigned char *xint   = (unsigned char*) ETHERNET_XINT;
  unsigned char *rint   = (unsigned char*) ETHERNET_RINT;
  unsigned char *mac    = (unsigned char*) ETHERNET_MAC;
  unsigned char *xmode  = (unsigned char*) ETHERNET_XMODE;
  unsigned char *rmode  = (unsigned char*) ETHERNET_RMODE;
  unsigned char *conf0  = (unsigned char*) ETHERNET_CONF0;
  unsigned char *conf1  = (unsigned char*) ETHERNET_CONF1;
  unsigned char *buf    = (unsigned char*) ETHERNET_BUF;
  unsigned char *xstart = (unsigned char*) ETHERNET_XSTART;
  unsigned char k, k0, k1;
  unsigned int i, bla;
  unsigned char foo;
  unsigned int packetsize;
  unsigned int *led = (unsigned int*) LED_RED;

  *xint    =  0x00;    /* disable transmit interrupts    */
  *rint    =  0x00;    /* disable receive interrupts     */
  *rmode   =  0x6f;    /* Accept all packets             */
  *conf0   =  0xfe;    /* Disable Transmitter / Receiver,
                          buswidth 8bit,
                          bufferwidth 8bit,
                          8 kByte TxBuf, 16kByte total bufsize,
                          sramsize 32 kbyte
                       */

   *conf1    = 0x78;   /* internal encoder/decoder aktive, monitorpins active,
                          enable power for all functions,
                          select dlcr0..7, bmr8-f,
                          eop is low active,
                          byte order: not used
                       */

   *xmode    = 0x04;   /* set pin 95 to low ,
                          loopback on,
                          normal network operation
                       */


   *conf0    =  0x7e;    /* Enable Transmitter / Receiver,
                            buswidth 8bit,
                            bufferwidth 8bit,
                            8 kByte TxBuf, 16kByte total bufsize,
                            sramsize 32 kbyte
                         */



}

// send a packet an receive it (loopback mode)
// The first 2 bytes specify the length of the packet ( little endian order).
// See the MB86960 data sheet (page 16) for details)

void EthernetTest()
{

  unsigned char *buf    = (unsigned char*) ETHERNET_BUF;
  unsigned char *xstat  = (unsigned char*) ETHERNET_XSTAT;
  unsigned char *rstat  = (unsigned char*) ETHERNET_RSTAT;
```

```
  unsigned char *xstart = (unsigned char*) ETHERNET_XSTART;
  unsigned int packetsize, i;
  unsigned char k, k0, k1;
  unsigned int *led = (unsigned int*) LED_GREEN;

  *buf = PACKETSIZE_LO;
  *buf = PACKETSIZE_HI;       // PacketLength
  packetsize = PACKETSIZE_HI * 256 + PACKETSIZE_LO;

  for(i=0; i < packetsize; i++)
     *buf = (i % 255);        // fill the tranmitter buffer
                         // with characters:
                                          // 0,1,2 .. fe, 0, 1, ..

  *xstart = 0x81;                // send the packet



    // use green led as a progress indicator (flashing led )

    *led = 0xffffffff;                     // switch on green led
    while( (*rstat ) == 0);                // wait for a packet
    *rstat = 0x80;                         // packet received:
                                     // clear the receive status bit

    *led = 0x0;                            // switch off green led
    k = *buf; k = *buf;                    // skip 4 header bytes
    k0 = *buf; k1 = *buf;   // see datasheet page 17 for
                                     // details

    // analyze the packet
    // the payload must be consists of characters: 0,1,2,..fd,fe,0,1,2,...

    for(i=0; i < packetsize; i++) {
      k = *buf;
      if (  k != i % 255 ) HaltOnError();          // halt on error
    }                                // (switch on red led an wait forever)
}


main()
{
 unsigned int *led = (unsigned int*) LED_RED;
 unsigned char k;
 unsigned char *xstart = (unsigned char*) ETHERNET_XSTART;

 InitEthernet();

 while (1)  EthernetTest();
}
```

Anhang F

Treiber

Treiber für Flash-RAM

```
/*----------------------------------------------------------------------

                      Treiber fuer FLASH-RAM
                     ========================

 Datei: flash.c

----------------------------------------------------------------------*/


/* #undef FLASH0_DEBUG */
#define FLASH0_DEBUG

/* Default Parameter */

#define FLASH0_BLOCKSIZE         512
#define FLASH0_BLOCKS_PER_TRACK  FLASH0_BLOCKS
#define FLASH0_HEADS             1
#define FLASH0_RETRY 128
#define FLASH0_OFFSET            0/* Startaddress of flash0     */


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <vxworks.h>
#include <iosLib.h>
#include <blkIo.h>
#include <dosfslib.h>
#include <dirent.h>
#include <DosFsLib.h>
```

```
#include <usrlib.h>
#include <time.h>

STATUS sysFlashMySet(char *, int, int);
STATUS sysFlashGet(char *, int, int);

void test_stat(char *path);
```

```
/* Driver Routines */

/*
   Read Routine

   Arguments:
   device      .. pointer to device descriptor
   startBlock  .. starting block to read
   numBlocks   .. number of blocks to read
   buf         .. pointer to buffer to receive data

 */

STATUS flashBlockRead(
                      BLK_DEV * device, int startBlock,
                      int numBlocks, char *buf
                     )
{

#ifdef FLASH0_DEBUG
   fprintf(stderr, „<R%d>\n“, startBlock);
#endif

   if (startBlock + numBlocks > device->bd_nBlocks)
      return FALSE;
   else
      return sysFlashGet(
           buf,
           numBlocks * device->bd_bytesPerBlk,
           startBlock * device->bd_bytesPerBlk
     );
}



/*
   Write Routine

   Arguments:   device      .. pointer to device descriptor
   startBlock  .. starting block to write
   numBlocks   .. number of blocks to write
   buf         .. pointer to buffer of data to write

 */


STATUS flashBlockWrite(BLK_DEV * device, int startBlock, int numBlocks,
                      char *buf)
{
#ifdef FLASH0_DEBUG
   fprintf(stderr, „<W%d>\n“, startBlock);
#endif

   if (startBlock + numBlocks > device->bd_nBlocks)
      return FALSE;
   else
      return sysFlashMySet(
             buf,
             numBlocks * device->bd_bytesPerBlk,
             startBlock * device->bd_bytesPerBlk
     );
}
```

```
/*
   I/O Control Routine

   Arguments:   device   .. pointer to device descriptor
   funcCode  .. ioctl() function code
   arg       .. function specific argument
 */


STATUS flashIOCTL(BLK_DEV * device, int funcCode, int arg)
{
   return OK;
}



/*
   device Reset Routine
   Arguments:  device     ..  pointer to the drivers device descriptor
                              structure
*/


STATUS flashReset(BLK_DEV * device)
{
   return OK;
}



/*
   Status Check Routine

   Arguments:  pdevice     ..  pointer to device descriptor
 */


STATUS flashStatusCheck(BLK_DEV * pdevice)
{
   return OK;
}

/*****************************************************************************/

BLK_DEV *flash0_DeviceCreate(int blocks, int mode)
{

   BLK_DEV *dev_flash0;

   dev_flash0 = (BLK_DEV *) malloc(sizeof(BLK_DEV));

   /* function to read blocks */
   dev_flash0->bd_blkRd = (FUNCPTR) flashBlockRead;

   /* function to write blocks */
   dev_flash0->bd_blkWrt = (FUNCPTR) flashBlockWrite;

   /* function to ioctl device */
   dev_flash0->bd_ioctl = (FUNCPTR) flashIOCTL;

   /* function to reset device */
```

```
   dev_flash0->bd_reset = (FUNCPTR) flashReset;

   /* function to check status */
   dev_flash0->bd_statusChk = (FUNCPTR) flashStatusCheck;

   /* removable medium flag */
   dev_flash0->bd_removable = FALSE;

   /* number of blocks on device */
   dev_flash0->bd_nBlocks = blocks;

   /* number of bytes per block */
   dev_flash0->bd_bytesPerBlk = FLASH0_BLOCKSIZE;

   /* number of blocks per track */
   dev_flash0->bd_blksPerTrack = blocks;

   /* number of heads */
   dev_flash0->bd_nHeads = FLASH0_HEADS;

   /* retry count for I/O errors */
   dev_flash0->bd_retry = FLASH0_RETRY;

   /* O_RDONLY |O_WRONLY| O_RDWR */
   dev_flash0->bd_mode = mode;

   /* dev ready status changed,
      initialize to TRUE to cause
      the disk to be mounted     */
   dev_flash0->bd_readyChanged = TRUE;

   return dev_flash0;
}


/*
   create a dos filesystem and mount it to path
   ============================================
 */


STATUS flash_mkdosfs(char *path, int blocks, int mode)
{
   BLK_DEV *dev_flash0;
   DOS_VOL_DESC *volume;

   dev_flash0 = flash0_DeviceCreate(blocks, mode);
   if (dev_flash0 == NULL) {
#ifdef FLASH0_DEBUG
      perror(„Can not create flash-device“);
#endif
      return ERROR;
   }
   /* use dosFsMkfs to format the disk */

   dosFsMkfsOptionsSet(DOS_OPT_LONGNAMES);
   volume = dosFsMkfs(path, dev_flash0);
   if (volume == NULL) {
#ifdef FLASH0_DEBUG
      perror(„dosFSMkfs“);
#endif
      return ERROR;
   }
```

```
#ifdef FLASH0_DEBUG
   fprintf(stderr, „DOS FS created\n“);
#endif

   return OK;
}


/*
   Mount flash device
   ==================

*/

STATUS flash_mount(char *path, int blocks, int mode)
{
   BLK_DEV *dev_flash0;
   DOS_VOL_DESC *volume;

   dev_flash0 = flash0_DeviceCreate(blocks, mode);
   if (dev_flash0 == NULL) {
#ifdef FLASH0_DEBUG
      perror(„Can not create flash-device“);
#endif
      return ERROR;
   }
   /* use dosFsDevInit to use a formated disk */
   /* Note: Arg3 must be NULL                  */
   volume = dosFsDevInit(path, dev_flash0, 0);
   if (volume == NULL) {
#ifdef FLASH0_DEBUG
      perror(„dosFSDevInit“);
#endif
      return ERROR;
   } else
#ifdef FLASH0_DEBUG
      fprintf(stderr, „volume mounted\n“);
#endif

   return OK;
}

STATUS mkdosfs()
{
   flash_mkdosfs(„/“, 12288, O_RDWR);
}

/*-------------------------------------------------------------------

                      Treiber fuer FLASH-RAM
                      ======================

 Datei: Makefile

-------------------------------------------------------------------*/

LIB_PATH = C:/tornado / target / lib

CC = ccppc
OPTIONS = -DCPU = PPC403 \
-DVXWORKS - DNOFORK \
-BC:\Tornado / host / x86 - win32 / lib / gcc - lib / \
-mcpu = 403 - g - mstrict - align - ansi - nostdinc - O2 \
```

```
-fvolatile - fno - builtin - fno - for -scope
   \
       -Wall - I / h - I. \
 -IC: \Tornado \ target \ config \ all \
 -IC: \Tornado \ target / h \
 -IC: \Tornado \ target / src / config \
 -IC: \Tornado \ target / src / drv


 flash.o:flash.c
       $(CC) - c $(OPTIONS) flash.c

 flash.i:flash.c
       $(CC) $(OPTIONS) - E - <flash.c > flash.i

 install:flash.o
       arppc - r $(LIB_PATH) / libPPC403gnuvx.a flash.o

 clean:
   rm - f flash.o
       rm - f flash.i
```

Treiber für RAM Drive

```
/*-------------------------------------------------------------------------

                      Treiber fuer RAM Drive
                     ======================

Datei: ramdrive.c

-------------------------------------------------------------------------*/


#define RAMDRIVE0_RAMADDR          0/* where it is in memory (0..malloc) */
#define RAMDRIVE0_BLOCKSIZE        512
#define RAMDRIVE0_OFFSET           0/* Startaddress of RAMDRIVE0     */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <vxworks.h>
#include <iosLib.h>
#include <blkIo.h>
#include <dosfslib.h>
#include <dirent.h>
#include <DosFsLib.h>
#include <usrlib.h>
#include <time.h>
#include <ramdrv.h>


STATUS ramdrive_mkdosfs(char *path, int blocks)
{
   BLK_DEV *dev_ramdrive0;
   DOS_VOL_DESC *volume;

   dev_ramdrive0 = ramDevCreate(
            RAMDRIVE0_RAMADDR,/* where it is in memory (0..malloc) */
            RAMDRIVE0_BLOCKSIZE,/* number of bytes per block */
            blocks,/* number of blocks per track */
            blocks,/* number of blocks on device */
            RAMDRIVE0_OFFSET/* no. of blks to skip at start of device */
       );


   if (dev_ramdrive0 == NULL) {
#ifdef RAMDRIVE_DEBUG
      perror(„Can not create ramdisk device“);
#endif
      return ERROR;
   }
#ifdef RAMDRIVE_DEBUG
   else
      fprintf(stderr, „ramdrive device created. \n“);
#endif

   /* use dosFsMkfs to format the disk */

#ifdef RAMDRIVE_DEBUG
   fprintf(stderr, „formating disk\n“);
#endif
```

```
   dosFsMkfsOptionsSet(DOS_OPT_LONGNAMES);
   volume = dosFsMkfs(path, dev_ramdrive0);

   if (volume == NULL) {
#ifdef RAMDRIVE_DEBIG
      perror(„dosFSMkfs“);
#endif
      return ERROR;
   }
#ifdef RAMDRIVE_DEBUG
   else
      fprintf(stderr, „DOS FS created\n“);
#endif

   return OK;
}

/*-------------------------------------------------------------------------

                  Treiber fuer RAM Drive
                  ----------------------

Datei: Makefile

-------------------------------------------------------------------------*/


LIB_PATH = C:/tornado / target / lib

CC = ccppc
OPTIONS = -DCPU = PPC403 \
-DVXWORKS - DNOFORK \
-BC:\Tornado / host / x86 - win32 / lib / gcc - lib / \
-mcpu = 403 - g - mstrict - align - ansi - nostdinc - O2 \
-fvolatile - fno - builtin - fno - for -scope
   \
       -Wall - I / h - I. \
 -IC: \Tornado \ target \ config \ all \
 -IC: \Tornado \ target / h \
 -IC: \Tornado \ target / src / config \
 -IC: \Tornado \ target / src / drv


 ramdrive.o:ramdrive.c
       $(CC) - c $(OPTIONS) ramdrive.c


 install:ramdrive.o
       arppc - r $(LIB_PATH) / libPPC403gnuvx.a ramdrive.o

 clean:
   rm - f ramdrive.o
```

Treiber für Xilinx FPGA XC-4000

```
/*-----------------------------------------------------------------------*/

                    Treiber fuer  Xilinx XC4000 FPGA
                   ==================================

 Datei: xc4000.c

-----------------------------------------------------------------------*/


#include <stdlib.h>
#include <stdio.h>
#include <iosLib.h>
#include <string.h>
#include  <timers.h>
#include <vxworks.h>
#include <semLib.h>
#include <sysLib.h>
#include <errnoLib.h>
#include „xc4000.h“



/*************************************************************************/

/*
   Functions
   ---------
*/


void mysleep(int l)
{
   unsigned int i;
   unsigned char *dummy = (unsigned char *) 0xf2000009;

   for (i = 0; i < 10 * l; i++)
      *dummy = 0x00;/* I/O Access for delay */
}



int XC4000_Read(DEV_XC4000 * device, char *buffer, int n)
{
#ifdef XC4000_DEBUG
   fprintf(stderr, „XC4000_Read\n“);
#endif
   return OK;
}



/*
   XC4000_Write

   write a designfile into the fpga.

   returned values:    bytes written to fpga ( 0 if finished)
   -1 on error
```

```
 */

int XC4000_Write(DEV_XC4000 * device, char *buffer, int n)
{
   int i, size;
   char ccc;
   /*unsigned char *led1 = (unsigned char*) 0xf2800000; */

   int m[] =
   {1, 2, 4, 8, 16, 32, 64, 128};
   size = 0;
   i = 0;

   while ((!(sysInByte(device->xc4000_prog) & 0x40)) && (size < n)) {
      if (*buffer & m[i])
    *(device->xc4000_io) = 0x80;/* write a binary 1 */
      else
    *(device->xc4000_io) = 0x00;/* write a binary 0 */
      i++;
      if (i == 8) {
    i = 0;
    buffer++;
    size++;
      }
      /* write the last byte */
      if ((sysInByte(device->xc4000_prog) & 0x40)) {
    for (i = 0; i < 7; i++)
       if (*buffer & m[i])
          *(device->xc4000_io) = 0x80;
       else
          *(device->xc4000_io) = 0x00;
    size++;
      }
   }

   mysleep(1);
   /* *led1 = 0xff; */
   ccc = sysInByte((int) device->xc4000_prog);
   /* *led1 = 0x00; */

#ifdef XC4000_DEBUG
   fprintf(stderr, „XC4000_Write (%d byte) (prog=0x%02x)\n“, size, ccc);
#endif

   return size;
}



int XC4000_IOCTL(DEV_XC4000 * device, int functioncode, int arg)
{
   char *s;

#ifdef XC4000_DEBUG
   fprintf(stderr, „XC4000_IOCTL:\n“);
   fprintf(stderr, „             Functioncode = %d\n“, functioncode);
   fprintf(stderr, „             Arguments = %d\n“, arg);
#endif

   switch (functioncode) {
   case FPGA_GET_NAME:
      s = malloc(strlen(device->xc4000_name) + 1);
      strcpy(s, device->xc4000_name);
```

```
      return (int) s;
      break;
   case FPGA_GET_VENDOR:
      s = malloc(strlen(device->xc4000_vendor) + 1);
      strcpy(s, device->xc4000_vendor);
      return (int) s;
      break;
   case FPGA_GET_STATUS:
      if (*(device->xc4000_io) & 0x80)
    return FPGA_READY;
      else
    return FPGA_NOT_READY;
      break;
   default:
      return -1;
   }
}

int XC4000_Open(DEV_XC4000 * device, char *remainder, int mode)
{
   struct timespec *t;
   STATUS status;

   status = semTake(device->xc4000_semId, NO_WAIT);
   if (status != OK) {
      errnoSet((int) FPGA_BUSY);
      return ERROR;
   }
   t = (struct timespec *) malloc(sizeof(struct timespec));

   /*fprintf(stderr, „XC4000_Open: remainder = ,%s‘\n“, remainder); */
   if (remainder[0] != 0)
      return ERROR;

   t->tv_sec = (time_t) 0;
   t->tv_nsec = 4000;/* 4 usec */
   *(device->xc4000_prog) = 0xc0;
   nanosleep(t, NULL);/* SlaveOut on   */
   *(device->xc4000_prog) = 0x40;
   nanosleep(t, NULL);/* Prog off      */
   *(device->xc4000_prog) = 0xc0;
   nanosleep(t, NULL);/* Prog on       */

   /* sleep(1) .. delay 3.8 usec (40 MHz sysclock) --> wait 1.5 ms minimum  */
   t->tv_sec = 0;
   t->tv_nsec = 1500000;/* 1.5 ms */
   nanosleep(t, NULL);
   /*mysleep(400); */

   return (int) device;
}


int XC4000_Create(DEV_XC4000 * device, char *remainder, int mode)
{

#ifdef XC4000_DEBUG
   fprintf(stderr, „XC4000_Create\n“);
#endif

   return OK;
}
```

```
int XC4000_Close(DEV_XC4000 * device)
{
#ifdef XC4000_DEBUG
   fprintf(stderr, „XC4000_Close\n“);
   fprintf(stderr, „prog = 0x%02x \n“, *(device->xc4000_prog));
#endif
   *(device->xc4000_prog) = 0x80;/* switch off ‚slaveout‘ */
   mysleep(1);
#ifdef XC4000_DEBUG
   fprintf(stderr, „prog = 0x%02x \n“, *(device->xc4000_prog));
   fprintf(stderr, „Close Done.\n“);
#endif

   semGive(device->xc4000_semId);
   return OK;
}


/*
   Install driver for a xc4000 device:

   parameter:     progaddr .. address of prog register
   ioaddr   .. address of io register
   name     .. device name (ascii string)
   vendor   .. vendor name (    „       )
   node     .. device path (    „       )

   example:

   InstallXC4000( 0xf3000000, 0xf3400000, „xc4010e“, „Xilinx“, „/dev/fpga0“ );

   All I/O operations to „/dev/fpga0“ will be redirected to the device driver.
 */


STATUS InstallXC4000(unsigned char *progaddr, unsigned char *ioaddr,
                     char *name, char *vendor, char *node)
{
   int DriverNumber;
   SEM_ID semId;
   STATUS status;

   DEV_XC4000 *dev_fpga;


   DriverNumber = iosDrvInstall(
             (FUNCPTR) XC4000_Create,
             NULL,
             (FUNCPTR) XC4000_Open,
             (FUNCPTR) XC4000_Close,
             (FUNCPTR) XC4000_Read,
             (FUNCPTR) XC4000_Write,
             (FUNCPTR) XC4000_IOCTL
       );

#ifdef XC4000_DEBUG
   fprintf(stderr, „Device Number = %d\n“, DriverNumber);
#endif

   semId = semBCreate(SEM_Q_FIFO, SEM_FULL);
   if (semId == NULL) {
#ifdef XC4000_DEBUG
```

```
      perror(„Can not create semaphore for /dev/fpga0“);
#endif
      return ERROR;
   }
   dev_fpga = (DEV_XC4000 *) malloc(sizeof(DEV_XC4000));
   dev_fpga->xc4000_prog = (unsigned char *) 0xf3000000;
   dev_fpga->xc4000_io = (unsigned char *) 0xf3400000;
   dev_fpga->xc4000_name = (char *) malloc(8);
   strcpy(dev_fpga->xc4000_name, name);
   dev_fpga->xc4000_vendor = (char *) malloc(8);
   strcpy(dev_fpga->xc4000_vendor, vendor);
   dev_fpga->xc4000_semId = semId;
   dev_fpga->xc4000_node = (char *) malloc(16);
   strcpy(dev_fpga->xc4000_node, node);

   status = iosDevAdd((DEV_HDR *) dev_fpga, dev_fpga->xc4000_node,
            DriverNumber);

   return status;
}


/*****************************************************************************/

/* support for xilinx 4010e device (spyder core p1) */

void Drv()
{
   InstallXC4000((unsigned char *) 0xf3000000, (unsigned char *) 0xf3400000,
                 „xc4010e“, „Xilinx“, „/dev/fpga0“);
}
#undef XC4000_DEBUG


/*---------------------------------------------------------------------------

                      Treiber fuer XC-4000 FPGA
                     ---------------------------

  Datei: xc4000.h

---------------------------------------------------------------------------*/

#define FPGA_READY 0
#define FPGA_NOT_READY 1
#define FPGA_BUSY (M_ioLib | 20)
#define FPGA_GET_NAME     0
#define FPGA_GET_VENDOR   1
#define FPGA_GET_STATUS   2

/* header struct */

typedef struct {
   DL_NODE node;
   short drvNum;
   char *name;
   unsigned char *xc4000_prog;/* address of prog register */
   unsigned char *xc4000_io;  /* address of io register   */
   char *xc4000_name;         /* device name string       */
   char *xc4000_vendor;       /* vendor string            */
   SEM_ID xc4000_semId;       /* semaphore id             */
   char *xc4000_node;         /* device path              */
} DEV_XC4000;
```

```
/*-------------------------------------------------------------------------

                    Treiber fuer XC-4000 FPGA
                    =========================

 Datei: Makefile

-------------------------------------------------------------------------*/


LIB_PATH = C:/tornado / target / lib
CC = ccppc
OPTIONS = -DCPU = PPC403 \
-DVXWORKS - DNOFORK \
-BC:\Tornado / host / x86 - win32 / lib / gcc - lib / \
-mcpu = 403 - g - mstrict - align - ansi - nostdinc - O2 \
-fvolatile - fno - builtin - fno - for -scope
   \
       -Wall - I / h - I. \
 -IC: \Tornado \ target \ config \ all \
 -IC: \Tornado \ target / h \
 -IC: \Tornado \ target / src / config \
 -IC: \Tornado \ target / src / drv




 xc4000.o:xc4000.c
       $(CC) - c $(OPTIONS) xc4000.c

 xc4000.i:xc4000.c
       cd 1.0 $(CC) - c $(OPTIONS) - E - <xc4000.c > xc4000.i


 install:xc4000.o
       arppc - r $(LIB_PATH) / libPPC403gnuvx.a xc4000.o

 clean:
   rm - f xc4000.o
   rm - f xc4000.i
```

Anhang G

FTP Server

```
/*-------------------------------------------------------------------------
                                FTP Server
                               ============

 Datei: ftpd.c (enthält main routine)

-------------------------------------------------------------------------*/

#include „ftpd.h“
#include <parsearg.c>
#include <alice.socket.c>
#include <file.c>
#include <convert.c>

unsigned int msgsock, sock_out;
unsigned int pid;

void upperstring(char *in);
void splitarg(char *string, char d, char **Command, char **Options);
void parsearg(char *string, char d, int *argc, char ***argv);
void do_list(FILE * f, char *path, int verbose);
char *GetUnixlikeEntry(char *dirname, char *basename);
void free_args(int *margc, char ***margv);

/* ------------------------------------------------------------------*/


/*
   tools used by misc. functions
   ==============================

 */
```

```
/*
   int FPRINTF(FILE * fp, const char *fmt,...)
   {
   int ret = 0;
   va_list ap;

   if ((fp == stderr) && (LOG))
   {
   va_start (ap, fmt);
   ret = FPRINTF (fp, fmt, ap);
   va_end (ap);
   }

   if ( fp != stderr )
   {
   va_start (ap, fmt);
   ret = FPRINTF (fp, fmt, ap);
   va_end (ap);
   }

   return ret;
   }
 */


#ifdef VXWORKS
char *rstrerror(int errorcode)
{
   char *s;
   char e[100];

   sprintf(e, „(%d)“, errorcode);
   s = (char *) malloc(256);

   switch (errorcode) {
   case S_dosFsLib_VOLUME_NOT_AVAILABLE:
      strcpy(s, „Volume not available“);
      break;
   case S_dosFsLib_DISK_FULL:
      strcpy(s, „Disk full“);
      break;
   case S_dosFsLib_FILE_NOT_FOUND:
      strcpy(s, „File not found“);
      break;
   case S_dosFsLib_NO_FREE_FILE_DESCRIPTORS:
      strcpy(s, „No more free file descriptors“);
      break;
   case S_dosFsLib_INVALID_NUMBER_OF_BYTES:
      strcpy(s, „Invalid number of bytes“);
      break;
   case S_dosFsLib_FILE_ALREADY_EXISTS:
      strcpy(s, „File already exists“);
      break;
   case S_dosFsLib_ILLEGAL_NAME:
      strcpy(s, „Illegal name“);
      break;
   case S_dosFsLib_CANT_DEL_ROOT:
      strcpy(s, „Can not delete root“);
      break;
   case S_dosFsLib_NOT_FILE:
      strcpy(s, „Not a file“);
      break;
```

```
case S_dosFsLib_NOT_DIRECTORY:
   strcpy(s, „Not a directory“);
   break;
case S_dosFsLib_NOT_SAME_VOLUME:
   strcpy(s, „Not same volume“);
   break;
case S_dosFsLib_READ_ONLY:
   strcpy(s, „Read only“);
   break;
case S_dosFsLib_ROOT_DIR_FULL:
   strcpy(s, „Root directory full“);
   break;
case S_dosFsLib_DIR_NOT_EMPTY:
   strcpy(s, „Directory not empty“);
   break;
case S_dosFsLib_BAD_DISK:
   strcpy(s, „Bad Disk“);
   break;
case S_dosFsLib_NO_LABEL:
   strcpy(s, „No Label“);
   break;
case S_dosFsLib_INVALID_PARAMETER:
   strcpy(s, „Invalid parameter“);
   break;
case S_dosFsLib_NO_CONTIG_SPACE:
   strcpy(s, „No config space“);
   break;
case S_dosFsLib_CANT_CHANGE_ROOT:
   strcpy(s, „Can not change root“);
   break;
case S_dosFsLib_FD_OBSOLETE:
   strcpy(s, „File descriptor obsolete“);
   break;
case S_dosFsLib_DELETED:
   strcpy(s, „Deleted“);
   break;
case S_dosFsLib_NO_BLOCK_DEVICE:
   strcpy(s, „No block device“);
   break;
case S_dosFsLib_BAD_SEEK:
   strcpy(s, „Bad seek“);
   break;
case S_dosFsLib_INTERNAL_ERROR:
   strcpy(s, „Internal error“);
   break;
case S_dosFsLib_WRITE_ONLY:
   strcpy(s, „Write only“);
   break;
case S_ioLib_NO_DRIVER:
   strcpy(s, „No Driver“);
   break;
case S_ioLib_UNKNOWN_REQUEST:
   strcpy(s, „Unknown request“);
   break;
case S_ioLib_DEVICE_ERROR:
   strcpy(s, „Device error“);
   break;
case S_ioLib_DEVICE_TIMEOUT:
   strcpy(s, „Device timeout“);
   break;
case S_ioLib_WRITE_PROTECTED:
   strcpy(s, „Write protected“);
   break;
```

```
   case S_ioLib_DISK_NOT_PRESENT:
      strcpy(s, „Disk not present“);
      break;
   case S_ioLib_NO_FILENAME:
      strcpy(s, „No filename“);
      break;
   case S_ioLib_CANCELLED:
      strcpy(s, „Cancelled“);
      break;
   case S_ioLib_NO_DEVICE_NAME_IN_PATH:
      strcpy(s, „No device name in path“);
      break;
   case S_ioLib_NAME_TOO_LONG:
      strcpy(s, „Name too long“);
      break;
   case S_ioLib_UNFORMATED:
      strcpy(s, „Unformated“);
      break;
   case FPGA_BUSY:
      strcpy(s, „Device busy“);
      break;
   default:
      strerror_r(errorcode, s);
   }
   return s;
}
#endif

#ifdef VXWORKS
int RegisterConnection()
{
   int TaskId, i, count;
   char *s;

   for (i = 0, count = 0; i < MAX_CONNECTIONS; i++)
      if (connection[i].TaskId)
    count++;

   if (count < MAX_CONNECTIONS) {
      TaskId = taskIdSelf();
      for (i = 0; connection[i].TaskId; i++);
      connection[i].TaskId = TaskId;
      s = (char *) malloc(MAXPATHLEN);
      if (s) {
    strcpy(s, „/“);
    connection[i].virtual_cwd = s;
      } else {
    PERROR(„malloc“);
    return -1;
      }
   }
   return count;
}
#endif


#ifdef VXWORKS
void UnRegisterConnection()
{
   int TaskId;
   int i;
   char *s;
```

```
   TaskId = taskIdSelf();
   for (i = 0; (i<MAX_CONNECTIONS) && (connection[i].TaskId != TaskId); i++);
   if (i < MAX_CONNECTIONS) {
      connection[i].TaskId = NULL;
      s = connection[i].virtual_cwd;
      if (s)
    free(s);
      s = NULL;
   }
}
#endif

int blacklist(char *path)
{
   char buf[MAXPATHLEN];
   int margc, i;
   char **margv;
   int found = 0;


   parsearg(SPECIALDIRS, ,:‘, &margc, &margv);
   Realpath(path, buf);


   for (i = 0; i < margc; i++)
      if (strstr(buf, margv[i]) == buf) {
    if (!strcmp(buf, margv[i]))
       found = 1;
    else if (strstr(buf + strlen(margv[i]), „/“))
       found = 1;
      }
   free_args(&margc, &margv);
   return found;
}

int blacklist2(char *path)
{
   char buf[MAXPATHLEN];
   int margc, i;
   char **margv;


   parsearg(SPECIALFILES, ,:‘, &margc, &margv);

   Realpath(path, buf);


   for (i = 0; i < margc; i++)
      if (!strcmp(path, margv[i]))
    return 1;

   free_args(&margc, &margv);
   return 0;
}



int whitelist(char *path)
{
   char buf[MAXPATHLEN];
   int margc, i;
   char **margv;
```

```
   parsearg(SPECIALROOT, ,:‘, &margc, &margv);
   Realpath(path, buf);

   for (i = 0; i < margc; i++)
      if (!strcmp(buf, margv[i]))
    return 1;

   free_args(&margc, &margv);
   return 0;
}


/*
   find the device_name

   example:      xc4010e    --> /dev/fpga0
   xc4010e-1  --> /dev/fpga1   (second fpga)

   If theres no match, NULL will be returned;
*/


char *fpga_device_name(char *fpga_name)
{
   int i;
   int devicenames_margc, specialfiles_margc;
   char **devicenames_margv, **specialfiles_margv;
   char *device_name = NULL;

   parsearg(SPECIALFILES, ,:‘, &specialfiles_margc, &specialfiles_margv);
   parsearg(DEVICENAMES, ,:‘, &devicenames_margc, &devicenames_margv);

   for (i = 0; i < specialfiles_margc; i++) {
      if (!strcmp(fpga_name, specialfiles_margv[i]))
    if (strcmp(devicenames_margv[i], „ „)) {
       device_name = (char *) malloc(strlen(devicenames_margv[i]) + 1);
       strcpy(device_name, devicenames_margv[i]);
    }
   }

   free_args(&specialfiles_margc, &specialfiles_margv);
   free_args(&devicenames_margc, &devicenames_margv);

   return device_name;
}


char *GetUserName(int userid)
{
   char *username, *uid;
   int margc, i;
   char **margv;
   char *s0, *s;

   s = (char *) malloc(8);
   sprintf(s, „%d“, userid);

   parsearg(UID, ,,‘, &margc, &margv);

   for (i = 0, s0 = NULL; (i < margc) && (s0 == NULL); i++) {
      splitarg(margv[i], ,:‘, &uid, &username);
      if ((uid && username) && (!strcmp(uid, s))) {
```

```
     s0 = (char *) malloc(strlen(username));
     strcpy(s0, username);
        }
        if (uid)
     free(uid);
        if (username)
     free(username);
    }

    free_args(&margc, &margv);
    if (s0)
       return s0;
    else
       return s;
}


char *GetGroupName(int groupid)
{
    char *groupname, *gid;
    int margc, i;
    char **margv;
    char *s0, *s;

    s = (char *) malloc(8);
    sprintf(s, „%d“, groupid);

    parsearg(GID, ,,‘, &margc, &margv);

    for (i = 0, s0 = NULL; (i < margc) && (s0 == NULL); i++) {
       splitarg(margv[i], ,:‘, &gid, &groupname);
       if ((gid && groupname) && (!strcmp(gid, s))) {
     s0 = (char *) malloc(strlen(groupname));
     strcpy(s0, groupname);
       }
       if (gid)
     free(gid);
       if (groupname)
     free(groupname);
    }

    free_args(&margc, &margv);
    if (s0)
       return s0;
    else
       return s;
}




void do_special_list(FILE * data, int verbose)
{
    int margc, i;
    char **margv;

    parsearg(SPECIALFILES, ,:‘, &margc, &margv);
    for (i = 0; i < margc; i++)
     if (verbose)
     FPRINTF(data,
     „d-w--w--w-   1 root     root         1024 Feb 23 23:12 %s\r\n“,margv[i]);
     else
      FPRINTF(data, „%s\r\n“, margv[i]);
```

```
   free_args(&margc, &margv);
}


void CopyToFPGA(FILE * f, char *source_path, char *dest_path)
{
   char buf[MAXPATHLEN];
   int special_margc, dest_margc, i;
   char *fpga_name = NULL;
   char **special_margv, **dest_margv;
   int pos, format, error;
   char *dirname, *basename;
   char resolved_path[MAXPATHLEN], *path;
   FILE *fpga, *source;
   int devicedescriptor, sourcedescriptor;



   /* compare the last component of the destinations directory part and
      FPGA names

      example:

      a)

      SPECIALFILES = „4010“ , „4010XL“, „4036“   .. special directories
      dest = „/fpga/4010XL/design.rbt“
      ^^^^^^
      design.rbt will be load to a 4010XL device.


      b)

      SPECIALFILES = „4010“ , „4010XL“, „4036“   .. special directories
      dest = „/fpga/unknown/design.rbt“
      ^^^^^^

      ‚unknown‘ does not match any fpga name. an error will be returned.

    */

   parsearg(SPECIALFILES, ‚:‘, &special_margc, &special_margv);
   Realpath(dest_path, buf);
   parsearg(buf, ‚/‘, &dest_margc, &dest_margv);


   /* example: dest = /fpga/4010/design.rbt */
   pos = dest_margc - 2;
   for (i = 0; i < special_margc; i++) {
      if (!strcmp(dest_margv[pos], special_margv[i]))
    fpga_name = special_margv[i];
   }

   /* example: dest = /fpga/4010 */
   if (fpga_name == NULL) {
      pos = dest_margc - 1;
      for (i = 0; i < special_margc; i++) {
    if (!strcmp(dest_margv[pos], special_margv[i]))
       fpga_name = special_margv[i];
      }
   }
   if (fpga_name == NULL) {
```

```
      splitarg(buf, , ,, &dirname, &basename);
      FPRINTF(f, „550 %s: No such device.\n“, dirname);
      FPRINTF(stderr, „550 %s: No such device.\n“, dirname);
      free_args(&special_margc, &special_margv);
      free_args(&dest_margc, &dest_margv);
      if (dirname)
    free(dirname);
      if (basename)
    free(basename);
      return;
   }
   FPRINTF(stderr, „Loading to %s device\n“, fpga_name);

   path = fpga_device_name(fpga_name);
   if (path == NULL) {
      FPRINTF(f, „550 %s: No matching device found.\r\n“, fpga_name);
      FPRINTF(stderr, „550 %s: No matching device found.\n“, fpga_name);
      return;
   }
   FPRINTF(stderr, „Using %s device\n“, path);

   source = fopen(Realpath(source_path, resolved_path), „r“);
   if (source == NULL) {
      FPRINTF(f, „550 %s: %s.\r\n“, resolved_path, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, resolved_path, STRERROR(errno));
      return;
   }
   fpga = fopen(Realpath(path, resolved_path), „wb“);
   if (fpga == NULL) {
      error = errno;
      if (error == S_dosFsLib_FILE_NOT_FOUND) {
    FPRINTF(f, „550 %s: No such device.\r\n“, resolved_path);
    FPRINTF(stderr, „550 %s: No such device.\n“, resolved_path);
      } else {
    FPRINTF(f, „550 %s: %s.\r\n“, resolved_path, STRERROR(error));
    FPRINTF(stderr, „550 %s: %s.\n“, resolved_path, STRERROR(error));
      }
      return;
   }
   sourcedescriptor = fileno(source);
   devicedescriptor = fileno(fpga);
   format = LoadFPGA(sourcedescriptor, devicedescriptor);



   if (format == UNKNOWN) {
      FPRINTF(f, „550 %s unsupported file format\r\n“, source_path);
      FPRINTF(stderr, „550 %s unsupported file format\n“, source_path);
   } else {
      FPRINTF(f, „250 %s device loaded successfully\r\n“, fpga_name);
      FPRINTF(stderr, „250 %s device loaded successfully\n“, fpga_name);
   }

   if (fpga)
      fclose(fpga);
   if (source)
      fclose(source);
   free_args(&special_margc, &special_margv);
   free_args(&dest_margc, &dest_margv);
   return;
}
```

```
void UploadToFPGA(FILE * f, int *passive_mode, int *data_sock,
                  int *data_ip_addr, int *data_port, char *dest_path)
{
   char buf[MAXPATHLEN];
   int special_margc, dest_margc, i;
   char *fpga_name = NULL;
   char **special_margv, **dest_margv;
   int pos, format, error;
   char *dirname, *basename;
   char resolved_path[MAXPATHLEN];
   int sourcedescriptor, devicedescriptor;
   char *path;
   FILE *data, *fpga;


   /* compare the last component of the destinations directory part with
      FPGA names

      example:

      a)

      SPECIALFILES = „4010“ , „4010XL“, „4036“   .. special directories
      dest = „/fpga/4010XL/design.rbt“
      ^^^^^^
      design.rbt will be load to a 4010XL device.


      b)

      SPECIALFILES = „4010“ , „4010XL“, „4036“   .. special directories
      dest = „/fpga/unknown/design.rbt“
      ^^^^^^

      ‚unknown‘ does not match any fpga name. an error will be returned.

    */

   parsearg(SPECIALFILES, ‚:‘, &special_margc, &special_margv);
   Realpath(dest_path, buf);
   parsearg(buf, ‚/‘, &dest_margc, &dest_margv);

   /* example: dest = /fpga/4010/design.rbt */
   pos = dest_margc - 2;
   for (i = 0; i < special_margc; i++) {
      if (!strcmp(dest_margv[pos], special_margv[i]))
    fpga_name = special_margv[i];
   }

   /* example: dest = /fpga/4010 */
   if (fpga_name == NULL) {
      pos = dest_margc - 1;
      for (i = 0; i < special_margc; i++) {
    if (!strcmp(dest_margv[pos], special_margv[i]))
       fpga_name = special_margv[i];
      }
   }
   /* error */
   if (fpga_name == NULL) {
      splitarg(buf, , ,, &dirname, &basename);
      FPRINTF(f, „550 %s: No matching device for this directory.\r\n“,
      dirname);
      FPRINTF(stderr, „550 %s: No matching device for this directory.\n“,
```

```
   dirname);
   free_args(&special_margc, &special_margv);
   free_args(&dest_margc, &dest_margv);
   if (dirname)
 free(dirname);
   if (basename)
 free(basename);
   return;
}
/* get the file and load the fpga device */

if (
 MakeDataConnection(f, passive_mode, data_sock, data_ip_addr,
                         data_port, &sock_out, &data, „r“)
    )
   return;

FPRINTF(stderr, „UploadToFPGA: Data Connection established\n“);


/* Realpath(dest, resolved_path); */
FPRINTF(stderr, „Loading to %s device\n“, fpga_name);


path = fpga_device_name(fpga_name);
if (path == NULL) {
   FPRINTF(f, „550 %s: No matching device found.\r\n“, fpga_name);
   FPRINTF(stderr, „550 %s: No matching device found.\n“, fpga_name);
   return;
}
fpga = fopen(path, „wb“);
if (fpga == NULL) {
   Realpath(path, resolved_path);
   error = errno;
   if (error == S_dosFsLib_FILE_NOT_FOUND) {
 FPRINTF(f, „550 %s: No such device.\r\n“, resolved_path);
 FPRINTF(stderr, „550 %s: No such device.\n“, resolved_path);
   } else {
 FPRINTF(f, „550 %s: %s.\r\n“, resolved_path, STRERROR(error));
 FPRINTF(stderr, „550 %s: %s.\n“, resolved_path, STRERROR(error));
   }
   return;
}
FPRINTF(f, „150 Opening BINARY mode data connection for %s.\r\n“,
        resolved_path);
fflush(f);

FPRINTF(stderr, „150 Opening BINARY mode data connection for %s.\n“,
       resolved_path);
fflush(stderr);

FPRINTF(stderr, „Waiting for data\n“);

sourcedescriptor = fileno(data);
devicedescriptor = fileno(fpga);

format = LoadFPGA(sourcedescriptor, devicedescriptor);

if (feof(f))
   FPRINTF(stderr, „Client has closed control connection\n“);

if (fpga)
   fclose(fpga);
```

```
   if (format == UNKNOWN) {
      FPRINTF(f, „550 File format not supported\r\n“);
      fflush(f);
      FPRINTF(stderr, „550 File format not supported.\n“);
      fflush(stderr);
   } else {
      FPRINTF(f, „226 Transfer complete.\r\n“);
      fflush(f);
      FPRINTF(stderr, „226 Transfer complete.\n“);
      fflush(stderr);
   }

   if (data)
      fclose(data);

   *data_ip_addr = 0;/* invalidate old ip_address and port */
   *data_port = 0;
   if (*passive_mode)
      *passive_mode = 0;/* return to active mode */

   free_args(&special_margc, &special_margv);
   free_args(&dest_margc, &dest_margv);
   return;

}


int checkpass(char *user, char *pass)
{
   if (pass == NULL)
      return 0;
   if (user == NULL)
      return 0;

   if ((!strcmp(user, „ftp“) || !strcmp(user, „anonymous“)) &&
        strstr(pass, „@“))
      return 1;
   else if ((!strcmp(user, „fert“)) && (!strcmp(pass, „fert0“)))
      return 1;
   else
      return 0;
}


void term()
{
   FPRINTF(stderr, „good bye.\n“);
   close(msgsock);
   close(sock_out);

#ifdef VXWORKS
   taskDelete(taskIdSelf());
#endif

#ifdef LINUX
   exit(0);
#endif/* ; */
}

char *ReadString(FILE * f)
{
```

```
   char s[1024], *r;
   unsigned int l;
   unsigned char *status;

   status = fgets(s, 1023, f);
   if (status == NULL) {
      return NULL;/* Error. (maybe client have closed socket) */
   }
   l = strlen(s);
   if ((s[l - 1] == 0x0a) || (s[l - 1] == 0x0d))
      s[l - 1] = 0;/* strip lf/cr */
   if ((s[l - 2] == 0x0a) || (s[l - 2] == 0x0d))
      s[l - 2] = 0;
   r = (char *) malloc(strlen(s) + 1);
   if (r == NULL) {
      FPRINTF(stderr, „malloc failed\n“);
      PERROR(„“);
   }
   strcpy(r, s);
   return r;
}


int IsAString(unsigned char *s)
{
   int i;
   for (i = 0; i < strlen(s); i++)
      if ((s[i] < 32) || (s[i] > 128))
    return 0;
   return 1;
}


int MakeDataConnection(FILE * f, int *passive_mode, int *data_sock,
                       int *data_ip_addr, int *data_port,
                       int *sock_out, FILE ** data, char *mode)
{
   char host[256];


   if (*passive_mode) {

      *sock_out = accept(*data_sock, 0, 0);
      if (*sock_out == -1) {
       FPRINTF(f, „425 Can‘t build data connection: Connection refused.\r\n“);
       FPRINTF(stderr,
               „425 Can‘t build data connection: Connection refused.\n“);
    return -1;
      }
      *data = fdopen(*sock_out, mode);
      if (*data == NULL) {
    FPRINTF(f,
                „425 Can‘t build data connection: Connection refused.\r\n“);
    FPRINTF(stderr,
                „425 Can‘t build data connection: Connection refused.\n“);
    return -1;
      }
   } else {

      if ((*data_ip_addr == 0) || (*data_port == 0)) {
    FPRINTF(stderr,
                 „425 Can‘t build data connection: Connection refused.\n“);
    FPRINTF(f,
```

```
                „425 Can‘t build data connection: Connection refused.\r\n“);
    return -1;
      }
      FPRINTF(stderr, „IP address = 0x%08x\n“, *data_ip_addr);
      sprintf(host, „%d.%d.%d.%d“,
        (*data_ip_addr & 0xff000000) >> 24,
        (*data_ip_addr & 0x00ff0000) >> 16,
        (*data_ip_addr & 0x0000ff00) >> 8,
        (*data_ip_addr & 0x000000ff));
      FPRINTF(stderr, „Host name = %s\n“, host);

      *sock_out = connect_to(host, *data_port);
      if (*sock_out == 0) {
    FPRINTF(f,
                „425 Can‘t build data connection: Connection refused.\r\n“);
    FPRINTF(stderr,
                „425 Can‘t build data connection: Connection refused.\n“);
    return -1;
      }
      *data = fdopen(*sock_out, mode);
      if (*data == NULL) {
    FPRINTF(f,
                „425 Can‘t build data connection: Connection refused.\r\n“);
    FPRINTF(stderr,
                „425 Can‘t build data connection: Connection refused.\n“);
    return -1;
      }
   }

   return 0;/* on success */
}




/* free memory used by margv struct */

void free_args(int *margc, char ***margv)
{
   unsigned int i;

   if (*margv == NULL)
      return;       /* nothing to free */

   for (i = 0; i < *margc; i++)
      if (*((*margv) + i))
    free(*((*margv) + i));

   if (*margv)
      free(*margv);
}


/* ---------------------------------------------------------------------- */

/*
   Partial implementaion of the ftp command set.
   ============================================

   - used by the connection  task (function ‚fnord‘)

 */
```

```
void cmd_noop(FILE * f)
{
   FPRINTF(f, „200 NOOP command succussful\r\n“);
   FPRINTF(stderr, „200 NOOP command succussful\n“);
}


void cmd_type(FILE * f, char *type, char *Command, char *Options)
{
   int margc;
   char **margv;
   char t;


   parsearg(Options, , ,, &margc, &margv);

   if (margc != 1) {
      FPRINTF(f,
      „500 ,%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
      „500 ,%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }
   upperstring(margv[0]);

   if (!strcmp(margv[0], „A“))
      t = ‚A‘;
   else if (!strcmp(margv[0], „E“))
      t = ‚E‘;
   else if (!strcmp(margv[0], „I“))
      t = ‚I‘;
   else if (!strcmp(margv[0], „L“))
      t = ‚L‘;
   else {
      FPRINTF(f,
      „500 ,%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
      „500 ,%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }

   FPRINTF(f, „200 Type set to %c.\r\n“, t);
   FPRINTF(stderr, „200 Type set to %c.\n“, t);

   *type = t;
   free_args(&margc, &margv);
   return;
}


void cmd_cwd(FILE * f, char **user, char **pass, char *Command, char *Options)
{
   int error;
   char resolved_path[MAXPATHLEN];
#ifdef VXWORKS
   char *EntryString;
#endif

   if (!checkpass(*user, *pass)) {
```

```
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   /* without arguments stay in current directry */
   if (!strcmp(Options, „“)) {
      FPRINTF(f, „250 CWD command successful.\r\n“);
      FPRINTF(stderr, „250 CWD command successful.\n“);
      return;
   }
   /* fake access to special directories */


   if (blacklist(Realpath(Options, resolved_path))) {
      SETCWD(resolved_path);
      FPRINTF(f, „250 CWD command successful.\r\n“);
      FPRINTF(stderr, „250 CWD command successful.\n“);
      return;
   }
   /* the vxworks braindead chdir-function does not recognize
      invalid paths..
    */

#ifdef VXWORKS
   EntryString = GetUnixlikeEntry(„“, Realpath(Options, resolved_path));
   if (EntryString == NULL) {
      FPRINTF(f, „550 %s: %s.\r\n“, Options, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, Options, STRERROR(errno));
      if (EntryString)
    free(EntryString);
      return;/* maybe directory does not exist */
   }
   if (EntryString[0] != ‚d‘) {
      FPRINTF(f, „550 %s: Is not a directory.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Is not a directory.\n“, Options);
      if (EntryString)
    free(EntryString);
      return;
   }
#endif


   if (chdir(Realpath(Options, resolved_path)) == -1) {
      error = errno;
      FPRINTF(f, „550 %s: %s\r\n“, Options, STRERROR(error));
      FPRINTF(stderr, „550 %s: %s\n“, Options, STRERROR(error));
      return;
   }
   SETCWD(resolved_path);
   FPRINTF(f, „250 CWD command successful.\r\n“);
   FPRINTF(stderr, „250 CWD command successful.\n“);
   return;
}

void cmd_cdup(FILE * f, char **user, char **pass, char *Command,
              char *Options)
{
   char d[4];

   strcpy(d, „..“);
   cmd_cwd(f, user, pass, Command, d);
}
```

```
void cmd_quit(FILE * f, int *do_exit)
{
   FPRINTF(f, „221 Goodbye.\r\n“);
   FPRINTF(stderr, „221 Goodbye.\n“);
   *do_exit = 1;
}

char *cmd_user(FILE * f, char *Command, char *Options, int ConnectionNo,
               int *do_exit)
{

   char *s;

   if (!strcmp(Options, „“)) {
      /* No Argument */
      FPRINTF(f, „500 ,USER ,: command not understood.\r\n“);
      FPRINTF(stderr, „500 ,USER ,: command not understood.\n“);
      return NULL;
   }
   s = (char *) malloc(strlen(Options) + 1);
   strcpy(s, Options);

   if (ConnectionNo >= MAX_CONNECTIONS) {
      FPRINTF(f,
              „530 No more connections. (max = %d)\r\n“, MAX_CONNECTIONS);
      FPRINTF(stderr,
              „530 No more connections. (max = %d)\n“, MAX_CONNECTIONS);
      *do_exit = 1;/*  refuse login */
   } else {

      if (!strcmp(s, „anonymous“) || !strcmp(s, „ftp“)) {
    if (!ANONYMOUS) {
       FPRINTF(f, „530 User %s unknown.\r\n“, s);
       FPRINTF(stderr, „530 User %s unknown.\n“, s);
       *do_exit = 1;
    } else {
       FPRINTF(f, „331 Guest login ok, send your complete e-mail \
                    address as password.\r\n“);
       FPRINTF(stderr, „331 Guest login ok, send your complete e-mail \
                   address as password.\n“);
    }
      } else {
    FPRINTF(f, „331 Password required for %s\r\n“, s);
    FPRINTF(stderr, „331 Password required for %s\n“, s);
      }
   }

   return s;


}

char *cmd_pass(FILE * f, char **user, char *Command, char *Options,
               int ConnectionNo)
{
   char *s;

   if (*user == NULL) {
      FPRINTF(f, „503 Login with USER first.\r\n“);
      FPRINTF(stderr, „503 Login with USER first.\n“);
      return NULL;
   }
   if (!checkpass(*user, Options)) {
```

```
      FPRINTF(f, „530 Login incorrect.\r\n“);
      FPRINTF(stderr, „530 Login incorrect.\n“);
      *user = NULL;
      return NULL;
   }
   FPRINTF(f, „230 User %s logged in. (%d / %d)\r\n“,
           *user, ConnectionNo + 1, MAX_CONNECTIONS);
   FPRINTF(stderr, „230 User %s logged in.( %d / %d)\n“,
           *user, ConnectionNo + 1, MAX_CONNECTIONS);
   s = (char *) malloc(strlen(Options) + 1);
   strcpy(s, Options);
   return s;
}

void cmd_port(FILE * f, char **user, char **pass, int *data_ip_addr,
              int *data_port, char *Command, char *Options)
{
   int margc;
   char **margv;

   *data_ip_addr = 0;
   *data_port = 0;

   FPRINTF(stderr, „p0) Errorcode = %s\n“, STRERROR(errno));
   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, ,,‘, &margc, &margv);
   if (margc < 6) {
      FPRINTF(f, „500 Illegal PORT Command.\r\n“);
      FPRINTF(stderr, „500 Illegal PORT Command.\n“);
      free_args(&margc, &margv);
      return;
   }

   *data_ip_addr = (atoi(margv[0]) << 24) + (atoi(margv[1]) << 16) +
       (atoi(margv[2]) << 8) + atoi(margv[3]);

   *data_port = (atoi(margv[4]) << 8) + atoi(margv[5]);

   FPRINTF(f, „200 PORT command successful (%s).\r\n“, Options);
   FPRINTF(stderr, „200 PORT command successful (%s).\n“, Options);

   /*for debugging only */
   /*
      FPRINTF(f, „200- PORT command successful (%s).\r\n“, Options);
      FPRINTF(f, „ Data IP Address  = 0x%08x\r\n“, *data_ip_addr);
      FPRINTF(f, „ Data Port        = 0x%04x\r\n“, *data_port);
      FPRINTF(f, „200 \r\n“);
    */

   FPRINTF(stderr, „p2) Errorcode = %s\n“, STRERROR(errno));
   free_args(&margc, &margv);
   return;
}


void cmd_pasv(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *host_ip_addr)
{
```

```
unsigned int sock, length;
struct sockaddr_in server;
unsigned int local_port, local_ip_addr;


if (!checkpass(*user, *pass)) {
   FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
   FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
   return;
}
*passive_mode = 1;

/* create a socket */
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
   FPRINTF(f, „500 Can not enter passive mode. (socket)\r\n“);
   FPRINTF(stderr, „500 Can not enter passive mode. (socket)\n“);
   return;
}
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(0);/* get a free port */
if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
   FPRINTF(f, „500 Can not enter passive mode. (bind)\r\n“);
   FPRINTF(stderr, „500 Can not enter passive mode. (bind)\n“);
   return;
}
/* Find out assigned port number and print it out */
length = sizeof(server);
if (getsockname(sock, (struct sockaddr *) &server, &length)) {
   FPRINTF(f, „500 Can not enter passive mode. (getsockname)\r\n“);
   FPRINTF(stderr, „500 Can not enter passive mode. (getsockname)\n“);
   return;
}
local_port = ntohs(server.sin_port);
local_ip_addr = *host_ip_addr;

if (local_ip_addr == -1) {
   FPRINTF(f, „500 Can not enter passive mode. (IP_ADDR invalid)\r\n“);
   FPRINTF(stderr, „500 Can not enter passive mode. (IP_ADDR invalid)\n“);
   return;
}
if (listen(sock, 5)) {
   FPRINTF(f, „500 Can not enter passive mode. (listen)\r\n“);
   FPRINTF(stderr, „500 Can not enter passive mode. (listen)\n“);
   close(sock);
   return;
}

FPRINTF(f, „227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)\r\n“,
   (local_ip_addr & 0xff000000) >> 24,
   (local_ip_addr & 0x00ff0000) >> 16,
   (local_ip_addr & 0x0000ff00) >> 8,
   (local_ip_addr & 0x000000ff),
   (local_port & 0xff00) >> 8,
   (local_port & 0x00ff)
    );

FPRINTF(stderr, „227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)\n“,
   (local_ip_addr & 0xff000000) >> 24,
   (local_ip_addr & 0x00ff0000) >> 16,
   (local_ip_addr & 0x0000ff00) >> 8,
   (local_ip_addr & 0x000000ff),
```

```
      (local_port & 0xff00) >> 8,
      (local_port & 0x00ff)
       );


   *data_sock = sock;

}


void cmd_size(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options)
{
   int margc, entry_argc;
   char **margv, **entry_argv;
   char *EntryString;
   char resolved_path[MAXPATHLEN];

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
   if (margc < 1) {
      FPRINTF(f,
               „500 ‚%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
               „500 ‚%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Permission denied.\n“, Options);
      return;
   }

   EntryString = GetUnixlikeEntry(„“, Realpath(Options, resolved_path));
   if (EntryString == NULL) {
      FPRINTF(f, „550 %s: %s.\r\n“, Options, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, Options, STRERROR(errno));
      return;/* maybe file does not exist */
   }

   if (EntryString[0] != ‚-‘) {
      FPRINTF(f, „550 %s: not a plain file\r\n“, Options);
      FPRINTF(stderr, „550 %s: not a plain file\n“, Options);
      return;
   }
   parsearg(EntryString, , ,, &entry_argc, &entry_argv);
   FPRINTF(f, „213 %s\r\n“, entry_argv[4]);
   FPRINTF(stderr, „213 %s\n“, entry_argv[4]);

   free_args(&entry_argc, &entry_argv);
   free_args(&margc, &margv);
   return;
}
```

```
void cmd_rnfr(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options, char **RenameFrom)
{
   int margc;
   char **margv;
   char *EntryString;
   char resolved_path[MAXPATHLEN];

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      *RenameFrom = NULL;
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
   if (margc < 1) {
      FPRINTF(f,
           „500 ‚%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
           „500 ‚%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      *RenameFrom = NULL;
      return;
   }

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, resolved_path);
      FPRINTF(stderr, „550 %s: Permission denied.\n“, resolved_path);
      *RenameFrom = NULL;
      return;
   }

   EntryString = GetUnixlikeEntry(„“, Realpath(Options, resolved_path));
   if (EntryString == NULL) {
      FPRINTF(f, „550 %s: %s.\r\n“, Options, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, Options, STRERROR(errno));
      *RenameFrom = NULL;
      return;/* maybe file does not exist */
   } else {
      FPRINTF(f, „350 File exists, ready for destination name\r\n“);
      FPRINTF(stderr, „350 File exists, ready for destination name\n“);
      *RenameFrom = (char *) malloc(strlen(resolved_path) + 1);
      if (*RenameFrom == NULL) {
    PERROR(„rnfr: malloc\n“);
    return;
      }
      strcpy(*RenameFrom, resolved_path);
   }

   free_args(&margc, &margv);
   return;
}


void cmd_rnto(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options, char **RenameFrom)
{
   int margc, error;
   char **margv = NULL;
   char resolved_path[MAXPATHLEN];
```

```
   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      *RenameFrom = NULL;
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
   if (margc < 1) {
      FPRINTF(f,
              „500 ,%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
              „500 ,%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      *RenameFrom = NULL;
      return;
   }
   if (*RenameFrom == NULL) {
      FPRINTF(f, „503 Bad sequence of commands.\r\n“);
      FPRINTF(stderr, „503 Bad sequence of commands.\n“);
      return;
   }

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      CopyToFPGA(f, *RenameFrom, resolved_path);
      *RenameFrom = NULL;
      return;
   }

   /* rename it here */
   error = rename(*RenameFrom, Realpath(Options, resolved_path));
   if (error < 0) {
      FPRINTF(f, „550 rename: %s\r\n“, STRERROR(errno));
      FPRINTF(stderr, „550 rename: %s\n“, STRERROR(errno));
      *RenameFrom = NULL;
   } else {
      FPRINTF(f, „250 RNTO command successful.\r\n“);
      FPRINTF(stderr, „250 RNTO command successful.\n“);
      *RenameFrom = NULL;
   }

   free_args(&margc, &margv);
   return;
}


void cmd_retr(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options)
{
   int sock_out;
   FILE *data;
   FILE *file;
   int margc;
   char **margv;
   char *EntryString;
   int rsize = 0, wsize = 0;
   char buf[1024];
   int filedescriptor, filedescriptor2;
   char resolved_path[MAXPATHLEN];

   if (!checkpass(*user, *pass)) {
```

```
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
   if (margc < 1) {
      FPRINTF(f,
         „500 ‚%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
         „500 ‚%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Permission denied.\n“, Options);
      return;
   }

   /* check the files mode. You can download plain files only */
   EntryString = GetUnixlikeEntry(„“, Realpath(Options, resolved_path));
   if (EntryString == NULL) {
      /*FPRINTF(stderr, „GetUnixLikeEntry: failed\n“); */
      FPRINTF(f, „550 %s: %s.\r\n“, Options, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, Options, STRERROR(errno));
      return;/* maybe file does not exist */
   }

  if (EntryString[0] != ‚-‘) {
      FPRINTF(f, „550 %s: not a plain file\r\n“, Options);
      FPRINTF(stderr, „550 %s: not a plain file\n“, Options);
      return;
   }
   /* beam it up */

   file = fopen(Options, „rb“);
   if (file == NULL) {
      FPRINTF(f, „550 %s: %s.\r\n“, Options, STRERROR(errno));
      FPRINTF(stderr, „550 %s: %s.\n“, Options, STRERROR(errno));
      return;
   }
   if (
    MakeDataConnection(f, passive_mode, data_sock, data_ip_addr,
         data_port, &sock_out, &data, „rb+“)
       )
      return;

   FPRINTF(f,
            „150 Opening BINARY mode data connection for %s.\r\n“, Options);
   FPRINTF(stderr,
            „150 Opening BINARY mode data connection for %s.\n“, Options);
   fflush(f);
   fflush(stderr);

   filedescriptor = fileno(data);
   filedescriptor2 = fileno(file);
   do {
      rsize = read(filedescriptor2, buf, sizeof(buf));
      if (rsize > 0)
    wsize = write(filedescriptor, buf, rsize);
      fflush(data);
```

```
   }
   while (rsize && wsize);

   if ((rsize < 0) || (wsize < 0))
      FPRINTF(stderr, „retreive: %s\n“, STRERROR(errno));


   FPRINTF(f, „226 Transfer complete.\r\n“);
   FPRINTF(stderr, „226 Transfer complete.\n“);
   if (*passive_mode)
      close(*data_sock);
   fclose(data);
   close(sock_out);
   if (file)
      fclose(file);

   if (feof(f))
      FPRINTF(stderr, „Client has closed control connection\n“);

   *data_ip_addr = 0;/* invalidate old ip_address and port */
   *data_port = 0;
   if (*passive_mode)
      *passive_mode = 0;/* return to active mode */

   free_args(&margc, &margv);
   return;
}


void cmd_stor(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options)
{
   int sock_out;
   FILE *data;
   FILE *file;
   int margc;
   char **margv;
   char *EntryString;
   int rsize, wsize = 0;
   char buf[4096];
   int filedescriptor;
   char resolved_path[MAXPATHLEN];


   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
   if (margc < 1) {
      FPRINTF(f,
        „500 ‚%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
        „500 ‚%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      UploadToFPGA(f, passive_mode, data_sock, data_ip_addr, data_port,
```

```
                resolved_path);
   return;
}

/* If target exists, check targets mode. */

EntryString = GetUnixlikeEntry(„“, Realpath(Options, resolved_path));
if (EntryString != NULL) {
   if (EntryString[0] == ‚d‘) {
 FPRINTF(f, „553 %s: Is a directory.\r\n“, Options);
 FPRINTF(stderr, „553 %s: Is a directory.\n“, Options);
 return;
   } else if (EntryString[0] != ‚-‘) {
 FPRINTF(f,
              „553 %s: Operation not supported by device.\r\n“, Options);
 FPRINTF(stderr,
              „553 %s: Operation not supported by device.\n“, Options);
 return;
   }
}
FPRINTF(stderr, „stor: MakeDataConnection\n“);

if (
 MakeDataConnection(f, passive_mode, data_sock, data_ip_addr,
                         data_port, &sock_out, &data, „r“)
    )
   return;

FPRINTF(stderr, „stor: Data Connection established\n“);

/* get it  */
file = fopen(Realpath(Options, resolved_path), „wb“);

if (file == NULL) {
   FPRINTF(f, „550 %s: %s.\r\n“,
        Realpath(Options, resolved_path), STRERROR(errno));
   FPRINTF(stderr, „550 %s: %s.\n“,
        Realpath(Options, resolved_path), STRERROR(errno));
   return;
}
FPRINTF(f, „150 Opening BINARY mode data connection for %s.\r\n“, Options);
fflush(f);

FPRINTF(stderr, „150 Opening BINARY mode data connection for %s.\n“,
        Options);
fflush(stderr);

FPRINTF(stderr, „Waiting for data\n“);

filedescriptor = fileno(data);
do {
   rsize = read(filedescriptor, buf, sizeof(buf));
   FPRINTF(stderr, „<r%d>“, rsize);
   fflush(stderr);
   if (rsize < 0)
 PERROR(„STOR (read data)“);
   if (rsize > 0)
 wsize = fwrite(buf, 1, rsize, file);
   FPRINTF(stderr, „<w%d>“, wsize);
   fflush(stderr);
} while (rsize > 0);
FPRINTF(stderr, „done.\n“);
```

```
   if (feof(f))
      FPRINTF(stderr, „Client has closed control connection\n“);

   if (file)
      fclose(file);

   FPRINTF(f, „226 Transfer complete.\r\n“);
   fflush(f);
   FPRINTF(stderr, „226 Transfer complete.\n“);
   fflush(stderr);


   if (*passive_mode) {
      fclose(data);
      FPRINTF(stderr, „data_sock closed\n“);
   } else {
      fclose(data);
   }



   FPRINTF(stderr, „ data closed\nsock_out closed\n“);

   *data_ip_addr = 0;/* invalidate old ip_address and port */
   *data_port = 0;
   if (*passive_mode)
      *passive_mode = 0;/* return to active mode */

   free_args(&margc, &margv);

   FPRINTF(stderr, „command STOR finished\n“);
   return;
}



void cmd_list(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options)
{
   int sock_out, error;
   FILE *data;
   char *buf = NULL;

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }

   /* Create socket sock_out and stream data to communicate with client.
      If an error occurs, -1 will be returnd. Error message handling is done
      bye MakeDataConnections
    */

   FPRINTF(stderr, „Setup Data Connection\n“);
   if (
    MakeDataConnection(f, passive_mode, data_sock, data_ip_addr,
                           data_port, &sock_out, &data, „w“)
       )
      return;
```

```
    FPRINTF(stderr, „Data connection established\n“);
    FPRINTF(f, „150 Opening ASCII mode data connection for list.\r\n“);
    fflush(f);
    FPRINTF(stderr, „150 Opening ASCII mode data connection (list)\n“);

    if (!strcmp(Options, „“)) {
       buf = (char *) malloc(4096);
       if ((GETCWD(buf, 4096)) == NULL) {
     error = errno;
     FPRINTF(f, „550 %s\r\n“, STRERROR(error));
     FPRINTF(stderr, „550 %s\n“, STRERROR(error));
     return;
       } else
     FPRINTF(stderr, „list %s\n“, buf);

       do_list(data, buf, LONG);
       if (whitelist(buf))
     do_special_list(data, LONG);
       if (buf)
     free(buf);
    } else {
       do_list(data, Options, LONG);
       if (whitelist(Options))
     do_special_list(data, LONG);
    }

    if (feof(f))
       FPRINTF(stderr, „Client has closed control connection\n“);

    if (*passive_mode) {
       fclose(data);
    } else {
       fclose(data);
       close(sock_out);
    }

    FPRINTF(f, „226 Transfer complete.\r\n“);
    fflush(f);
    FPRINTF(stderr, „226 Transfer complete.\r\n“);
    fflush(stderr);



    *data_ip_addr = 0;/* invalidate old ip_address and port */
    *data_port = 0;
    if (*passive_mode)
       *passive_mode = 0;/* return to active mode */

    return;
}


void cmd_nlst(FILE * f, char **user, char **pass, int *passive_mode,
              int *data_sock, int *data_ip_addr, int *data_port,
              char *Command, char *Options)
{
    int sock_out, error;
    FILE *data;
    char *buf = NULL;


    if (!checkpass(*user, *pass)) {
       FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
```

```
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   /* Create socket sock_out and stream data to communicate with client.
      If an error occurs, -1 will be returnd. Error message handling is done
      bye MakeDataConnections
    */

   if (
    MakeDataConnection(f, passive_mode, data_sock, data_ip_addr,
          data_port, &sock_out, &data, „w“)
       )
      return;


   FPRINTF(f, „150 Opening ASCII mode data connection.\r\n“);
   FPRINTF(stderr, „150 Opening ASCII mode data connection.\n“);

   if (!strcmp(Options, „“)) {

      buf = (char *) malloc(4096);
      if ((GETCWD(buf, 4096)) == NULL) {
    error = errno;
    FPRINTF(f, „550 %s\r\n“, STRERROR(error));
    FPRINTF(stderr, „550 %s\n“, STRERROR(error));
    return;
      }
      do_list(data, buf, SHORT);
      if (buf)
    free(buf);
   } else {
      do_list(data, Options, SHORT);
   }

   if (*passive_mode)
      close(*data_sock);
   fclose(data);
   close(sock_out);

   FPRINTF(f, „226 Transfer complete.\r\n“);
   fflush(f);
   FPRINTF(stderr, „226 Transfer complete.\n“);
   fflush(f);

   *data_ip_addr = 0;/* invalidate old ip_address and port */
   *data_port = 0;
   if (*passive_mode)
      *passive_mode = 0;/* return to active mode */

}




void cmd_pwd(FILE * f, char **user, char **pass, char *Command,
             char *Options)
{
   char *buf = NULL;
   int error;

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
```

```
      return;
   }
   buf = (char *) malloc(4096);
   if ((GETCWD(buf, 4096)) == NULL) {
      error = errno;
      FPRINTF(f, „550 %s\r\n“, STRERROR(error));
      FPRINTF(stderr, „550 %s\r\n“, STRERROR(error));
      return;
   }
   FPRINTF(stderr, „pwd: cwd = ‚%s‘ (%d byte)\n“, buf, strlen(buf));
   FPRINTF(f, „257 \“%s\“ is current directory.\r\n“, buf);
   FPRINTF(stderr, „257 \“%s\“ is current directory.\n“, buf);

   if (buf)
      free(buf);
   return;

}


void cmd_mkd(FILE * f, char **user, char **pass, char *Command, char *Options)
{
   int margc;
   char **margv;

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);

   if (margc == 0) {
      FPRINTF(f, „500 ‚%s %s‘: command not understood.\r\n“, Command,
              Options);
      FPRINTF(stderr, „500 ‚%s %s‘: command not understood.\n“, Command,
              Options);
      free_args(&margc, &margv);
      return;
   }
   free_args(&margc, &margv);

   /* block access to special directories */
   if (blacklist(Options)) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Permission denied.\n“, Options);
      return;
   }
#ifdef LINUX
   if (mkdir(Options, 0x01ed) < 0) {/* 755 = 1 1110 1101 = 0x01ED */
      FPRINTF(f, „550 %s\r\n“, STRERROR(errno));
      FPRINTF(stderr, „550 %s\n“, STRERROR(errno));
      return;
   }
#endif

#ifdef VXWORKS
   if (mkdir(Options) < 0) {
      FPRINTF(f, „550 %s\r\n“, STRERROR(errno));
      FPRINTF(stderr, „550 %s\n“, STRERROR(errno));
      return;
   }
#endif
```

```
   FPRINTF(f, „257 \“%s\“ new directory created.\r\n“, Options);
   FPRINTF(stderr, „257 \“%s\“ new directory created.\n“, Options);
   return;
}

void cmd_dele(FILE * f, char **user, char **pass, char *Command,
              char *Options)
{
   int margc;
   char **margv;
   char resolved_path[MAXPATHLEN];

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);

   if (margc != 1) {
      FPRINTF(f,
               „500 ,%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
               „500 ,%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }
   free_args(&margc, &margv);

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Permission denied.\r\n“, Options);
      return;
   }

   /* 755 = 1 1110 1101 = 0x01ed */
   if (remove(Realpath(Options, resolved_path)) < 0) {
      FPRINTF(f, „550 %s\r\n“, STRERROR(errno));
      FPRINTF(stderr, „550 %s\n“, STRERROR(errno));
      return;
   }
   FPRINTF(f, „250 DELE Command successful.\r\n“);
   FPRINTF(stderr, „250 DELE Command successful.\n“);
   return;

}


void cmd_rmd(FILE * f, char **user, char **pass, char *Command, char *Options)
{
   int margc;
   char **margv;
   char resolved_path[MAXPATHLEN];

   if (!checkpass(*user, *pass)) {
      FPRINTF(f, „530 Please login with USER and PASS.\r\n“);
      FPRINTF(stderr, „530 Please login with USER and PASS.\n“);
      return;
   }
   parsearg(Options, , ,, &margc, &margv);
```

```
   if (margc != 1) {
      FPRINTF(f,
               „500 ,%s %s‘: command not understood.\r\n“, Command, Options);
      FPRINTF(stderr,
               „500 ,%s %s‘: command not understood.\n“, Command, Options);
      free_args(&margc, &margv);
      return;
   }
   free_args(&margc, &margv);

   /* block access to special directories */
   if (blacklist(Realpath(Options, resolved_path))) {
      FPRINTF(f, „550 %s: Permission denied.\r\n“, Options);
      FPRINTF(stderr, „550 %s: Permission denied.\r\n“, Options);
      return;
   }

   /* 755 = 1 1110 1101 = 0x01ed */
   if (rmdir(Realpath(Options, resolved_path)) < 0) {
      FPRINTF(f, „550 %s\r\n“, STRERROR(errno));
      FPRINTF(stderr, „550 %s\n“, STRERROR(errno));
      return;
   }
   FPRINTF(f, „250 RMD Command successful.\r\n“);
   FPRINTF(stderr, „250 RMD Command successful.\n“);
   return;

}



void cmd_syst(FILE * f)
{
   FPRINTF(f, „215 UNIX Type: L8\r\n“);
   FPRINTF(stderr, „215 UNIX Type: L8\n“);
}

void cmd_help(FILE * f)
{
 FPRINTF(f,
          „214-The following commands are recognized (* unimplemented).\r\n“);
 FPRINTF(f,“USER   PORT   STOR    MSAM*   RNTO    NLST    MKD     CDUP\r\n“);
 FPRINTF(f,“PASS   PASV   APPE*   MRSQ*   ABOR*   SITE*   XMKD*   XCUP*\r\n“);
 FPRINTF(f,“ACCT*  TYPE   MLFL*   MRCP*   DELE    SYST    RMD     STOU*\r\n“);
 FPRINTF(f,“SMNT*  STRU*  MAIL*   ALLO*   CWD     STAT*   XRMD*   SIZE\r\n);
 FPRINTF(f,“REIN*  MODE*  MSND*   REST*   XCWD*   HELP    PWD     MDTM*\r\n“);
 FPRINTF(f,“QUIT   RETR   MSOM*   RNFR    LIST    NOOP    XPWD* \r\n“);
 FPRINTF(f,“214 \r\n“);

 FPRINTF(stderr,
         „214-The following commands are recognized (* unimplemented).\n“);
 FPRINTF(stderr, „USER   PORT   STOR    MSAM*  RNTO   NLST   MKD    CDUP\n“);
 FPRINTF(stderr, „PASS   PASV   APPE*   MRSQ*  ABOR*  SITE*  XMKD*  XCUP*\n“);
 FPRINTF(stderr, „ACCT*  TYPE   MLFL*   MRCP*  DELE   SYST   RMD    STOU*\n“);
 FPRINTF(stderr, „SMNT*  STRU*  MAIL*   ALLO*  CWD    STAT*  XRMD*  SIZE\n“);
 FPRINTF(stderr, „REIN*  MODE*  MSND*   REST*  XCWD*  HELP   PWD    MDTM*\n“);
 FPRINTF(stderr, „QUIT   RETR   MSOM*   RNFR   LIST   NOOP   XPWD* \n“);
 FPRINTF(stderr, „214 \n“);
}
```

```
/* ------------------------------------------------------------------------ */

/*
   the connection handler
   ======================

   This  task/function communicates with the ftp client.

 */


void fnord(int *sock_ptr, FILE ** logfile_ptr)
{
   unsigned int data_sock, sock, msgsock;
   FILE *f, *logfile;
   char *user, *pass;
   char type = ‚I‘;
   char *s;
   char *Command, *Options, *RenameFrom;
   int data_ip_addr, data_port, passive_mode;
   struct sockaddr_in host;
   int hostsize;
   int host_ip_addr;
   int ConnectionNo;
   int do_exit = 0;

#ifdef VXWORKS
   FPRINTF(stderr, „This is Task 0x%x\n“, taskIdSelf());
   FPRINTF(stderr, „Ready to handle a new ftp connection\n“);
   fflush(stderr);
#endif

   logfile = *logfile_ptr;
   sock = *sock_ptr;

   SETCWD(„/“);

   {
      FPRINTF(stderr, „accept this connection\n“);
      msgsock = accept(sock, 0, 0);
      if (msgsock == -1) {
    PERROR(„accept“);
    return;
      } else
    FPRINTF(stderr, „accept ok\n“);

      /* get the local ip address: */
      hostsize = sizeof(host);
      getsockname(msgsock, (struct sockaddr *) &host, &hostsize);
      host_ip_addr = ntohl(host.sin_addr.s_addr);

      FPRINTF(stderr, „Open a stream socket..\n“);
      f = fdopen(msgsock, „rb+“);
      if (f == NULL) {
    FPRINTF(stderr, „Error open socket\n“);
    PERROR(„Open Stream“);
    return;
      } else
    FPRINTF(stderr, „Control connection ok\n“);
```

```
#ifdef FORK
      FPRINTF(f, „220 Ferts FTP server ready.(fork enabled)\r\n“);
      FPRINTF(stderr, „220 Ferts FTP server ready.(fork enabled)\n“);
#endif

#ifdef NOFORK
      FPRINTF(f, „220 Ferts FTP server ready.(fork disabled)\r\n“);
      FPRINTF(stderr, „220 Ferts FTP server ready.(fork disabled)\n“);
#endif

      /* initialize */

#ifdef VXWORKS
      ConnectionNo = RegisterConnection();
      if (ConnectionNo < 0) {
   FPRINTF(f, „550 Internal Server error.\r\n“);
   FPRINTF(f, „550 Internal server error.\n“);
   fclose(f);
      }
#endif

      do_exit = 0;
      Command = NULL;
      user = NULL;
      pass = NULL;
      RenameFrom = NULL;
      s = NULL;
      data_port = 0;
      data_ip_addr = 0;
      data_sock = 0;
      passive_mode = 0;



#ifdef LINUX
      ConnectionNo = 0;
#endif

      /* handle the ftp commands */
      do_exit = 0;
      while (!feof(f) && ((Command == NULL) || (do_exit == 0))) {
   s = ReadString(f);/* get a line */
   if (s == NULL) {
      Command = NULL;
      continue;
   }      /* eof ? */
   splitarg(s, , ,, &Command, &Options);
   upperstring(Command);
   FPRINTF(stderr, „> %s %s\n“, Command, Options);
   fflush(stderr);
   if (!strcmp(Command, „USER“))
    user = cmd_user(f, Command, Options, ConnectionNo, &do_exit);
   else if (!strcmp(Command, „PASS“))
    pass = cmd_pass(f, &user, Command, Options, ConnectionNo);
   else if (!strcmp(Command, „PASV“))
    cmd_pasv(f, &user, &pass, &passive_mode, &data_sock, &host_ip_addr);
   else if (!strcmp(Command, „PORT“))
    cmd_port(f, &user, &pass, &data_ip_addr, &data_port,
                  Command, Options);
   else if (!strcmp(Command, „LIST“))
    cmd_list(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
```

```
                &data_port, Command, Options);
else if (!strcmp(Command, „NLST“))
 cmd_nlst(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                &data_port, Command, Options);
else if (!strcmp(Command, „PWD“))
   cmd_pwd(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „CWD“))
   cmd_cwd(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „CDUP“))
   cmd_cdup(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „MKD“))
   cmd_mkd(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „DELE“))
   cmd_dele(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „RMD“))
   cmd_rmd(f, &user, &pass, Command, Options);
else if (!strcmp(Command, „RNFR“))
   cmd_rnfr(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                  &data_port, Command, Options, &RenameFrom);
else if (!strcmp(Command, „RNTO“))
   cmd_rnto(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                  &data_port, Command, Options, &RenameFrom);
else if (!strcmp(Command, „RETR“))
   cmd_retr(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                  &data_port, Command, Options);
else if (!strcmp(Command, „STOR“))
   cmd_stor(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                  &data_port, Command, Options);
else if (!strcmp(Command, „SIZE“))
   cmd_size(f, &user, &pass, &passive_mode, &data_sock, &data_ip_addr,
                  &data_port, Command, Options);
else if (!strcmp(Command, „TYPE“))
   cmd_type(f, &type, Command, Options);
else if (!strcmp(Command, „SYST“))
   cmd_syst(f);
else if (!strcmp(Command, „NOOP“))
   cmd_noop(f);
else if (!strcmp(Command, „HELP“))
   cmd_help(f);
else if (!strcmp(Command, „QUIT“))
   cmd_quit(f, &do_exit);
else if (!strcmp(Command, „SHUTDOWN“)) {
   fclose(f);
   close(msgsock);
   return;
} else if (!strcmp(Command, „MSAM“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „APPE“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MRSQ“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „ABOR“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „SITE“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „XMKD“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
```

```
} else if (!strcmp(Command, „XCUP“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „ACCT“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MLFL“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MRCP“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „STOU“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „SMNT“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „STRU“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MAIL“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „ALLO“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „STAT“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „XRMD“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „REIN“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MODE“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MSND“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „REST“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „XCWD“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MDTM“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „MSOM“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else if (!strcmp(Command, „XPWD“)) {
   FPRINTF(f, „502 %s command not implemented.\r\n“, Command);
   FPRINTF(stderr, „502 %s command not implemented.\n“, Command);
} else {
   FPRINTF(f, „500 ‚%s %s‘: command not understood.\r\n“, Command,
               Options);
   FPRINTF(stderr, „500 ‚%s %s‘: command not understood.\n“, Command,
               Options);
}
```

```
      };


      FPRINTF(stderr, „Closing connection..\n“);
      fclose(f);
      FPRINTF(stderr, „.\n“);
      close(msgsock);
      FPRINTF(stderr, „Done.\n“);

      /* Don‘t forget to free memory */
      if (user)
    free(user);
      if (pass)
    free(pass);
      if (s)
    free(s);
      if (Command)
    free(Command);
      if (Options)
    free(Options);

#ifdef VXWORKS
      UnRegisterConnection();
#endif

#ifdef FORK
      exit(0);
#endif

#ifdef VXWORKS
      taskDelete(0);/* happy suicide :-) */
#endif
   }
}




/* ------------------------------------------------------------- */

/*
   „main task“
   ==================

   create a socket and wait for connections.
   if a client connects, spawn a new task / fork a new process to handle it

 */



#ifdef VXWORKS
int ftpd_main()
#endif
#ifdef LINUX
int main()
#endif
{
   unsigned int sock;
   unsigned int i;
```

```
   struct sockaddr_in server;
   unsigned int port = 0;
   unsigned int length, result;
   FILE *logfile;
   fd_set ready;
#ifdef LINUX
   char logname[256];
#endif
#ifdef VXWORKS
   char TaskName[256];
   int connection_number = 0;
   int TaskID;
#endif

   signal(SIGTERM, term);/* catch SIGTERM and SIGINT signal */
   signal(SIGINT, term);/* signal handler is term() */

   /* initialize the global array ‚connection‘ */

   for (i = 0; i < MAX_CONNECTIONS; i++) {
      connection[i].TaskId = (int) NULL;
      connection[i].virtual_cwd = NULL;
   }



   /* create a socket */

   sock = socket(AF_INET, SOCK_STREAM, 0);
   if (sock < 0) {
      PERROR(„opening socket“);
      return -1;
   }

   /* Name Socket using wildcards */
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY;

   port = 21;
   server.sin_port = htons(port);
   FPRINTF(stderr, „Trying to bind to port #%d\n“, port);
   if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
    FPRINTF(stderr, „Can not bind to port %d (%s)\n“, port, STRERROR(errno));

      /* Try to bind to other port */
      server.sin_port = htons(0);
      if (bind(sock, (struct sockaddr *) &server, sizeof(server))) {
    PERROR(„Can not bind“);
    return -1;
      }
   }

   /* Find out assigned port number and print it out */
   length = sizeof(server);
   if (getsockname(sock, (struct sockaddr *) &server, &length)) {
      PERROR(„getting socket name“);
      return -1;
   }
#ifdef FORK
   FPRINTF(stderr, „Fork enabled\n“);
#endif

#ifdef NOFORK
```

```
   FPRINTF(stderr, „Fork disabled\n“);
#endif

   port = ntohs(server.sin_port);
   FPRINTF(stderr, „localaddr: 0x%x\n“, (unsigned int) port);
   FPRINTF(stderr, „localport: #%d\n“, ntohs(server.sin_port));


#ifdef LINUX
   sprintf(logname, „ftpd-%d.log“, (int) ntohs(server.sin_port));
   logfile = fopen(logname, „w“);
#endif

#ifdef VXWORKS
   logfile = stderr;
#endif


   /* start accepting connections */

   do {
      if (listen(sock, 5))
    PERROR(„listen“);
      else
    FPRINTF(stderr, „listen ok\n“);


      FPRINTF(stderr, „main task: waiting for connection\n“);
      FD_ZERO(&ready);
      FD_SET(sock, &ready);
      result = select(sock + 1, &ready, NULL, NULL, NULL);
      if (result == -1) {
    PERROR(„error executing select“);
    continue;
      }



#ifdef FORK
      pid = fork();
      if (pid == 0)
    fnord(&sock, &logfile); /* create a new process and call the
                                    connection-handler */
#endif

#ifdef NOFORK
#ifdef LINUX
      pid = 0;
      fnord(&sock, &logfile);/* call the connection handler  */
#endif
#ifdef VXWORKS
      sprintf(TaskName, „Connection %d“, connection_number);

      TaskID = taskSpawn(TaskName, 100, 0, 65536, (FUNCPTR) fnord,
               (int) &sock, (int) &logfile, 0, 0, 0, 0, 0, 0, 0, 0);

      if (TaskID == ERROR) {
    FPRINTF(stderr, „Can not spawn task\n“);
    return -1;
      } else {
    FPRINTF(stderr, „New Task spawned. TaskID = %x\n“, TaskID);
      }
      connection_number++;
```

```
#endif
#endif


#ifdef VXWORKS

      taskDelay(500);
#endif
   }
   while (1);


}

/*---------------------------------------------------------------------------

                                FTP Server
                               ============
 Datei: file.c


---------------------------------------------------------------------------*/


/*
   Useful routines for directory browsing
   ======================================

 */


/*
   Realpath
   ---------

   returns a canonical absolute pathname
   The Realpath() functiuon derives, from the pathname pointed to by
   „filename“, an absolute pathname that names the same file, without
   „.“, „..“ or „//“.
   The absolute pathname is stored in the buffer pointed to bye
   „resolved_path“.

   Note: This implementation of ‚Realpath‘ does not resolve symbolic
   links and does not check if the path exists.
   Use the C-Library Implementation of ‚Realpath‘ if you
   want to compile with Linux.
 */

/*
   support ‚virtual directories‘
   The ‚virtual_cwd‘ variable is set by the CWD command;

   virtual_cwd is global - there are only one virtual current work directory
   for all tasks. This „doesn‘t matter“, because the ‚real‘ cwd of the VxWorks
   DosFS is global...

 */



int FPRINTF(FILE * f, const char *fmt,...)
{
```

```
   int ret;
   va_list ap;

   if ((f != stderr) || (FTPD_DEBUG)) {
      va_start(ap, fmt);
      ret = vfprintf(f, fmt, ap);
      va_end(ap);
      return ret;
   } else
      return 0;
}

void PERROR(char *s)
{
   if (FTPD_DEBUG)
      perror(s);

}

#ifdef VXWORKS
char *GETCWD(char *buf, int max_buf_length)
{
   int TaskId, i;

   TaskId = taskIdSelf();
   for (i = 0;(i < MAX_CONNECTIONS) && (connection[i].TaskId != TaskId); i++);
   if (i < MAX_CONNECTIONS)
      strcpy(buf, connection[i].virtual_cwd);
   else {
      FPRINTF(stderr, „No path entry found. Reset path to ,/‘\n“);
      strcpy(buf, „/“);
   }
   return buf;
}
#endif

#ifdef LINUX
char *GETCWD(char *buf, int max_buf_length)
{
   return (getcwd(buf, max_buf_length));
}
#endif

#ifdef VXWORKS
void SETCWD(char *path)
{
   int TaskId, i;

   TaskId = taskIdSelf();
   for (i = 0;(i < MAX_CONNECTIONS) && (connection[i].TaskId != TaskId); i++);
   if (i < MAX_CONNECTIONS)
      strcpy(connection[i].virtual_cwd, path);
   else
      FPRINTF(stderr, „Can not set current workdirectory\n“);
}
#endif

#ifdef LINUX
void SETCWD(char *path)
{
   return;
}
#endif
```

```
char *Realpath(char *file_name, char *resolved_path)
{
   char buf[4096];
   int margc, i, j, l;
   char **margv;
   char cwd[1024];

   if (file_name[0] != ,/‘) {/* relative path to current workdirectory */
      GETCWD(cwd, 1024);
      strcpy(buf, cwd);
      if (strcmp(cwd, „/“))
    strcat(buf, „/“);
      strcat(buf, file_name);
   } else
      strcpy(buf, file_name);

   if (buf[strlen(buf) - 1] != ,/‘)
      strcat(buf, „/“);

   parsearg(buf, ,/‘, &margc, &margv);


   for (i = 0; i < margc; i++)
      if (!strcmp(margv[i], „.“)) {/*free(margv[i]); */
    margv[i] = NULL;
      } else if (!strcmp(margv[i], „..“)) {
    /*free(margv[i]); */ margv[i] = NULL;
    for (j = i; j > 0; j--)
       if (margv[j - 1]) {/*free(margv[j-1]); */
          margv[j - 1] = NULL;
          j = 0;
       }
      }
   strcpy(resolved_path, „/“);
   for (i = 0; i < margc; i++)
      if (margv[i]) {
    strcat(resolved_path, margv[i]);
    strcat(resolved_path, „/“);
      }
   l = strlen(resolved_path);
   if ((l > 1) && (resolved_path[l - 1] == ,/‘))
      resolved_path[l - 1] = 0x00;

   return resolved_path;

}




/*
   Get user/group/other permissions in a human readable format
 */


char *GetPermissionString(int permbits, int flag)
{
   char *s;

   s = (char *) malloc(5);
   strcpy(s, „---“);
```

```
   if (s == NULL) {
      PERROR(„malloc“);
      return NULL;
   }
   if ((flag == SUID) || (flag == SGID)) {
      if (permbits & 0x01)
    s[2] = ‚s‘;
      else
    s[2] = ‚S‘;
   } else if (flag == STICKY) {
      if (permbits & 0x01)
    s[2] = ‚t‘;
      else
    s[2] = ‚T‘;
   } else {
      if (permbits & 0x01)
    s[2] = ‚x‘;
   }

   if (permbits & 0x04)
      s[0] = ‚r‘;
   if (permbits & 0x02)
      s[1] = ‚w‘;

   return s;
}



/*
   Explore the files type
 */

char *GetTypeString(int type)
{
   char *tc;

   tc = (char *) malloc(2);
   if (tc == NULL) {
      PERROR(„malloc“);
      return NULL;
   }
   strcpy(tc, „ „);
   switch (type) {
   case 0x14:
      tc[0] = ‚s‘;
      break;/*   socket           */
   case 0x12:
      tc[0] = ‚l‘;
      break;/*   symlink          */
   case 0x10:
      tc[0] = ‚-‘;
      break;/*   reg. file        */
   case 0x06:
      tc[0] = ‚b‘;
      break;/*   blockdevice      */
   case 0x04:
      tc[0] = ‚d‘;
      break;/*   directory        */
   case 0x02:
      tc[0] = ‚c‘;
      break;/*   characterdevice  */
   case 0x01:
```

```
      tc[0] = ‚p‘;
      break;/*   fifo             */
   default:
      tc[0] = ‚-‘;
   }
   return tc;
}



/*
   Convert the number of the mounth to its name.
 */

char *GetMounthString(int mounth)
{
   char *s;

   s = (char *) malloc(4);
   if (s == NULL) {
      PERROR(„malloc“);
      return NULL;
   }
   switch (mounth) {
   case 0:
      strcpy(s, „Jan“);
      break;
   case 1:
      strcpy(s, „Feb“);
      break;
   case 2:
      strcpy(s, „Mar“);
      break;
   case 3:
      strcpy(s, „Apr“);
      break;
   case 4:
      strcpy(s, „May“);
      break;
   case 5:
      strcpy(s, „Jun“);
      break;
   case 6:
      strcpy(s, „Jul“);
      break;
   case 7:
      strcpy(s, „Aug“);
      break;
   case 8:
      strcpy(s, „Sep“);
      break;
   case 9:
      strcpy(s, „Oct“);
      break;
   case 10:
      strcpy(s, „Nov“);
      break;
   case 11:
      strcpy(s, „Dec“);
      break;
   default:
      strcpy(s, „Foo“);
      break;
```

```
   }

   return s;
}


/*
   Get an unix like list of a file.

   It consists of the files permissions, number of symbolic links,
   owner, group, size, modification date, and name.

   The permission field consists of 10 characters:
   The first character may be one of the following:

   d   the entry is a directory.
   l   the entry is a symbolic link.
   b   the entry is a block special file.
   c   the entry is a character special file.
   p   the entry is a fifo (or „named pipe“) special file.
   -    the entry is an ordinary file.
   p   the entry is a FIFO.
   s   the entry is a socket.

   The next 9 characters are interpreted as three sets of three
   bits each.  The first set refers to the owner‘s permissions;
   the next to permissions of others in the user-group  of  the
   file;  and  the  last  to  all others.  Within each set, the
   three characters indicate permission to read, to write,  and
   to  execute  the  file  as  a  program, respectively.  For a
   directory, `execute‘‘ permission  is  interpreted  to  mean
   permission  to  search  the  directory for a specified file.

   Example:

   drwxr-xr-x     1    -2   -2         512  3  9 1999 DEV1:/HOT_PICS


   See your favorite unix user manual for details.
 */


char *GetUnixlikeEntry(char *dirname, char *basename)
{
   struct stat buf;
   struct tm *modtime;
   int flagbits, typebits, u, g, o;
   char *EntryString;
   char *user, *group, *other, *type;
   int suid, sgid, sticky;
   char *Filename, *temp;
   int l;

   if (strcmp(dirname, „“)) {
      temp = (char *) malloc(strlen(dirname) + 1);
      if (temp == NULL) {
    PERROR(„malloc“);
    return NULL;
      }
      strcpy(temp, dirname);
      for (l = strlen(temp) - 1; temp[l] == ,/‘; l--)
    temp[l] = 0x00;
      Filename = (char *) malloc(strlen(temp) + strlen(basename) + 2);
```

```
   if (Filename == NULL) {
 PERROR(„malloc“);
 return NULL;
   }
   strcpy(Filename, „“);
   strcat(Filename, temp);
   strcat(Filename, „/“);
   strcat(Filename, basename);
   free(temp);
} else {
   Filename = (char *) malloc(strlen(basename) + 1);
   if (Filename == NULL) {
 PERROR(„malloc“);
 return NULL;
   }
   strcpy(Filename, basename);
}


if (stat(Filename, &buf) == -1) {
   PERROR(„STAT FAILED“);
   return NULL;/* error */
}
flagbits = (buf.st_mode & 0x0e00) >> 9;/* suid, sgid, sticky */
typebits = (buf.st_mode & 0xf000) >> 12;/* type of file */

sticky = flagbits & 0x01;
sgid = flagbits & 0x02;
suid = flagbits & 0x04;

u = (buf.st_mode & 0x01c0) >> 6;
user = GetPermissionString(u, suid);
g = (buf.st_mode & 0x0038) >> 3;
group = GetPermissionString(g, sgid);
o = (buf.st_mode & 0x0007);
other = GetPermissionString(o, sticky);
type = GetTypeString(typebits);

/*
   example:

   buf.st_mode equals 0x000081ff:

   000081ff = 1000   000 111 111 111
   XXXX   UGS

   U..suid, G.sgid, S..sticky

   XXXX: 0x14 .. socket     06 .. blockdevice       01 .. fifo
   0x12 .. symlink    04 .. directory
   0x10 .. reg. file  02 .. characterdevice
 */

modtime = (struct tm *) localtime(&buf.st_mtime);

l = strlen(basename);
EntryString = (char *) malloc(l + 100);
if (EntryString == NULL) {
   PERROR(„malloc“);
   return NULL;
}
sprintf(EntryString, „%s%s%s%s %3d %8s %8s %8d %s %2d %5d %s“,
   type,
```

```
      user,
      group,
      other,
      (int) buf.st_nlink,
      GetUserName(buf.st_uid),
      GetGroupName(buf.st_gid),
      (int) buf.st_size,
      (char *) GetMounthString(modtime->tm_mon),
      (int) modtime->tm_mday,
      (int) modtime->tm_year + 1900,
      basename);


   if (user)
      free(user);
   if (group)
      free(group);
   if (other)
      free(other);
   if (type)
      free(type);


   if (Filename)
      free(Filename);
   return EntryString;
}



/*************************************************************************/



void do_list(FILE * data, char *path, int verbose)
{
   DIR *directory;
   struct dirent *entry = NULL;
   char *EntryString = NULL;
   int error;

   /*
     Required list format:

     FPRINTF(data, „total 4\r\n“);
     FPRINTF(data, „drwxr-xr-x   4 root   root  1024 Feb 24 20:41 .\r\n“);
     FPRINTF(data, „drwx------  19 root   root  1024 Feb 24 20:40 ..\r\n“);
     FPRINTF(data, „drwxr-xr-x   1 root   root  1024 Feb 23 23:12 run\r\n“);
     FPRINTF(data, „drwxr-xr-x   1 root   root  1024 Feb 23  1999 store\r\n“);
     return;
    */


   EntryString = GetUnixlikeEntry(„“, path);

   if (EntryString == NULL) {
      FPRINTF(stderr, „GetUnixLikeEntry: failed\n“);
      return;/* list nothing */
   }
   if (EntryString[0] == ‚d‘) {/* entry is a directory */
      FPRINTF(data, „total 0\r\n“);
      FPRINTF(stderr, „total 0\n“);
```

```
      directory = opendir(path);
      if (directory == NULL) {
    FPRINTF(stderr, „Can not open directory ,%s‘\n“, path);
    return;/* on any error: list nothing */
      }
      do {
    entry = readdir(directory);
    if (entry) {


       /* do not list special directories here */
       if (whitelist(path) && blacklist2(entry->d_name))
          continue;
       if (blacklist(path))
          continue;

       EntryString = GetUnixlikeEntry(path, entry->d_name);
       if (EntryString) {
          if (verbose) {
        FPRINTF(data, „%s\r\n“, EntryString);
        FPRINTF(stderr, „%s\r\n“, EntryString);
          } else {
        FPRINTF(data, „%s\r\n“, entry->d_name);
        FPRINTF(stderr, „%s\r\n“, entry->d_name);
          }
       } else
          FPRINTF(stderr, „EntryString = NULL.\n“);
    } else {
       error = errno;
       FPRINTF(stderr, „entry=NULL. (%s)\n“, STRERROR(error));
    }
      }
      while (entry);

      closedir(directory);
   } else {
      /* do not list special directories here */
      if (blacklist(path))
    return;/* do not list special directories here */
      if (verbose)
    FPRINTF(data, „%s\r\n“, EntryString);
      else
    FPRINTF(data, „%s\r\n“, entry->d_name);
   }

}

/*-------------------------------------------------------------------------

                                FTP Server
                               ============
 Datei: alice.socket.c (Netzwerk Routinen)

-------------------------------------------------------------------------*/


#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#ifdef LINUX
#include <netdb.h>
#endif

#ifdef VXWORKS
#include „vxWorks.h“
#include „sockLib.h“
#include „inetLib.h“
#include „taskLib.h“
#include „ioLib.h“
#include „fioLib.h“
#include „semLib.h“
#include „iv.h“
#include „intLib.h“
#include „sysLib.h“
#include „wdLib.h“
#include „kernelLib.h“
#endif

#include <stdio.h>
#include <string.h>

#ifdef VXWORKS

int ReadIPAddr(char *ip_addr)
{
   int margc;
   char **margv;
   int a, b, c, d;
   char *error;

   parsearg(ip_addr, ,.‘, &margc, &margv);
   if (margc != 4)
      return 0;
   a = strtol(margv[0], &error, 10);
   if (*error)
      return 0;
   b = strtol(margv[0], &error, 10);
   if (*error)
      return 0;
   c = strtol(margv[0], &error, 10);
   if (*error)
      return 0;
   d = strtol(margv[0], &error, 10);
   if (*error)
      return 0;

   return (a << 24) + (b << 16) + (c << 8) + d;
}

 /* converts an ip address (dotted notation) to a network host entry */

struct hostent *gethostbyname(const char *name)
{
   return NULL;
}

#endif

void upperstring(char *in)
{
   int i, l = strlen(in);
   for (i = 0; i < l; i++)
```

```
      if ((in[i] > ,`‘) && (in[i] < ,{,))
    in[i] &= 223;
}


int connect_to(char *domain, int port)
{
   int sock;
   FILE *Logfile = stderr;
   struct sockaddr_in server;
#ifdef LINUX
   struct hostent *hp;
#endif
   struct hostent *gethostbyname();
#ifdef VXWORKS
   int netorder_ip_addr;
#endif

   /* Create socket on which to send. */
   sock = socket(AF_INET, SOCK_STREAM, 0);
   if (sock < 0) {
      PERROR(„error opening stream socket“);
      exit(-1);
   }
   /* Construct name, with no wildcards, [...] */
   server.sin_family = AF_INET;

#ifdef LINUX
   hp = gethostbyname(domain);
   if (hp == 0) {
      FPRINTF(Logfile, „%s: unknown host“, domain);
      return (-2);
   }
   bcopy(hp->h_addr, &server.sin_addr, hp->h_length);
#endif


   /* convert internet hostaddress from the numbers and not notation into
      binary data in network byte order
    */

#ifdef VXWORKS
   netorder_ip_addr = inet_addr(domain);
   if (netorder_ip_addr == -1)
      FPRINTF(Logfile,
               „%s: unknown host (only ip-addresses supported)“, domain);
   bcopy((void *) &netorder_ip_addr, (void *) &server.sin_addr, 4);
#endif


   server.sin_port = htons(port);

   /* Connect to server */
   FPRINTF(Logfile, „trying %s\n“, domain);
   if (connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0) {
      PERROR(„error connecting stream socket“);
      return (-1);
   } else
      FPRINTF(Logfile, „connected to %s.\n“, domain);

   return sock;
}
```

```
/*-------------------------------------------------------------------------

                                FTP Server
                               ============
 Datei: convert.c

-------------------------------------------------------------------------*/

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define NEW_LINE   0
#define CONT_LINE  1
#define SKIP_LINE  2

#define UNKNOWN    0
#define XILINX_BIN 1
#define XILINX_RBT 2
#define XILINX_MCS 3


struct RBTStreamInfo {
   char *decoded_data;
   int decoded_bytes;/* number of decoded bytes */
   int last_state;
   char last_byte;
   long last_bit_no;
};


struct MCSStreamInfo {
   char *decoded_data;
   int decoded_bytes;/* number of decoded bytes */
   int last_state;
};

struct RawStreamInfo {
   char *decoded_data;
   int decoded_bytes;/* number of decoded bytes */
};


struct StreamInfo {
   int format;
   struct RawStreamInfo *raw_stream;
   struct RBTStreamInfo *rbt_stream;
   struct MCSStreamInfo *mcs_stream;
};



void parsearg(char *string, char d, int *argc, char ***argv);



char *AppendData(char *buf, char *source, int l, int *count)
{
   char *newbuf;

   /* If size is equal to zero, the call is equivalent to free(ptr) and
      it will do something nasty. Reject it .. */
```

```
   if ((l + *count) == 0)
      return buf;

   newbuf = (char *) realloc(buf, l + *count);
   if (newbuf == NULL) {
      PERROR(„error realloc“);
      return NULL;
   }
   bcopy(source, newbuf + *count, l);
   *count = *count + l;

   return newbuf;
}


int FlushStream(struct StreamInfo *stream, int fd)
{
   int wsize = 0;

   switch (stream->format) {

   case XILINX_RBT:
      wsize = write(fd, stream->rbt_stream->decoded_data,
                    stream->rbt_stream->decoded_bytes);
      stream->rbt_stream->decoded_data = NULL;
      stream->rbt_stream->decoded_bytes = 0;
      break;

   case XILINX_MCS:
      wsize = write(fd, stream->mcs_stream->decoded_data,
                    stream->mcs_stream->decoded_bytes);
      stream->mcs_stream->decoded_data = NULL;
      stream->mcs_stream->decoded_bytes = 0;
      break;

   case XILINX_BIN:
      wsize = write(fd, stream->raw_stream->decoded_data,
              stream->raw_stream->decoded_bytes);
      stream->raw_stream->decoded_data = NULL;
      stream->raw_stream->decoded_bytes = 0;
      break;
   }

   return wsize;
}



int DecodeRBT(char *rbt, int rbtsize, struct RBTStreamInfo *rbt_stream)
{
   char maske[] =
   {128, 64, 32, 16, 8, 4, 2, 1};
   unsigned long int i, offset = 0;
   unsigned char byte;
   unsigned int bit_no;
   char *temp;
   int state;


   state = rbt_stream->last_state;
   bit_no = rbt_stream->last_bit_no;
   byte = rbt_stream->last_byte;
```

```
    offset = 0;
    temp = (char *) malloc(rbtsize / 8);

    for (i = 0; i < rbtsize; i++) {

       switch (state) {

       case NEW_LINE:
     if (rbt[i] == 0x0a)
        state = NEW_LINE;
     else if ((rbt[i] == ,1‘) || (rbt[i] == ,0‘))
        state = CONT_LINE;
     else
        state = SKIP_LINE;
     break;

       case CONT_LINE:
     if (rbt[i] == 0x0a)
        state = NEW_LINE;
     else if ((rbt[i] == ,1‘) || (rbt[i] == ,0‘))
        state = CONT_LINE;
     else
        state = SKIP_LINE;
     break;

       case SKIP_LINE:
     if (rbt[i] == 0x0a)
        state = NEW_LINE;
     else
        state = SKIP_LINE;
     break;

       }


       if (state == CONT_LINE) {
     if (rbt[i] == ,1‘) {
        byte |= maske[7 - (bit_no % 8)];
        bit_no++;
     }
     if (rbt[i] == ,0‘) {
        byte &= ~maske[7 - (bit_no % 8)];
        bit_no++;
     }
     if ((bit_no % 8) == 0) {
        temp[offset] = byte;
        offset++;
     }
       }
    }       /* end of loop */

    rbt_stream->last_bit_no = bit_no;
    rbt_stream->last_byte = byte;
    rbt_stream->last_state = state;

    rbt_stream->decoded_data = AppendData(rbt_stream->decoded_data, temp,
                               offset, &(rbt_stream->decoded_bytes));
    if (temp)
       free(temp);
    return rbt_stream->decoded_bytes;


}
```

```
/*
   MCS Support not implemented
 */

int DecodeMCS(char *mcs, int mcssize, struct MCSStreamInfo *mcs_stream)
{
   return 0;
}



/*
   ReadFrom Source attemps to read up to ‚max‘ bytes from file descriptor
   fd into the buffer starting at buf.

   On success, the number of bytes read is returned. (zero indicates
   end of file)
   If this number is smaller than the number of bytes requested, fewer
   bytes are available - maybe we were close to end-of-file.
 */


int ReadFromSource(int fd, char *buf, int min)
{
   int count = 0;
   int rsize;
   char *temp;

   memset(buf, 0, min);
   temp = (char *) malloc(min);
   do {
      rsize = read(fd, temp, min - count);
      if (rsize > 0) {
    bcopy(temp, buf + count, rsize);
    count += rsize;
      }
   }
   while ((rsize > 0) && (count < min));

   if (temp)
      free(temp);
   return count;


   /*return read(fd, buf, min); */

}


/*
   Checks for XilinxBitstream format.

   Returned values:

   1 .. data in array pointed to by buf are probabley a xilinx bitstream
   0 .. no xilinx bitstream

   --------

   Xilinx bitstream format (header):
```

```
   11111111  0010       xxxx xxxxxxxx xxxxxxxx xxxx 1111        0
   fillbyte  preamble          24 bit count         fillbits    startbit

   Bitserial data is read from left to right, with the first bit
   in each byte is assigned to D0 (LSB) !

   Byte parallel data is effectively assembled from this bitstream:

   ff X4 XX XX fX
   ^^
   = 0010xxxx
   ^
   databit 0

 */

int XilinxBitStream(unsigned char *buf)
{

   unsigned char fillbits0;
   unsigned char fillbits1;
   unsigned char preamble;
   unsigned char startbit;

   fillbits0 = buf[0];
   preamble = (buf[1] & 0x0f);
   fillbits1 = (buf[4] & 0xf0) >> 4;
   startbit = (buf[5] & 0x01);

   if ((fillbits0 == 0xff) && (preamble == 0x04) && (fillbits1 == 0x0f) &&
       (startbit == 0x00))
      return 1;
   else
      return 0;

}







int DetectFileFormat(int fd, struct StreamInfo *stream)
{

   char buf[256];
   int rsize, format, l;

   format = UNKNOWN;

   rsize = ReadFromSource(fd, buf, 6);

   stream->raw_stream->decoded_data = buf;
   stream->raw_stream->decoded_bytes = rsize;

   if ((rsize > 5) && (XilinxBitStream(buf)))
      format = XILINX_BIN;

   do {
      /* try to decode rbt */
      if (format == UNKNOWN) {
    l = DecodeRBT(buf, rsize, stream->rbt_stream);
```

```
     if ((l > 5) && XilinxBitStream(stream->rbt_stream->decoded_data))
        format = XILINX_RBT;
       }
       /* try to decode mcs (not implemented) */
       if (format == UNKNOWN) {
     l = DecodeMCS(buf, rsize, stream->mcs_stream);
     if ((l > 5) && XilinxBitStream(stream->mcs_stream->decoded_data))
        format = XILINX_MCS;
       }
       if (format == UNKNOWN) {
     rsize = ReadFromSource(fd, buf, 6);
       }
    }
    while (
        (rsize > 0) &&
        (stream->rbt_stream->decoded_bytes < 6) &&
        (stream->mcs_stream->decoded_bytes < 6) &&
        (format == UNKNOWN)
         );


    stream->format = format;
    return format;

}


struct StreamInfo *InitStreamInfoStruct()
{
    struct StreamInfo *stream;
    struct RBTStreamInfo *rbt_stream;
    struct RawStreamInfo *raw_stream;
    struct MCSStreamInfo *mcs_stream;

    rbt_stream = (struct RBTStreamInfo *) malloc(sizeof(struct RBTStreamInfo));
    if (rbt_stream == NULL)
       return NULL;
    rbt_stream->decoded_data = NULL;
    rbt_stream->decoded_bytes = 0;
    rbt_stream->last_bit_no = 0;
    rbt_stream->last_byte = 0;
    rbt_stream->last_state = NEW_LINE;

    mcs_stream = (struct MCSStreamInfo *) malloc(sizeof(struct MCSStreamInfo));
    if (mcs_stream == NULL)
       return NULL;
    mcs_stream->decoded_data = NULL;
    mcs_stream->decoded_bytes = 0;

    raw_stream = (struct RawStreamInfo *) malloc(sizeof(struct RBTStreamInfo));
    if (raw_stream == NULL)
       return NULL;
    raw_stream->decoded_data = NULL;
    raw_stream->decoded_bytes = 0;

    stream = (struct StreamInfo *) malloc(sizeof(struct StreamInfo));
    if (stream == NULL)
       return NULL;
    stream->format = UNKNOWN;
    stream->rbt_stream = rbt_stream;
    stream->mcs_stream = mcs_stream;
    stream->raw_stream = raw_stream;
```

```
   return stream;
}


int LoadFPGA(int fd, int gd)
{
   struct StreamInfo *streaminfo;
   int format, rsize, wsize;
   char buf[4096];


   streaminfo = InitStreamInfoStruct();
   if (streaminfo == NULL) {
      PERROR(„Init StreamInfo“);
      return UNKNOWN;
   }
   switch ((format = DetectFileFormat(fd, streaminfo))) {
   case XILINX_BIN:
      FPRINTF(stderr, „Xilinx Bitstream Format\n“);
      break;
   case XILINX_RBT:
      FPRINTF(stderr, „Xilinx Bitstream (RBT encoded)\n“);
      break;
   case XILINX_MCS:
      FPRINTF(stderr, „Xilinx Bitstream (MCS encoded)\n“);
      break;
   default:
      FPRINTF(stderr, „Unknown file format\n“);
   }

   wsize = FlushStream(streaminfo, gd);

   /* copy */
   if (format == XILINX_BIN) {
      do {
    rsize = read(fd, buf, sizeof(buf));
    if (rsize > 0)
       wsize = write(gd, buf, rsize);
      }
      while ((rsize > 0) && (wsize == rsize));
   }
   if (format == XILINX_RBT) {
      do {
    rsize = ReadFromSource(fd, buf, sizeof(buf));
    if (rsize > 0) {
       DecodeRBT(buf, rsize, streaminfo->rbt_stream);
       wsize = FlushStream(streaminfo, gd);
    }
      }
      while (rsize > 0);
      FPRINTF(stderr, „\n“);
   }
   if (streaminfo)
      free(streaminfo);

   return format;
}


#ifdef TEST
void main(int argc, char **argv)
{
   FILE *f, *g;
```

```
   int fd, gd;

   if (argc < 3) {
      FPRINTF(stderr, „usage: %s <source> <dest>\n“, argv[0]);
      return;
   }
   f = fopen(argv[1], „r“);
   if (f == NULL) {
      PERROR(„Can not open file“);
      return;
   }
   fd = fileno(f);

   g = fopen(argv[2], „w“);
   if (g == NULL) {
      PERROR(„Can not open file“);
      return;
   }
   gd = fileno(g);

   LoadFPGA(fd, gd);

   if (f)
      fclose(f);
   if (g)
      fclose(g);

}
#endif

/*---------------------------------------------------------------------------

                                FTP Server
                               ============
 Datei: ftpd.h

---------------------------------------------------------------------------*/


/* do not print messages to stderr; #define FTPD_DEBUG 1  to enable */
#define FTPD_DEBUG 0

#define MAX_CONNECTIONS 8

/* allow anonymous logins;  #define ANONYMOUS 0 to disable */
#define ANONYMOUS 1
#define LISTEN_PORT 21

#define SPECIALROOT   „/fpga“
#define SPECIALDIRS   „/fpga/4010:/fpga/4036:/fpga/4010XL“
#define SPECIALFILES  „4010:4036:4010XL“

/* upload and move operations to ‚4010‘ directory will be routed to
/* No devices defined for ‚4036‘ and ‚4010XL‘ (use spaces)        */
#define DEVICENAMES   „/dev/fpga0: : „  /dev/fpga0 */

/* comma sep. list of userid and username */
#define UID „0:root,-2:fert,500:fert“

/* comma sep. list of groupid:groupname */
#define GID „0:root,-2:bastards“

/* enable ftp server */
```

```
#define INCLUDE_FFTPD                   /* enable ftp server */


/*
 SPECIALROOT, SPECIALDIR, SPECIALFILES:
   if you upload or move (rename) a file to a ‚special directory‘, the file
   will be written to /dev/fpga0  (defined in DEVICENAMES)

   SPECIALROOT is the parent directory, SPECIALFILES are the names of the
   special directories.
*/


/*****************************************************************************/



struct Connection {
                    int TaskId;
                    char *virtual_cwd;
                  };


struct Connection connection[MAX_CONNECTIONS];

#define FPGA_BUSY (M_ioLib | 20)

#define SHORT 0
#define LONG  1

#define SUID 4
#define SGID 2
#define STICKY 1


#ifdef LINUX
 #define STRERROR(x) (strerror(x))
 #define TRUE 0
 #define ERROR 0xffffff
#endif

#ifdef VXWORKS
 #define STRERROR(x) (rstrerror(x))
#endif

#ifdef LINUX
 #include <netdb.h>
 #include <sys/time.h>
 #include <sys/param.h>
 #include <sys/stat.h>
 #include <sys/types.h>
 #include <fcntl.h>
 #include <unistd.h>
 #include <sys/stat.h>
#endif

#ifdef VXWORKS
 #include „vxWorks.h“
 #include „sockLib.h“
 #include „sockFunc.h“
 #include „inetLib.h“
 #include „selectLib.h“
 #include „taskLib.h“
```

```
 #include „ioLib.h“
 #include „fioLib.h“
 #include „semLib.h“
 #include „iv.h“
 #include „intLib.h“
 #include „sysLib.h“
 #include „wdLib.h“
 #include „kernelLib.h“
 #include <blkIo.h>
 #include <DosFsLib.h>
 #include <usrlib.h>
#endif

 #include <sys/types.h>
 #include <sys/socket.h>
 #include <arpa/inet.h>
 #include <netinet/in.h>
 #include <unistd.h>
 #include <stdarg.h>
 #include <stdlib.h>
 #include <stdio.h>
 #include <signal.h>
 #include <string.h>
 #include <dirent.h>
 #include <time.h>
 #include <errno.h>


#define PORT 21 /* listen port */
#define WELCOME „220 Ferts FTP server ready.“

#ifndef MAXPATHLEN
 #define MAXPATHLEN 1024
#endif


#ifdef VXWORKS
 struct hostent {
  char *h_name;                 /* official name of host  */
  char **h_aliases;             /* Alias list. */
  int h_addrtype;               /* Host address type. */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of address. */
  #define h_addr h_addr_list[0]  /* address, for backward compatibility */
 };
#endif

void upperstring(char *in);
void splitarg(char *string, char d, char **Command, char **Options);
void parsearg( char *string, char d, int *argc, char ***argv);
void do_list(FILE *f, char *path, int verbose);
char *GetUnixlikeEntry(char *dirname, char *basename);
void free_args( int *margc, char ***margv );
int blacklist( char *path);
int blacklist2( char *path);
int whitelist( char *path);
char *Realpath( char *filename, char *buf );
int MakeDataConnection(FILE *f, int *passive_mode, int *data_sock,
                       int *data_ip_addr, int *data_port, int *sock_out,
                       FILE **data, char *mode);
char *GetUserName(int);
char *GetGroupName(int);
int FPRINTF(FILE *f, const char *fmt,...);
```

```
void PERROR(char *s);

#ifdef VXWORKS
 char *rstrerror(int errorcode);
#endif
```

```
/*-----------------------------------------------------------------------

                                FTP Server
                               ============
 Datei: Makefile

-----------------------------------------------------------------------*/


#
#                                FTP Server
#                               ============
# Datei: Makefile
#
#*


LIB_PATH = C:/tornado / target / lib
CC = ccppc
OPTIONS = -DCPU = PPC403 \
-DVXWORKS \
-DNOFORK \
-BC:\Tornado / host / x86 - win32 / lib / gcc - lib / \
-mcpu = 403 - g - mstrict - align - ansi - nostdinc - O2 \
-fvolatile - fno - builtin - fno - for -scope
   \
       -Wall - I / h - I. \
 -IC: \Tornado \ target \ config \ all \
 -IC: \Tornado \ target / h \
 -IC: \Tornado \ target / src / config \
 -IC: \Tornado \ target / src / drv



 all: ftpd.o

 ftpd.i:ftpd.c
       $(CC) - c $(OPTIONS) - E - <ftpd.c > ftpd.i

 ftpd.o:ftpd.c
       $(CC) - c $(OPTIONS) ftpd.c

 install:ftpd.o
       arppc - r $(LIB_PATH) / libPPC403gnuvx.a ftpd.o

 clean:
   rm - f ftpd.o
   rm - f ftpd.i
```