Diplomarbeit

Entwicklung und Realisierung der FiberLink-Testplatine

Michael Klein

Universität des Saarlandes
Fachrichtung 6.2 – Informatik
Lehrstuhl für Rechnerarchitektur und Parallelrechner
Prof. Dr. W. J. Paul

April 2001

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt habe. Ich habe diese Arbeit keinem anderen Prüfungsamt vorgelegt.

Saarbrücken, im April 2001

Michael Klein

Danksagung

Ich möchte den folgenden Personen danken, ohne die die Erstellung dieser Arbeit nicht möglich gewesen wäre:

- Professor Paul für die Vergabe dieses interessanten Themas,
- meiner Mutter für das Ermöglichen meines Studiums,
- meiner Freundin Silke Meiers für vollkommenes Verständnis in den schwierigen Zeiten,
- Jochen Preiß für die gute Zusammenarbeit in dem Projekt,
- Professor Opower, Markus Scholl, Günther Renz und Stefan Scharl für die Zusammenarbeit und die Lösung einiger Probleme elektrischer Natur,
- Peter Bach, Michael Bosch, Cedric Lichtenau und Jörg Fischer für die Hilfe bei allen anfänglichen Problemen,
- Herrn Dellmer von IBM Research Germany für das kostenlose Überlassen von 10 PowerPC-Prozessoren,
- Jochen Preiß für das Korrekturlesen dieser Arbeit,
- Peter Bach für die Kontrolle der Schematics
- und dem gesamten Lehrstuhl und insbesondere dem Labor in seinen wechselnden Besetzungen für die gute Atmosphäre.

Murphys laws of hardware engineering:

1. Complex hardware tends to produce complex errors.
2. Simple hardware tends to produce complex errors.
3. Optimized hardware tends to produce more sophisticated errors.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren hat die Geschwindigkeit von Computerhardware in rasantem Maße zugenommen. Während vor 20 Jahren Hochleistungsrechner noch mit Taktraten von unter einem MHz arbeiteten sind heute sogar in Heim-PCs aus dem Supermarkt Prozessorgeschwindigkeiten von 1 GHz und mehr gebräuchlich [3]. Gleichzeitig sind die Preise für diese PCs dank eben jener Supermärkte auf ein erschwingliches Niveau gefallen.

Neben den wachsenden Anforderungen an die Leistungsfähigkeit dieser Systeme sind insbesondere auch die zu verarbeitenden Datenmengen stark angestiegen. Aus diesem Grund werden nicht nur schnelle Prozessoren, sondern auch Methoden zur schnellen Datenübertragung immer wichtiger.

Heutige Systeme übertragen Daten im Normalfall über elektrische Leitungen, teilweise als Freiluftverkabelung, meistens jedoch in Form einer Leiterplatte. Diese elektrische Datenübermittlung erweist sich aber immer mehr als begrenzender Faktor bei der erreichbaren Gesamtleistung des Systems. Schon ab 100 MHz Taktfrequenz und wenigen Zentimetern Leitungslänge treten bei der Übertragung über elektrischen Leitungen physikalische Seiteneffekte auf, die für den Entwickler nur sehr schwer kontrollierbar sind.

Glasfaserverbindungen versprechen hingegen potentielle Übertragungsgeschwindigkeiten im Bereich von mehreren GHz. Durch die Entwicklung von sehr dichten Glasfaserarrays, neuartigen Steckverbindern und in Platinen eingelassenen optischen Übertragungslagen erschließt sich hier ein enormes Potential zur Steigerung der Datenübertragungsraten. Die Technologie dahinter steckt jedoch noch in den Kinderschuhen.

Bislang werden Glasfaserübertragungswege vor allem im Langstreckenbereich eingesetzt (ATM [1], Gigabit-Ethernet [31]). Für kurze Strecken (insbesondere innerhalb eines einzelnen Rechners) bot sich bisher die Übertragung über optische Leitungen aufgrund folgender Aspekte nicht an:

Bei den momentan existierenden optischen Verbindungen werden die Signale

in externen Übersetzerbausteinen von elektrisch nach optisch und umgekehrt umgewandelt. Die Probleme der elektrischen Übertragung werden nicht gelöst, sondern lediglich auf das kürzere Stück zwischen Chip und Umsetzer reduziert. Dies macht jedoch innerhalb eines Einzelrechners keinen Sinn.

Die Umwandlung der Signale "on Chip" scheiterte bisher hauptsächlich an der Größe der Sender- und Empfängerbauteile. Inzwischen gibt es jedoch Laser- und Receiverbauteile, die hinreichend klein sind, um sie gemeinsam mit der Chiplogik in einem Gehäuse unterbringen zu können.

Im Rahmen eines Projektes der Universität des Saarlandes und des Deutschen Zentrums für Luft- und Raumfahrt wird ein Prototyp eines solchen Chips entwickelt. Dabei handelt es sich um einen kombinierten Cache- und Speicherchip mit integrierter optischer Anbindung. Diese Arbeit befaßt sich mit dem Entwurf und der Realisierung einer Prozessorplatine, auf der später dieser Prototyp eingesetzt und getestet werden kann.

Die zum Verständnis der Arbeit nötigen Grundlagen werden in Kapitel 2 gelegt. In Kapitel 3 wird näher auf das Projekt eingegangen, die neuen Ansätze und Technologien vorgestellt und einen Überblick über die verschiedenen Teilgebiete des Projekts gegeben. In Kapitel 4 wird auf die Überlegungen eingegangen, die der tatsächlichen Implementierung der Platine vorausgingen. Außerdem werden die theoretischen Grundlagen für das Design gelegt. Kapitel 5 beschäftigt sich mit der praktischen Implementierung und den dabei aufgetretenen Problemen. Im abschließenden Kapitel 6 wird ein Statusbericht des Projektes abgegeben, die Arbeit noch einmal zusammengefaßt und auf die nächsten Schritte eingangen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen erörtert, die zum Verständnis der folgenden Kapitel notwendig sind. Weiterhin werden Begriffe eingeführt, die im Verlauf der Arbeit öfter verwendet werden.

In Abschnitt 2.1 werden die nötigen Grundlagen und das benutzte Vokabular aus dem Bereich der Hardware erläutert. Auf das grundlegende Konzept von Caches wird in Abschnitt 2.2 eingegangen. Im Abschnitt 2.3 schließlich werden einige Richtlinien zum Platinendesign aufgeführt, die helfen sollen, Implementierungsentscheidungen zu begründen.

2.1 Hardwaregrundlagen

Um im weiteren Verlauf dieser Arbeit kurz aber denoch präzise argumentieren zu können, ist es nötig, die wichtigsten benutzten Fachbegriffe zu erläutern sowie die verwendete Notation einzuführen.

2.1.1 Benutzte Begriffe

Ein *Signal* ist eine Leitung, deren Zustand zu jedem Zeitpunkt mit einem der logischen Werte 0, 1 oder *undefiniert* identifiziert werden kann. Im Falle einer elektrischen Leitung geschieht dies durch Bestimmen der anliegenden Spannung U und ihre Einordnung in einer Tabelle. Diese Tabelle ist von der verwendeten Technologie abhängig (siehe Abbildung 2.1, [17]). Der Fall *undefiniert* tritt im Normalfall nur kurzzeitig beim Umschaltvorgang zwischen 0 und 1 oder umgekehrt auf. Betrachtet man eine optische Leitung, so definiert man "Licht an" als 1 und "Licht aus" als 0.

Ein Signal wird von maximal einer Quelle "getrieben". Dies kann ein elektronisches Bauteil sein, das eine Spannung an die Leitung anlegt oder ein Laser, der einen Lichtpuls in eine optische Leitung abgibt. Von der Quelle aus betrachtet bezeichnen wir das Signal als *Ausgangssignal*.
Ein Signal seinerseits kann einen oder mehrere Eingänge an anderen Bausteinen "treiben", das heißt, diese Bausteine analysieren das anliegende Signal und

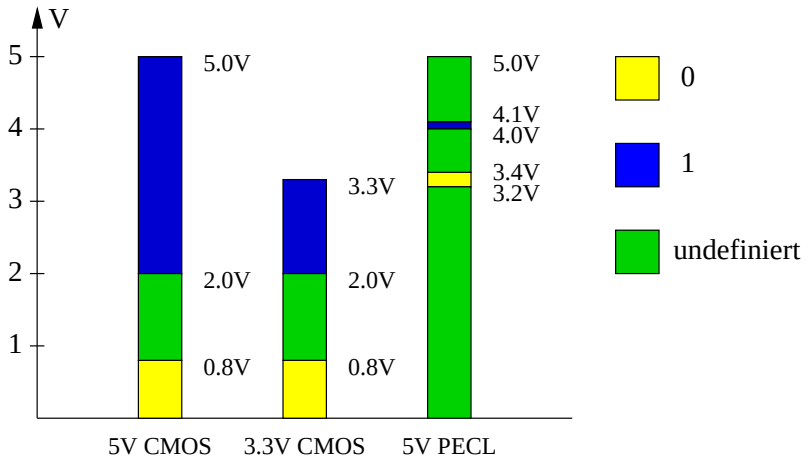

Abbildung 2.1: Zustandstabellen verschiedener Technologien

reagieren darauf. Wird das Signal von einem solchen Bauteil aus betrachtet, so bezeichnen wir es als *Eingangssignal*. Ein Signal, welches vom gleichen Bauteil sowohl als Ausgangssignal als auch als Eingangssignal genutzt werden kann, bezeichnen wir als *bidirektional*. Liegt ein Signal am Clock-Eingang eines Bausteins an und wird dementsprechend zur Taktung dieses Bausteins benutzt, so bezeichnen wir es als *Clock-Signal*.

Eine *Schaltung* ist eine Menge von miteinander verbundenen Bauteilen, wobei die Ausgangssignale der einzelnen Bauteile jeweils mit einem oder mehreren Eingangssignalen des gleichen oder anderer Bauteile zusammengeschlossen sind. Die Interaktion der Bauteile kann zum Beispiel in Form eines Diagramms verdeutlicht werden. Gebräuchlich sind dabei insbesondere 2 Arten von Diagrammen.

In einem *exakten Timing-Diagramm* beispielsweise werden die Veränderungen der Spannung der elektrischen Signale im Verlauf der Zeit aufgetragen (siehe Abbildung 2.2). Das *physikalische* Verhalten der beteiligten Bauteile bestimmt dabei das Aussehen des Diagramms.

In einem ersten Vereinfachungsschritt werden meist die Spannungen der Signale mit ihrem logischen Wert identifiziert. Desweiteren wird der Umschaltvorgang ignoriert. Man geht also davon aus, daß Signale nur den Wert 0 oder 1 annehmen und direkt zwischen diesen beiden Zuständen hin und her wechseln, ohne zwischendurch den Wert “undefiniert” anzunehmen. In einem *idealisierten Timingdiagramm* vernachlässigt man weiterhin noch die Verzögerszeiten durch Bauteile (siehe Abbildung 2.3). Das *logische* Verhalten der Bauteile entscheidet das Aussehen eines idealisierten Timing-Diagramms.

Eine exakte Timinganalyse ist nur an Stellen des Designs mit zeitkritischen Pfaden oder zur Bestimmung der maximalen Taktgeschwindigkeit eines Designteils notwendig. Aufgrund der vorher feststehenden niedrigen Taktrate von nur 10

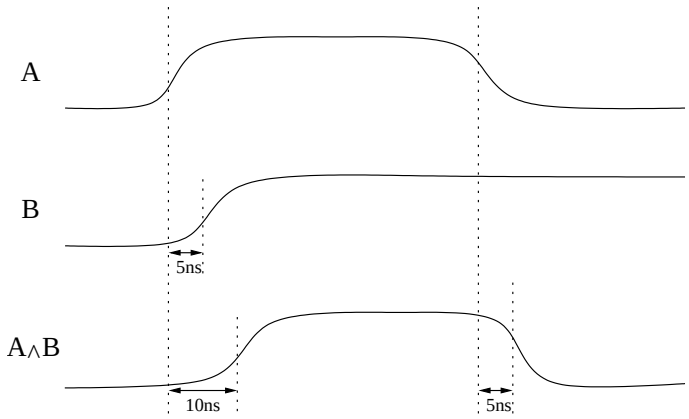

Abbildung 2.2: Exaktes Timing-Diagramm

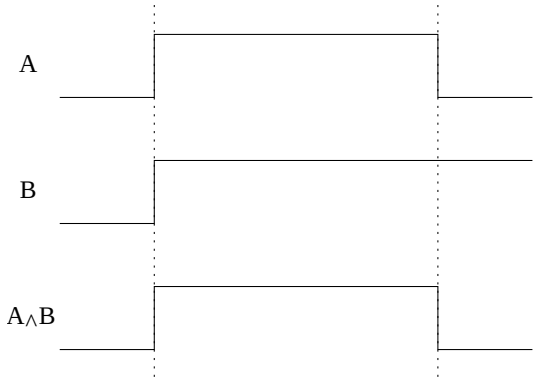

Abbildung 2.3: Entsprechendes idealisiertes Timing-Diagramm

MHz ist dies für die zu beschreibende Platine sehr einfach. Aus diesem Grund werden im Rahmen dieser Arbeit nur idealisierte Timingdiagramme betrachtet.

In Abhängigkeit seines logischen Wertes wird einem Signal einer der Zustände *aktiv* oder *inaktiv* zugewiesen. Wenn dem logischen Wert 0 der Zustand *inaktiv* und dem Wert 1 der Zustand *aktiv* zugewiesen wird, so bezeichnet man das Signal als *active high* oder *high-aktiv*. Im umgekehrten Falle (0 entspricht *aktiv* und 1 entspricht *inaktiv*) spricht man von einem *active low* oder *low-aktivem* Signal.

Ist die Quelle eines elektrischen Signals ausgeschaltet, so verhält sich das Signal elektrisch neutral. Liegt ein solches Signal an einem Bauteileingang an, so ist es im allgemeinen nicht möglich, das Verhalten des jeweiligen Bausteins vorherzusagen. Um dies zu verhindern benutzt man "Pull-Up" - oder "Pull-Down" - Widerstände (siehe Abbildung 2.4).

Treibt Ausgang A auf die Leitung, so fließt der Strom den "Weg des geringsten Widerstandes" direkt zum Eingang B und "ignoriert" den Weg über den Wider-

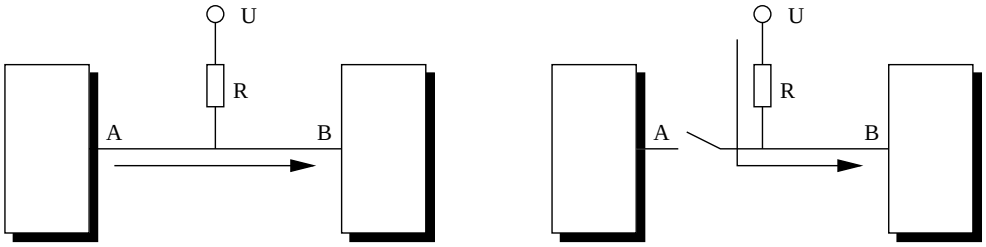

Abbildung 2.4: Pull-Up-Widerstand

stand R. Ist jedoch A elektrisch neutral, so fließt ein Strom von der Spannungsquelle U über den Widerstand R zum Eingang B. Der Eingang wird somit auf den Wert 1 gezogen. Diese Schaltung bezeichnet man als *Pull-Up-Widerstand*. Analog dazu kann man durch Verschalten von R mit der Masse-Lage der Platine den Wert von B auf 0 ziehen. Dies wird als *Pull-Down-Widerstand* bezeichnet.

2.1.2 Notation

Gehören mehrere Signale logisch zusammen, so werden sie zu *Bussen* zusammengefasst. Als abkürzende Schreibweise für die Signalgruppe $X_j, ..., X_i$ für $j > i$ benutzt man die Bezeichnung $X[j : i]$. Wird aus dem Zusammenhang klar, daß der gesammte Bus $X[n-1 : 0]$ gemeint ist, so wird dieser mit X bezeichnet.

Zum Speichern von Signalen werden Register benutzt. Viele Register besitzen neben dem Clock-Eingang zur Taktung des Registers auch noch ein Clock-Enable-Signal, mit dem entschieden werden kann, ob das gerade am Eingang anliegende Signal zur nächsten steigenden Clock-Flanke in das Register geschrieben wird oder nicht. Für das Register R wird dieses Signal mit *Rce* bezeichnet.

Da im Normalfall eindeutig ist, welches Clock-Signal an einem Register angeschlossen ist, wird dieses Signal in Zeichnungen und Beschreibungen meist weggelassen. In Abbildung 2.5 sind die Symbole für Register und Register mit Clock-Enable aufgeführt.

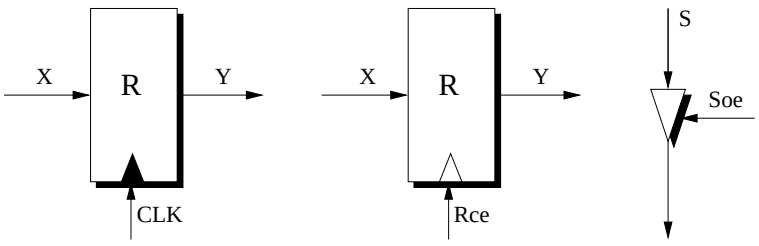

Abbildung 2.5: Symbole für Register ohne und mit Clock-Enable und Treiber mit Output-Enable

Einige Bauteile besitzen abschaltbare Ausgänge. Dies ist vorteilhaft, um daran angeschlossene Bauteile isoliert zu betrachten. Das Kontrollsignal, mit dem sich das Signal S an seiner Quelle abschalten läßt, wird mit *Soe* (S Output Enable) bezeichnet (siehe Abbildung 2.5).

2.2 Das Konzept des Cachings

In diesem Abschnitt wird erläutert, was ein Cache ist und wieso Caches in modernen Rechnersystemen eingesetzt werden. Dazu wird das sogenannte Lokalitätsprinzip vorgestellt und erklärt.

Zur Bewältigung großer Datenmengen muß der Prozessor möglichst *schnell* auf einen möglichst *großen* Hauptspeicher zugreifen können. Hauptspeicher in den heute üblichen Größenordungen zwischen 32MB und 1GB wird jedoch aus Kostengründen nur in der langsamen DRAM-Technologie (Dynamic Random Access Memory) angeboten. Dieses wird momentan bis maximal mit 300 MHz betrieben. Speicher in der schnellen SRAM-Technologie (Static RAM) hingegen kann zwar mit Taktraten bis zu 1 GHz und mehr betrieben werden, ist jedoch sehr teuer und wärmeintensiv.

Ein weiteres Problem ist der Overhead, der beim Zugriff auf den Speicher durch die Verarbeitung der Adressen entsteht. Selbst wenn man Hauptspeicher in Größenordnungen von 32MB und mehr aus SRAM aufbauen würde, könnte man diesen Speicher aufgrund des anfallenden Overheads durch große Adressen nicht mehr mit derartig hohen Taktraten betreiben.

Einen Ansatz zur Lösung des beschriebenen Problems kann man aus Untersuchungen des Ablaufverhaltens verschiedener Programme ziehen:

- Wenn auf eine Speicherzelle zugegriffen wird, dann wird im folgenden mit großer Wahrscheinlichkeit auch auf die nachfolgenden Zellen zugegriffen. Dies ist für den Programmspeicher leicht einzusehen (wenn kein Sprung vorliegt wird die im Speicher nachfolgende Instruktion ausgeführt). Dieses Verhalten nennt man *örtliche Lokalität.*

- Wenn eine Speicherzelle gerade gelesen wurde, so wird sie sehr wahrscheinlich in naher Zukunft noch einmal gelesen. Dies findet seine Begründung darin, daß Programme üblicherweise viele Schleifen beinhalten. Erfahrungsgemäß wird in 90% der Programmlaufzeit auf nur 10% des Programmcodes zurückgegriffen [13]. Dies bezeichnet man auch als *zeitliche Lokalität.*

Die aufgeführten Begründungen beziehen sich in diesem Fall nur auf den Programmspeicher, jedoch sind beide Punkte in ähnlichem Maße auch für den Datenspeicher zutreffend.

Zur Lösung der oben aufgeführten Speicherproblematik unter Ausnutzung des

Lokalitätsprinzips setzt man zwischen den großen, langsamen Hauptspeicher und den Prozessor einen kleinen, schnellen Zwischenspeicher. Dieser wird *Cache* genannt. Benötigt der Prozessor Daten, so schaut er im Cache nach, ob die gesuchten Daten schon dort liegen. Wenn ja, so werden die Daten direkt aus dem Cache übernommen, und der Zugriff auf den langsamen Hauptspeicher konnte gespart werden. Wenn nicht, so lädt der Cache die Daten selbständig aus dem Hauptspeicher nach. Dabei müssen bei vollem Cache eventuell andere Daten wieder aus dem Cache entfernt werden.

Um größtmöglichen Nutzen aus dem Lokalitätsprinzip zu ziehen, geht man beim Design von Caches folgendermaßen vor:

- Wenn ein Datum aus dem Hauptspeicher nachgeladen werden muß, so wird direkt eine ganze Gruppe von nebeneinanderliegenden Daten geladen. Diese Gruppe wird *Cache-Line* genannt. Üblicherweise werden zwischen 4 und 16 Datenzellen in einer Line gespeichert [21]. Auf diese Weise wird die örtliche Lokalität ausgenutzt, da die im folgenden vom Prozessor benötigten Daten mit großer Wahrscheinlichkeit schon im Cache vorhanden sind.
- Zur Ausnutzung der zeitlichen Lokalität sorgt man durch verschiedene Maßnahmen (Replacement-Strategien, Mehrwege-Caches,...) dafür, daß die wahrscheinlich in nächster Zeit benötigte Daten weitestmöglich im Cache verbleiben. Hierzu gibt es verschiedene Ansätze, die zum Beispiel in [30] genauer erklärt sind.

Ein Ansatz zur Steigerung der Performance von Caching-Systemen ist es, die Nachladezeit vom Hauptspeicher in den Cache zu verringern. Dies ist einerseits über die Erhöhung der übertragenen Cache-Line-Breite möglich. Eine Erhöhung der Busbreite um ganze Größenordnungen ist jedoch bei elektrischer Übertragung problematisch, da sich eng nebeneinanderliegende elektrische Leitungen gegenseitig stören [17]. Eine weitere Möglichkeit bietet die Erhöhung der Geschwindigkeit, mit der eine Line vom Speicher in den Cache übertragen wird. Auch hier stößt die elektrische Übertragung schon bald an ihre Grenzen. An diesen beiden Punkten findet sich also ein Ansatz zur Performancesteigerung durch den Einsatz optischer Leitungen.

2.3 Richtlinien zum Platinendesign

In diesem Abschnitt werden einige Grundregeln sowohl zum Schaltplanentwurf (Abschnitt 2.3.1) als auch zum Platinenlayout (Abschnitt 2.3.2) aufgeführt, wie sie zum Beispiel in [10] genannt sind. Diese dienen dazu, in den späteren Kapiteln die Vorgehensweise sowie die Wahl der Implementierung zu begründen.

2.3.1 Der Schaltplanentwurf

In diesem Abschnitt werden Richtlinien beschrieben, welche schon beim Entwurf des Schaltplans im Designsystem zu beachten sind.

- Aufbau des Designs in Funktionsgruppen

  Das Konzept der Platine sollte derart aufgebaut sein, daß sich einzelne Funktionsgruppen getrennt testen lassen. Dabei kann es sich sowohl um logisch als auch um später räumlich zusammenhängende Komponenten handeln. Durch diese Technik lassen sich Fehler gezielter finden. Desweiteren können die einzelnen Tests aufeinander aufgebaut werden. Dabei beginnt man üblicherweise mit den wichtigsten Teilen des Designs (z.B der Clockverteilung). Wenn diese erfolgreich getestet wurden, kann man in späteren Tests davon ausgehen, daß sie funktionieren und die Fehler in den neu hinzugenommenen Teilen liegen.

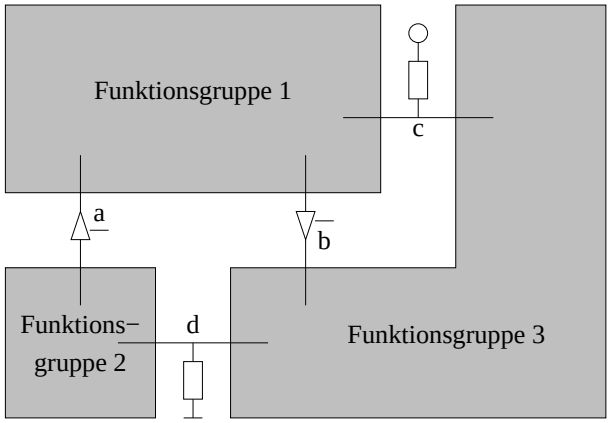

Abbildung 2.6: Schaltplanaufbau mit Funktionsgruppen

Um einen gezielten Funktionsgruppentest zu machen muß man die einzelnen Gruppen voneinander isolieren können. Dazu müssen alle Signale, die zwischen verschiedenen Gruppen verlaufen, abschaltbar sein (a,b). Um die zu testende Gruppe in einen definierten Zustand zu versetzen, kann es desweiteren nötig sein, daß einige Eingangssignale in jedem Fall einen definierten Wert besitzen müssen. Dies läßt sich durch Pull-Up- (c) bzw. Pull-Down-Widerstände (d) erreichen.

- JTAG

  Bei *JTAG* [16], [32] handelt es sich um einen 4 Bit breiten Bus, der das direkte Setzen und Auslesen von Signalen an den Pins der Bausteine erlaubt. Dazu ist jeder Pin des Bausteins mit einem BSC-Register (Boundary Scan Cell) versehen. Diese Register sind seriell in Form eines Shiftregisters miteinander verbunden.

  Über das Eingangssignal TDI (Test Data In) können die BSC-Zellen über den JTAG-Bus mit Daten beliefert werden. Die in den Zellen gespeicherten Daten können über das Signal TDO (Test Data Out) nach außen geschoben und dort ausgelesen werden. Mehrere JTAG-Chip werden miteinander verbunden, indem man jeweils den TDO-Ausgang eines Bausteins mit dem TDI-Eingang des nächsten Bauteils verbindet. So liegen

die BSC-Register aller JTAG-Chips in einem großen seriellen Pfad.

Über die Steuerleitung TMS (Test Mode Select) können die JTAG–Bausteine in verschiedene Modi gesetzt werden (Normalmodus, Testmodus, Selbsttest usw.). Im Normalmodus ist JTAG ausgeschaltet und der Chip arbeitet genauso wie sein Non-JTAG-Pendant.

Im Testmodus werden besondere JTAG-Instruktionen ausgeführt wie zum Beispiel das Sampling der Daten an den Pins oder das Beschreiben der BSCs, indem über den TDI-Eingang Daten in die Register geshiftet werden. Die Instruktionen werden dabei ebenfalls über den TDI-Eingang übertragen.

Über die letzte Leitung (TCK) wird der JTAG-Pfad mit einer eigenen Clock versorgt. Der interne Aufbau eines JTAG-Bausteins wird in Abbildung 2.7 gezeigt.

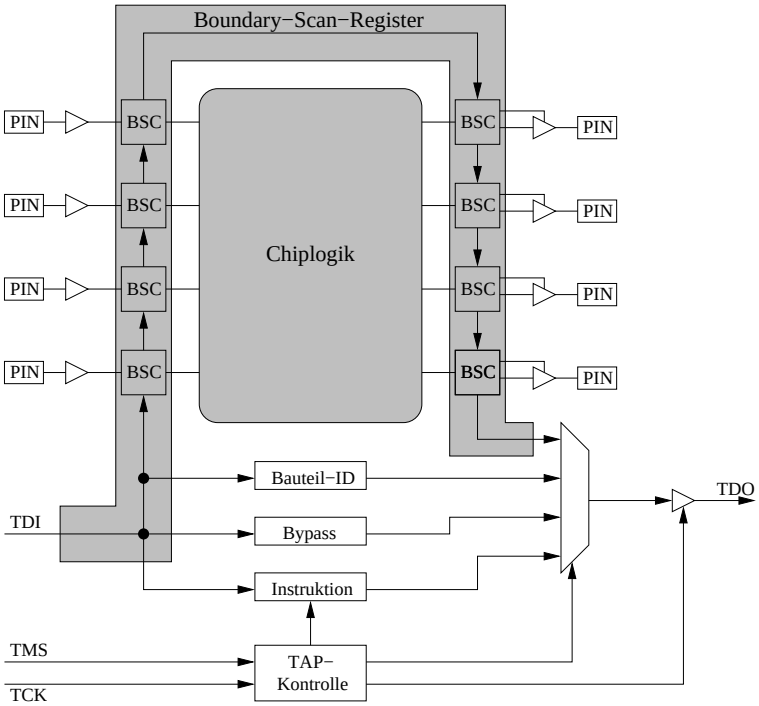

Abbildung 2.7: Aufbau eines JTAG-Bausteins

Durch die Benutzung von JTAG-fähigen Bausteinen ist es möglich, ein *Mustertest-Verfahren* anzuwenden. Dabei werden an eine Menge von Pins (*Stimuli*) ein Signalmuster angelegt und das ankommende Muster an anderen Stellen der Schaltung (*Responses*) ausgelesen. So lassen sich einfach Leitungen zwischen Bauteilen aber auch die Funktionsfähigkeit von Bauteilen oder die Richtigkeit von Bauteilprogrammierungen überprüfen [6].

Damit nicht alle Pins der Schaltung in einem einzigen seriellen Pfad liegen, existieren spezielle Controller-Bausteine, mit denen sich der JTAG-Pfad zu einer Baumstruktur umformen läßt (siehe Abbildung 2.8, [36]).

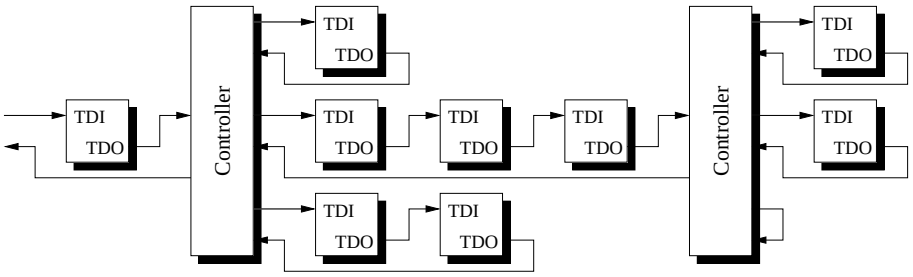

Abbildung 2.8: JTAG-Baumstruktur

In mehreren Fortgeschrittenen-Praktika ([7], [19], [24], [28]) wurde eine komfortable Benutzeroberfläche für JTAG namens *JTAG Deluxe* entwickelt, die unter anderem dem Benutzer die Low-Level-Routinen zur Wahl des JTAG-Pfades durch die Controller und den Aufbau des Teststrings abnimmt.

In Anbetracht der Tatsache, daß sich in den vorangegangen Projekten des Lehrstuhls das JTAG-Mustertestverfahren als das effizienteste Testwerkzeug erwiesen hat [6], sollten wo immer möglich JTAG-Bausteine eingesetzt werden. Da sich dieser Standard in der Industrie durchgesetzt hat, gibt es inzwischen eine breite Palette von JTAG-fähigen Bausteinen für fast alle Einsatzgebiete. Denoch bleibt auch in unserem Fall ein Rest an Bausteinen, die nicht in einer JTAG-Variante erhältlich sind (z.B. die RAM und ROM-Bausteine sowie der komplette ECL-Teil). In diesem Fall sollte man darauf achten, daß trotzdem weitestmöglich alle Ein- und Ausgänge dieser Bausteine per vorgeschaltetem JTAG-Baustein lesend und schreibend erreichbar sind (in Abbildung 2.9 am Beispiel eines RAM-Bausteins mit vorgeschaltetem JTAG-Treiber dargestellt).

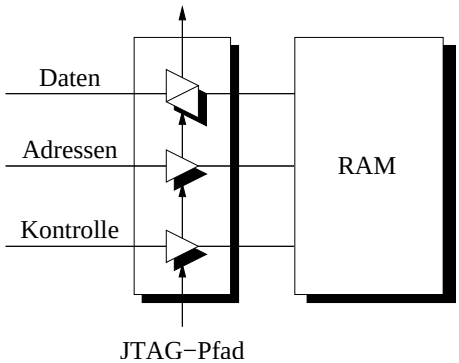

Abbildung 2.9: JTAG-Interface für nicht-JTAG-Baustein

- ISP-Bauteile

  Da beim Entwurf der Kontrollogik im allgemeinen mehr Fehler gemacht werden als beim Entwurf der Datenpfade ist es wünschenswert, die für die Kontrolle verantwortliche Bereiche so flexibel wie möglich zu halten. Ein Ansatz dabei ist die Verwendung von ISP-Bauteilen (In System Programmable). Dies sind frei programmierbare Logikbausteine, die nicht zum Löschen und Neubeschreiben von der Platine entfernt und in einen externen Schreiber eingesetzt werden müssen. Stattdessen werden sie auf der Platine über spezielle Eingangssignale mit den Informationen über ihre neuen Aufgaben versorgt. Dies hat den weiteren Vorteil, daß die mechanische Beanspruchung der Platine minimiert wird.

  Als Schnittstelle zur Reprogrammierung setzt sich immer mehr der oftmals ohnehin vorhandenen JTAG-Bus durch (siehe Abbildung 2.10, [38]). Diese Methode bietet für den Designer den großen Vorteil, daß sie standartisiert ist. Somit läuft die Programmierung aller JTAG-programmierbaren ISP-Bausteine gleich ab. Durch weitestmöglichen Einsatz von JTAG-ISPs können mit nur einer Software und ohne zusätzlichen Hardwareaufwand sehr leicht Änderungen in den kritischen Teilen der Kontrollogik vorgenommen werden.

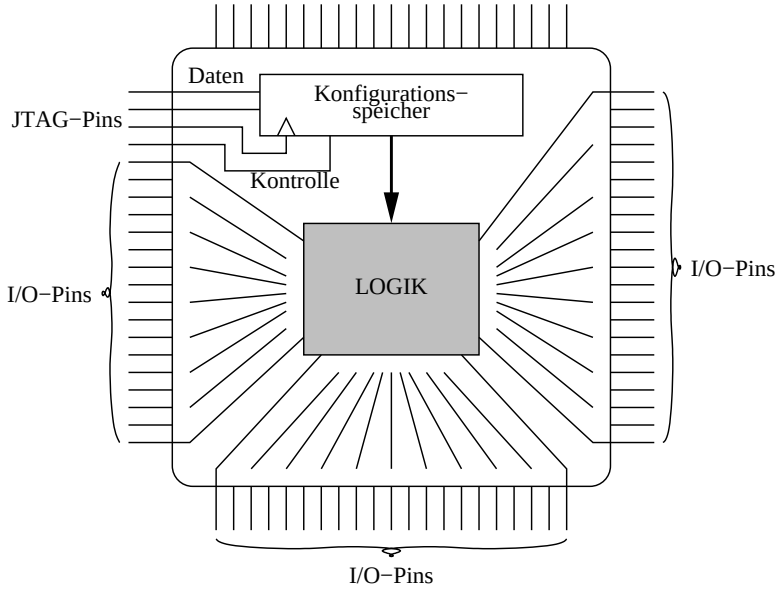

Abbildung 2.10: Baustein-Reprogrammierung über JTAG

2.3.2 Das Platinenlayout

Auch beim späteren Entwurf der Platine selbst (dem sogenannten *Place and Route*) sollten einige Richtlinien beachtet werden.

- SMD-Bauteile

  Nach der Auswahl des zu verwendenden Chips hat man im Normalfall die Entscheidungsfreiheit zwischen SMD-Bauteilen (Surface Mounted Device) oder konventionelle Durchsteck-Bauteilen. Während bei SMD-Bauteilen die Lötstelle auf der Bestückungsseite liegt, werden bei konventionelle Bauteilen die Pins durch sogenannte *Vias* in der Platine hindurchgesteckt und auf der Rückseite verlötet (siehe Abbildung 2.11).

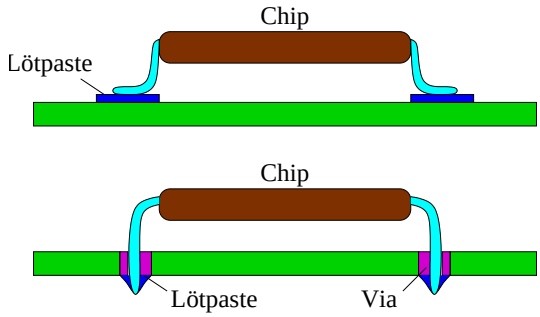

Abbildung 2.11: SMD und konventionelle Bauteile

SMD-Bauteile bieten folgende Vorteile und sollten deshalb immmer dann eingesetzt werden, wenn es eine SMD-Version des Bauteils gibt:

1. Dank der kürzeren Pins stellen SMD-Bauteile eine geringere kapazitative Last dar. Sie verursachen demzufolge weniger Störungen und verringern die Fehleranfälligkeit der Platine gegen elektrische Störungen [17].
2. Bei konventionellen Bausteinen muß man auf allen Platinenlagen mit den zu ziehenden Leitungen den durchgesteckten Pins ausweichen. Dies vergrößert und verkompliziert das Verlegen der Leitungen innerhalb der Platine sehr stark. Bei SMD hingegen genügt es, lediglich auf der Bauteillage die Pins zu umgehen. Auf allen anderen Lagen ist das Routing der Leitungen nicht eingeschränkt.
3. Sie besitzen einen kleineren Pinabstand und somit auch ein kleineres Gehäuse. So kann die Platine verkleinert und Geld gespart werden.
4. SMD ermöglicht die beidseitige Bestückung der Platine mit Bausteinen. Dies kann die Leiterplatte ebenfalls bedeutend verkleinern.

Aus oben genannten Gründen sind inzwischen für praktisch alle Bauteile entsprechende SMD-Versionen verfügbar.

Ein weiterer erkennbarer Trend ist die immer weitere Verkleinerung der Pin-Abstände (*Fine-Pitch-Bauteile*). Diese sind zwar noch platzsparender, stellen aber ein bedeutend höheres Fehlerrisiko dar. Schon geringe örtliche Abweichungen bei der Bestückung oder minimal falsche Dosierung

der Lötpaste führt sehr schnell zu Kurzschlüssen oder Wackelkontakten an den Pins. Die Sichtprüfung und das Anbringen von Testabgriffen erfordern viel Übung und Geschick. Teilweise sind jedoch Fine-Pitch-Bauteile nötig, um Chips mit sehr großer Pinanzahl (der auf unserer Platine eingesetzte Prozessor beispielsweise besitzt 240 Pins) überhaupt noch mit sinnvollen Außenmaßen herstellen zu können.

- Testabgriffe

  Um möglichst schnell und effizient das Verhalten der Platine überprüfen zu können, ist es von Vorteil, die wichtigsten Signal ohne großen Aufwand auslesen zu können. So können beispielsweise Timings der Platine erstellt und mit den geplanten Verhalten der Schaltung verglichen werden. Durch Multilagenplatinen, Fine-Pitch-Bauteile und große Pinmengen wird das gezielte Abgreifen der Signale erschwert. Zum Testen ist es jedoch unerläßlich, daß alle wichtigen Signale von außen erkennbar und erreichbar sind. Dies kann zum Beispiel durch Markierungen an den entsprechenden Pins erfolgt sein. Noch besser ist es, die wichtigsten Signale an einem extra dafür vorgesehenen Testabgriff nach außen zu führen. Dieser kann beispielsweise in Form eines Steckverbinders auf der Platine untergebracht sein.

- Platinenbeschriftung

  Ein wichtiger Schritt bei Test der Platine auf Bestückungs- und Fertigungsfehler ist die Sichtkontrolle. Um die Bausteine auf richtigen Sitz überprüfen zu können, sollte man die Position des Pin 1 auf der Platine deutlich kennzeichnen, so daß sie auch nach Auflöten des Bauteils noch sichtbar ist. Wichtige Signale sollten auf der Platine markiert werden. Auch die Einstellmöglichkeiten durch Jumper und ähnliches sollten kurz erläutert sein. Sind Vertauschungen zweier Anschlüsse möglich, so sollte die Beschriftung besonders auffällig sein (z.B. bei Stromanschlüssen).

Kapitel 3

Das FiberLink-Projekt

In diesem Kapitel soll ein grober Überblick über das Projekt gegeben werden. Dabei werden zunächst die verschiedenen Teilgebiete des Projektes aufgelistet und danach genauer auf das in dieser Arbeit beschriebenen Teilgebiet eingegangen.

3.1 Teilgebiete des Projektes

In Zusammenarbeit mit dem Institut für Technische Physik am Deutschen Zentrum für Luft- und Raumfahrt *DLR* [8] in Stuttgart wird im Rahmen des "FiberLink"-Projektes [22] ein Chip entwickelt, der sowohl einen Datenspeicher in CMOS-Technologie als auch eine schnelle Serialisierer- und Parallelisiererlogik in ECL und die Sender und Empfänger zur direkten Glasfaserübertragung enthält.

Der Prototyp kann entweder als Cache oder als Hauptspeicher betrieben werden. Aus mehreren diesen Chips wird eine optische Breitbandverbindung zwischen Cache und Hauptspeicher realisiert. Der zu bauende Chip wird aus diesem Grund im Folgenden kurz als *OCRAM* (Optical Cache- and RAM) bezeichnen. Andere potentielle Einsatzgebiete für die verwendetet Technologie sind in Kapitel 6 aufgeführt.

Eine kurze Einführung in das Projekt wird auch im Artikel "Optical Interconnect between Cache and Main Memory" in einer der nächsten Ausgaben des physikalischen Journal "Laser Opto" gegeben [20].

Das Projekt gliedert sich in folgende Teilgebiete:

- Am Lehrstuhl von Herrn Prof. Paul wurde von Jochen Preiß die interne Logik des Chips entwickelt [29]. Diese wurde in Form eines ASIC (Aplication Specific Integrated Circuit) implementiert.

- Die Technologie zur direkten Anbindung der optischen Sender- und Empfängereinheiten an den Logik-Die sowie der Aufbau eines Multichip-

Moduls - wie in Abbildung 3.1 angedeutet - wird von Dr. M. Scholl, G. Renz und S. Scharl an der DLR Stuttgart vorgenommen.

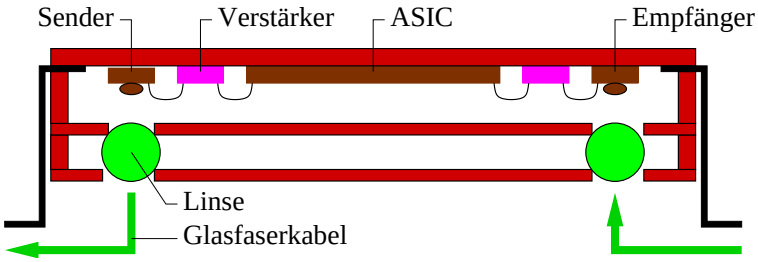

Abbildung 3.1: Schematischer Aufbau des Multichip-Moduls

- Inhalt dieser Diplomarbeit ist die Entwicklung einer Demonstrations- und Testumgebung für den Chip in Form einer lauffähigen Prozessorplatine (Abbildung 3.2) ebenfalls am Lehrstuhl von Herrn Prof. Paul.

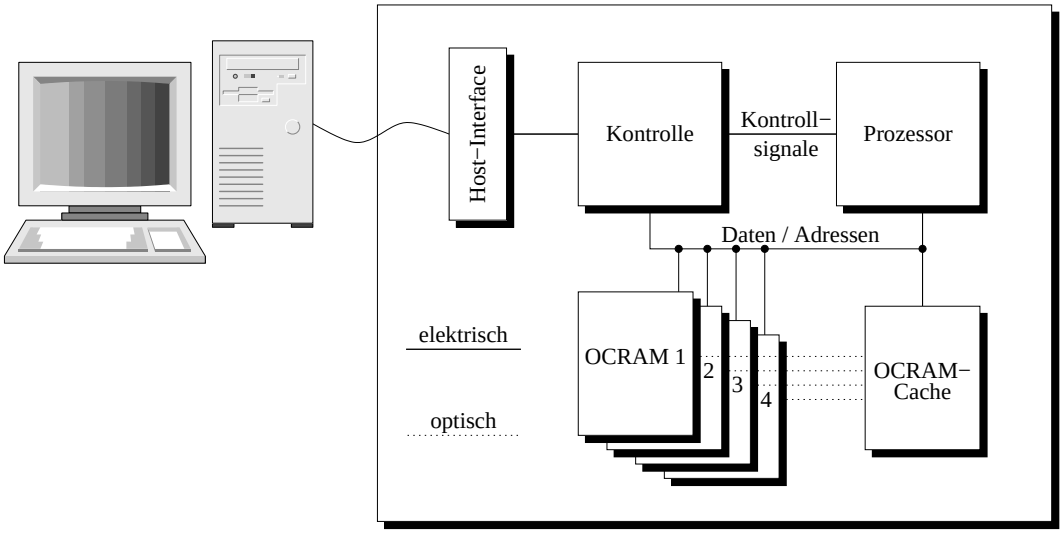

Abbildung 3.2: Grobaufbau der Platine

3.2 Die Testplatine

Auf der gebauten Platine soll ein kommerzieller Prozessor einen der neu entwickelten OCRAM-Chip als Cache benutzen. Dieser ist über Glasfaserkabel optisch mit 4 weiteren OCRAM-Chips verbunden, die als Hauptspeicher fungieren. Benötigt der Prozessor Daten, so fragt er beim Cache-Chip nach. Dieser fordert gegebenenfalls die Daten über die optische Verbindung aus den entsprechenden Speichermodulen an.

Die besondere Anforderung an diese Platine war der Entwurf einer Schaltung, die möglichst leicht zu debuggen, fehlertolerant und rekonfigurierbar sein soll. Da es sich bei dem OCRAM-Chip um einen experimentellen Prototypen handelt, der aufgrund seiner Komplexität enorm fehleranfällig ist, soll die Anzahl der potentiellen Fehlerquellen auf dem Rest der Platine so gering wie möglich gehalten werden. Fehler auf dem Chip sind nur durch ein teures Neudesign zu beseitigen. Daher sollte die Platine den Entwickler in die Lage versetzen, eventuell auf dem Chip vorhandene Fehler außerhalb des Chips beheben oder zumindest umgehen zu können.

Um die Fehlersuche so einfach wie möglich zu halten, wurde zuerst eine Testplatine ohne den neuen Chip entwickelt. Auf dieser soll der ohnehin zur Zeit noch nicht verfügbare OCRAM durch einen Standardspeicherbaustein und einen frei programmierbaren Logikbaustein simuliert werden. Der Rest der Schaltung kann so schon einmal auf Fehler untersucht werden. Später sollen die OCRAM-Chip dann einfach in den auf der Platine vorhandenen Speicherbus hineingehängt werden können (Abbildung 3.3).

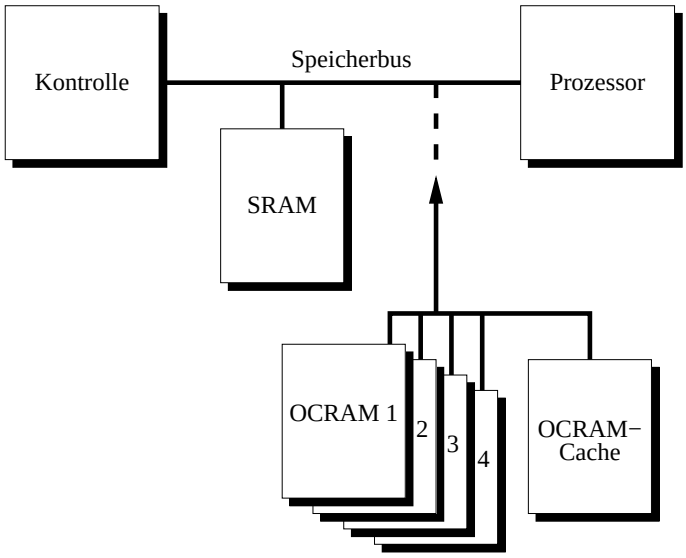

Abbildung 3.3: Speicherbus der Platine

Kapitel 4

Theoretische Vorüberlegungen

In diesem Kapitel werden zunächst die Anforderungen an die zu entwickelnde Platine aufgeführt. Danach wird genauer auf die einzelnen logischen Teile der Platine eingegangen und die Überlegungen erläutert, die für deren Implementierung entscheidend waren.

4.1 Anforderungen an die Platine

Bevor man mit dem Bau einer nichttrivialen Platine beginnt, sollte wie bei jedem größeren Projekt vorher ein Pflichtenheft erstellt werden, in dem folgende Punkte festgehalten werden:

- Minimalanforderungen

  Zu bauen ist eine Prozessorplatine, die eine realistische Test- und Demonstrationsumgebung für den OCRAM-Chip darstellt. Dazu muß man mit einem kommerziellen Prozessor auf den zu bauenden Cache-Chip zugreifen können.

- Wunschanforderungen

  Da es sich um eine experimentelle Umgebung handelt, wäre es wünschenswert, wenn die Schaltung leicht zu debuggen und fehlertolerant wäre. Um viele aussagekräftige Simulationsergebnisse zu erhalten, wäre es gut, wenn man von außen auf möglichst viele Aspekte des Systems wie z.B. Clockgeschwindigkeit, Anzahl der Speicher-Waitstates usw. einwirken könnte.

- Benutzte Schnittstellen

  Da mehrere Personen an diesem Projekt beteiligt sind, muß vorher eine Liste der Schnittstellen erstellt und diese Schnittstellen definiert werden. In unserem Fall ist dies das Interface zwischen Platine und Außenwelt, das Busprotokoll des OCRAM-Chips und die Kommunikation zwischen dem zu bauenden Logik-Die und den Elektro-Optischen Umsetzern.

Die Besonderheiten des Projekts müssen erkannt und festgehalten werden. Implementierungsmöglichkeiten müssen gefunden und gegeneinander abgewogen

werden.

In diesem Projekt standen mehrere solcher Entscheidungen im Mittelpunkt, die im folgenden näher erörtert werden sollen.

4.2 Die Hostanbindung

Im folgenden Abschnitt werden die Überlegungen erläutert, die für die Wahl des Hostinterface ausschlaggebend waren.

4.2.1 Anforderungen an das Interface

Zunächst einmal ist es wichtig, über ein Hostinterface eine Verbindung zwischen der Platine und der Außenwelt herzustellen (siehe Abbildung 4.1). Um für dieses Projekt sinnvoll einsetzbar zu sein, muß das Interface die folgenden Mindestanforderungen erfüllen:

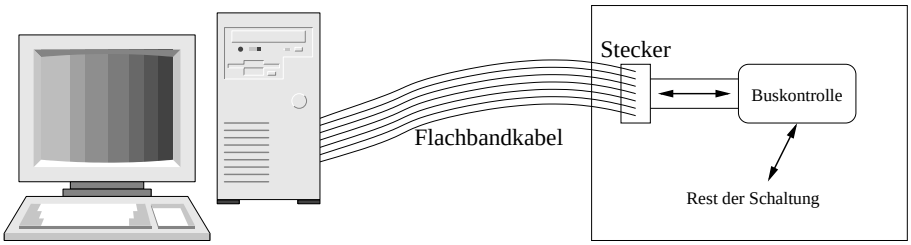

Abbildung 4.1: Verbindung zum Hostrechner

- Schreiben und Lesen des lokalen Speichers

  Da die Anbindung von Peripherie wie Tastatur, Maus oder Bildschirm viel zu aufwendig und fehleranfällig für das Projekt gewesen wäre, muß eine einfache Möglichkeit gefunden werden, den Prozessor mit Programmen und Daten zu versorgen und die Resultate zu überprüfen. Dies soll in unserem Fall durch direktes Setzen und Auslesen des Programm- und Datenspeichers geschehen (Abbildung 4.2).

  Zum Testen der Cache-Speicher-Anbindung ist es vorteilhaft, wenn man von außen sowohl auf den Cache als auch auf den Hauptspeicher zugreifen kann. Auf diese Art kann der Übertragungsweg über die optischen Leitungen unabhängig vom Rest der Platine getestet werden. Aus diesem Grund wurden dem Cache und dem Speicher jeweils eigene unabhängige Speicherbereiche zugeordnet. Diese sind direkt über das Hostinterface ansprechbar.

- JTAG-Unterstützung zur Fehlersuche und ISP-Programmierung

  Da zum Platinendebugging das JTAG-Mustertestverfahren eingesetzt werden soll, ist es nötig, die Platine mit einem JTAG-Anschluß zu versehen.

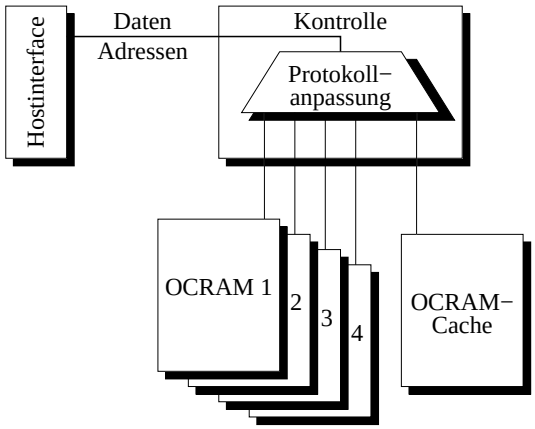

Abbildung 4.2: Speicherzugriff über das Host-Interface

Zur weiteren Senkung die Reparaturzeit und -kosten sollen die Bausteine, die zur Implementierung der fehleranfälligen Protokollsteuerungen zwischen Host, Prozessor und den verschiedenen Speichermodulen benutzt werden, frei programmierbar sein. An programmierbaren Logikbausteinen gibt es einerseits ASICs oder Anti-Fuse-Bausteinen (Bausteine, die über das gezielte Durchbrennen von Sicherungen programmiert werden, wie z.B. FPGAs der Firma ACTEL [2]), die nur einmal beschrieben werden können. Vorteilhaft sind jedoch Bausteine, die einfach, schnell und beliebig oft beschreibbar sind. Hierzu bieten sich, wie bereits erwähnt, JTAG-ISPs an. Im Projekt wurden solche Logikbausteine der Firma XILINX verwendet [37].

- Verschiedene Kontrollsignale wie Reset und Interrupts

  Zur Kommunikation zwischen Host und Platine benötigt man neben den Daten- und Adressleitungen auch noch zusätzliche Kontrollsignale. Wichtig sind unter anderem Reset-Signale, um das Board in einen definierten Zustand setzten zu können, sowie Interrupt-Leitungen zur Kommunikation zwischen Prozessor und Hostrechner (Abbildung 4.3).

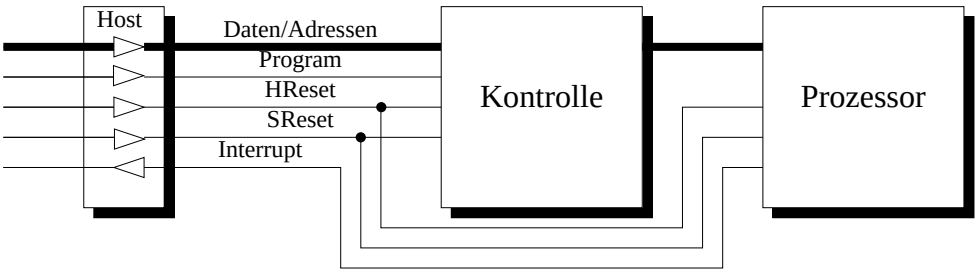

Abbildung 4.3: Kontrollsignale

4.2.2 Mögliche Alternativen

Zur Implementierung eines Hostinterface kamen mehrere Möglichkeiten in Frage. Zum einen kann man auf Standardinterfaces wie zum Beispiel die in IBM-kompatiblen PCs benutzten Protokolle *ISA* oder *PCI* [25] zurückgreifen. Es gibt fertige Bausteine wie den PLX PCI-9060 [27], die das komplette Busprotokoll übernehmen.

Allerdings sind bei diesen Fertiglösungen weder JTAG- noch irgendwelche frei benutzbare Leitungen vorgesehen, so daß diese über einen zusätzlichen Anschluss gelegt werden müßten. Außerdem bieten die vielfältigen Möglichkeiten, die der PLX dem Designer bietet und die für dieses Projekt irrelevant sind, auch Raum für potentielle Fehlerquellen.

Am unserem Lehrstuhl wurde jedoch zur Ansteuerung des experimentellen Rechners *SB-PRAM* ein spezielles Hostinterface entwickelt, welches genau die oben genannten Anforderungen erfüllt. Dabei handelt es sich um einen 100-poligen Standardsteckverbinder, über den folgende Signale übertragen werden:

- 32 gemultiplexte Adress- und Datenleitungen
- 4 Interface-spezifische Kontrolleitungen
- 4 frei belegbare Kontrolllleitungen, davon 3 zur Platine hin und 1 zurück
- 4 JTAG-Leitungen zum Debuggen und Programmieren der ISPs
- 56 Ground-Leitungen

Die endgültige Entscheidung für das *SB-PRAM*-Interface wurde aus folgenden Gründen getroffen:

- Bei einem experimentellen Prototypen sollte die Anzahl der potentiellen Fehlerquellen so gering wie möglich gehalten werden, um ein gezieltes Debugging der wirklichen Neuerungen zu erlauben. Das *SB-PRAM*-Interface ist getestet und funktioniert.
- Selbst wenn Fehler im Hostinterface auftreten sollten, so sind dessen Entwickler immer erreichbar und ansprechbar. Dies kann sich bei der Fehlersuche als vorteilhaft herausstellen.
- Zum *SB-PRAM*-Interface existiert die komfortable Benutzeroberfläche *PRAMOS*. Die ebenfalls fehleranfällige Arbeit der Programmierung eines Software-Interface kann somit eingespart werden.

Auf das genaue Busprotokoll des *SB-PRAM*-Interface wird in Kapitel 5 eingegangen.

4.3 Der Prozessor

Die Wahl des zu verwendenden Prozessors war eine der wichtigsten Entscheidungen des Designs, denn diese Wahl bestimmte das Busprotokoll, mit dem die OCRAM-Chips betrieben werden sollten.
Wichtig für die Entscheidung waren hauptsächlich folgende Kriterien:

- Das Busprotokoll des Prozessors muß komplett offengelegt sein, um Platine und OCRAM-Chip überhaupt entwerfen zu können.
- Der interne Cache des Prozessors sollte ausschaltbar sein, um eine sinnvolle Cache-Simulation durch das OCRAM zu ermöglichen.
- Der Prozessor sollte JTAG-fähig sein, da dies das Debugging der kritischen Prozessor-Umgebung stark vereinfacht.
- Es sollte sich um einen Standardprozessor handeln, der in der Welt anerkannt und verbreitet ist, um den gewonnenen Ergebnissen zusätzliches Gewicht zu verleihen.

Nach sehr vielen Telefonaten mit verschiedenen Hardwareherstellern, Distributoren, Groß- und Einzelhändlern gelangte ich schließlich zur Entwicklungsabteilung von IBM Deutschland, die uns freundlicherweise 10 Exemplare ihres mit 100 MHz getakteten PowerPC 603E [15] unentgeltlich als wissenschaftliche Muster zur Verfügung stellten. Da diese Prozessoren alle gestellten Anforderungen zur vollsten Zufriedenheit erfüllen, wurden sie in diesem Projekt eingesetzt. Auf die genauen Eigenschaften dieser CPUs werde ich dabei im Kapitel 5.2 näher eingehen.

An dieser Stelle möchte ich mich noch einmal bei Herrn Dellmer von IBM Research Germany für diese Spende recht herzlich bedanken. Ohne ihn wäre ich wohl heute noch am Telefonieren.

4.4 Buskontrolle und Busarbitrierung

Auf der Platine werden mehrere verschiedene Busprotokolle benutzt. Am wichtigsten dabei ist das durch den Prozessor vorgegebene Protokoll, das in einer vereinfachten Version auch im OCRAM-Chip implementiert wurde.

Da aus Kostengründen der auf dem OCRAM-Chip zur Verfügung stehende Speicherbereich sehr klein ist, muß zur Unterbringung größerer Programme und Datenmengen weiterer Speicherplatz vorgesehen werden. Aus diesem Grund wird neben dem OCRAM auch noch ein 4MB großer SRAM-Baustein auf der Platine eingesetzt. Dieser benutzt jedoch ein anderes Busprotokoll als der OCRAM-Chip. Desweiteren entspricht auch das Busprotokoll des Host-Interface nicht dem des Prozessors.

Aus diesen Gründen mußten auf der Platine Bausteine vorgesehen werden, die

die Angleichung zwischen den verschiedenen benutzten Protokolle übernehmen. Auch die Arbitrierung des gemeinsamen Adress- und Datenbusses muß in einem eigenen Baustein geschehen. Diese Aufgabenstellung übernimmt auf einem Standard-PC-Motherboard der sogenannte Chipsatz. In unserem Fall wird die Buskontrolle von zwei ISP-Bausteinen übernommen. Diese werden in den folgenden beiden Abschnitten näher beschrieben.

4.4.1 Das Kontroll-FPGA

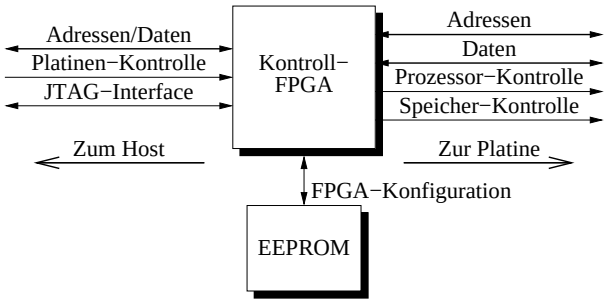

Abbildung 4.4: Das Kontroll-FPGA

Zwischen Host und den Rest der Platine wird ein Baustein eingesetzt, der die Kommunikation zwischen dem Prozessor-Speicherteil und dem Hostinterface regelt. Da dieser Baustein die Signale mehrerer breiter Busse (Datenbus, Adressbus und Hostbus) verarbeiten können muß, benötigt man einen Chip, der ausreichend viele I/O-Pins besitzt. Außerdem sollte man in ihm sowohl einen Zwischenspeicher für die vom Host kommenden Daten und Adressen als auch sowie einen Automaten zur Steuerung und Arbitrierung der verschiedenen Busse implementieren können. Hierzu bieten sich FPGAs (Field Programmable Gate Arrays) an.

Zur Impementierung der Busprotokolle wurde ein XILINX-FPGA des Typs XC-4062-XLA mit 240 Pins benutzt [39]. Bei diesem Baustein handelt es sich um einen JTAG-ISP, der es dem Designer ermöglicht, Schaltungen mit 40.000-130.000 Gatterequivalenten zu implementieren.

Das benutzte FPGA hat den Nachteil, daß es im ausgeschalteten Zustand seine Programmierung verliert. Damit die Konfiguration des FPGA nicht jedesmal beim Einschalten der Platine erst durch den Host-Rechner über JTAG erfolgen muß, ist außerdem ein EEPROM [5] zur Speicherung der entsprechenden Konfigurationsdaten vorgesehen (siehe Abbildung 4.4).

4.4.2 Das Speicher-EPLD

Das verwendete SRAM wird asynchron, das heißt unabhängig vom Taktsignal, betrieben. Deshalb müssen die entsprechenden Daten erst mit der Clock synchronisiert werden. Das heißt, man muß dafür sorgen, daß der SRAM-Chip genau zu den Taktflanken die gewünschten Daten liefert bzw. annimmt. Dazu

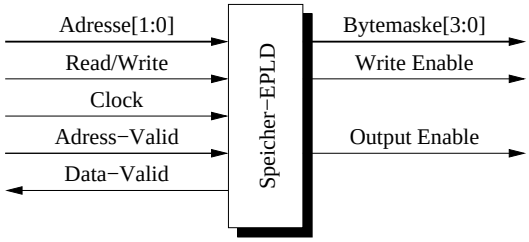

Abbildung 4.5: Das Speicher-EPLD

müssen die vom Prozessor benutzten Kontrollsignale verarbeitet und in die vom SRAM benötigten umgewandelt werden.

Ein weiteres Problem bei der Kommunikation der verschiedenen Komponenten auf der Platine ist die Tatsache, daß der verwendete SRAM-Baustein – im Gegensatz zum Rest der Schaltung – keine Byte-weise Adressierung benutzt, sondern eine Word-weise Adressierung, wobei ein Word genau 4 Bytes breit ist. Wenn das vom Prozessor angeforderte Word genau auf einer ganzen Wordadresse beginnt (*alligneter Zugriff*), so stellt dies kein Problem dar. Man vernachlässigt einfach die beiden unteren Adressbits des Prozessors. Es ist jedoch möglich, Programme so zu schreiben, daß Datenzugriffe nicht auf eine ganze Wordadresse fallen (*missalligneter Zugriff*) [15]. Abbildung 4.6 zeigt einen alligneten und einen missalligneten Zugriff auf ein grau unterlegtes Daten-Word.

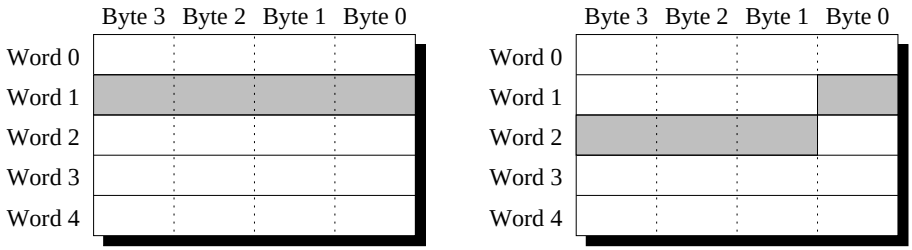

Abbildung 4.6: Alligneter und missalligneter Speicherzugriff

Da der OCRAM-Chip keine missalligneten Speicherzugriffe verarbeiten kann, sollten solche Zugriffe beim Programmieren vermieden werden. Sollte es jedoch nötig werden, missallignete Zugriffe auszuführen, so müßte man diese im FPGA in zwei getrennte Zugriffe zerlegen und die Ergebnisse zusammenfügen.

Beim verwendeten Speicherchip 8L32512V der Firma EDI [9] gibt es die Möglichkeit, über eine Maske die nicht gewünschten Datenbytes auszublenden. So kann bei missalligneten Zugriffen das gewünschte Word einfach aus zwei aufeinanderfolgenden Speicherzugriffen durch entsprechendes Shiften und anschließendes Verodern zusammengesetzt werden.

Die Synchronisierung der Daten mit dem Prozessor-Interface sowie die Erzeu-

gung der Maske sind in einem EPLD (Enhanced Programmable Logic Device) vom Typ 9536 der Firma XILINX implementiert [38]. Bei diesem Baustein handelt es sich wie beim FPGA um einen JTAG-ISP. Im Gegensatz zum FPGA bleibt die Konfiguration jedoch auch nach Entfernen der Stromversorgung vorhanden. Dafür sind die Anzahl der I/O-Pins und die mögliche Komplexität der implementierbaren Schaltungen geringer als beim FPGA.

Kapitel 5

Implementierung der Platine

In diesem Kapitel werden die Details der Implementierung vorgestellt. Dazu wird das Design in 5 funktionale Gruppen unterteilt (siehe Abbildung 5.1). Deren Implementierung wird dann in den jeweiligen Abschnitten erläutert.

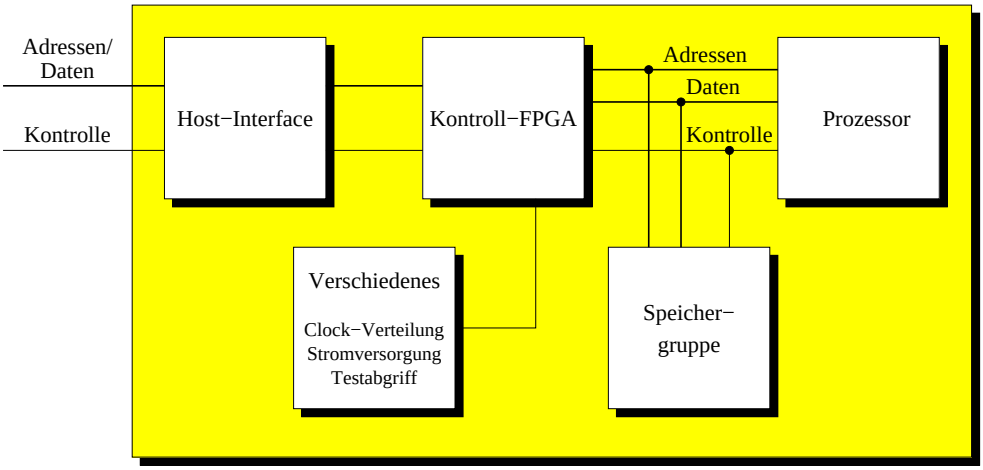

Abbildung 5.1: Unterteilung der Platine in Funktionsgruppen

5.1 Das Host-Interface

Das *Host-Interface* [6] stellt die Verbindung zwischen der Platine und der Außenwelt dar. Der Datenpfad, der dabei das FPGA der Platine mit dem Hostrechner verbindet, wird *Host-Pfad* genannt (siehe Abbildung 5.2).

5.1.1 Der Host-Pfad

Der Hostpfad erfüllt drei grundlegende Aufgaben:

- Die programmierbaren Komponenten FPGA, FPGA-EEPROM und Speicher-EPLD müssen über den Hostpfad reprogrammiert werden können.

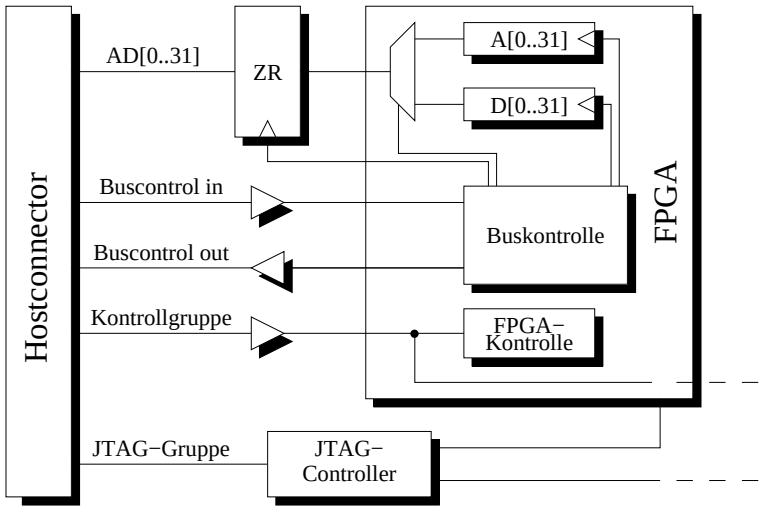

Abbildung 5.2: Datenpfade des Host-Interface

Die genaue Funktionsweise der JTAG-Programmierung wird in [7] erklärt.

- Signale wie Reset und Interrupt sowie Konfigurationssignale des FPGA müssen der Platine übermittelt werden können. Diese ermöglichen eine dynamische Kontrolle über die Boardfunktionen, also Möglichkeiten zur Interaktion mit der Platine im laufenden Betrieb.
- Als drittes muß der lokale Speicher der Platine beschrieben und ausgelesen werden können. Auf diese Art können Programme und Daten für den Prozessor in den Speicher gelegt und die Ergebnisse des auf dem Prozessor laufenden Programmes kontrolliert werden.

Im folgenden Kapitel werden die Implementierungen dieser einzelnen Gebiete genauer erläutert.

5.1.2 Der Host-Bus

Der Host-Bus gliedert sich wie bereits angedeutet in drei Signalgruppen, in denen die relevanten Signale der oben genannten Aufgabengebiete zusammengefaßt sind.

- Die JTAG-Gruppe besteht aus den Signalen **TDI**, **TDO**, **TCK** und **TMS**, die bereits in 2.3.1 erklärt wurden.
- Die Kontrollgruppe umfaßt die Signale **HRESET**, **SRESET**, **IRQ** und **FPGA-PROGRAM**. Diese werden einfach an ihren Zielort weitergeleitet und im jeweiligen Kapitel genauer erklärt.
- Den größten Teil des Busses stellt die Speicher-Gruppe dar. Diese besteht aus dem bidirektionalen Bus **AD[0..31]**, über den Daten und Adressen vom Host zur Platine übertragen und die ausgelesenen Ergebnisse in umgekehrter Richtung zurückübermittelt werden. Der Bus wird durch sechs

Steuersignale ergänzt, die im folgenden erklärt werden. Die Richtung der Signale ist dabei jeweils aus der Sicht der Platine angegeben.

Das Eingangssignal **HAS** (Host Address Strobe) wird vom Hostrechner genau dann aktiviert, wenn auf den Bus **AD** gültige Adressen liegen. Sobald gültige Daten auf dem Bus liegen, aktiviert der Host das Eingangssignal **HDS** (Host Data Strobe). Auf Acknowledge-Signale wird beim Hostinterface komplett verzichtet, stattdessen sind Adressen und Daten jeweils über einen Zeitraum von genau 2 Takten hin gültig.

Das Eingangssignal **HRW** (Host Read Write) dient zum Unterscheiden von Schreibzugriffen (**HRW**=0) und Lesezugriffen (**HRW**=1). Mit dem Ausgangssignal **HBUSY** (Hostinterface Busy) wird dem Hostrechner mitgeteilt, daß gerade ein Zugriff abgearbeitet wird. Das Eingangssignal **HDOE** (Host Data Output Enable) zeigt am Ende eines Lesezugriffs an, daß der Hostrechner bereit zum Empfang der auszulesenden Daten ist.

Desweiteren existiert das Signal **HSEL** (Host Selection), das die Signale **HAS**, **HDS** und **HDOE** für gültig erklärt. Nur wenn in der gleichen Phase zusätzlich zu den oben genannten Signale auch **HSEL** aktiviert ist, dürfen diese beachtet werden. Dieses Signal diente ursprünglich bei der *SB-PRAM* dazu, bei der Ansteuerung mehrerer Platinen die Kontrollsignale an alle Platinen anlegen zu können und die gewünschte Platine über das **HSEL**-Signal auszuwählen. In unserem Ein-Platinen-Fall ist davon auszugehen, das das Signal dauerhaft aktiv ist und deshalb ignoriert werden kann.

5.1.3 Busprotokoll des Host-Interface

Ein Problem beim Hostzugriff ist die Tatsache, daß das Hostinterface mit 5 Volt betrieben wird, der FPGA jedoch mit 3,3 Volt. Für den Weg vom Host zum FPGA stellt dies noch kein Problem dar, da laut den Spezifikationen des FPGA dieser problemlos Eingangssignale von 5 Volt verkraftet [39]. Der umgekehrte Weg würde wahrscheinlich nicht funktionieren, da über das Flachbandkabel zwischen Platine und Hostrechner zu viel Spannung verloren geht und die Bauteile auf der Hostseite eine stabile und hohe Spannung an den Eingangspins erwarten. Aus diesem Grunde wurden die mit 5 Volt betriebenen Zwischenregister ZR eingesetzt. Dabei handelt es sich um Register des Typs ABTH18504A der Firma Texas Instruments [35]. Diese kommen laut ihrer Spezifikationen problemlos mit niedrigen Eingangsspannungen zurecht, treiben ihre Daten jedoch garantiert mit einem hohen Potential in Richtung Hostrechner. Die Kontrollsignale werden über einen 5V Treiberbaustein des Typs ABT8652 der Firma Texas Instruments verstärkt [34]. Abbildung 5.3 zeigt die Datenpfade und Kontrollsignale, die beim Hostzugriff benötigt werden.

Zum Funktionieren des Hostinterface ist es nötig, sowohl die Systemclock als auch eine geviertelte Version der Systemclock über zwei getrennte Anschlüsse zum Hostrechner hin zu übermitteln. Alle Register und Zustanddiagramme des

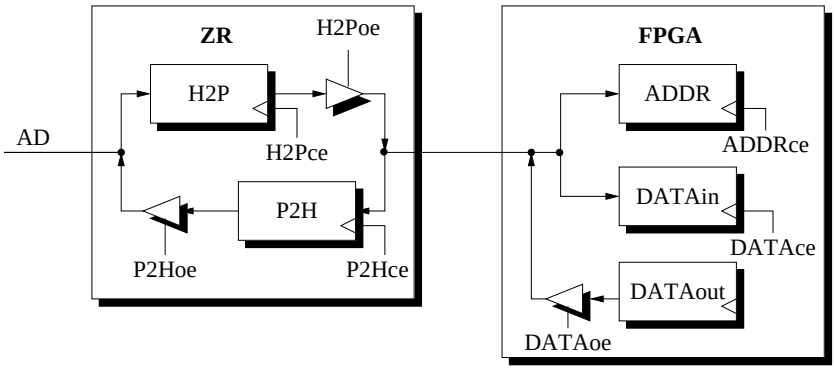

Abbildung 5.3: Datenpfade der Hostgruppe

Hostinterface werden dabei mit der geviertelten Clock getaktet. In unserem Fall wird diese Hostclock auf dem FPGA erzeugt. Beide Clocksignale werden aus den bereits genannten Gründen vor der Übertragung zum Host durch einem 5V-Clocktreiber vom Typ 49FCT805 verstärkt [33].

Abbildung 5.4 zeigt einen typischen Schreibzugriff des Hostinterface, Abbildung 5.5 einen typischen Lesezugriff. Das Protokoll jenseits des FPGA ist abhängig vom endgültigen Ziel des Zugriffs. Aus diesem Grund werden alle Daten im FPGA zwischengespeichert, um die verschiedenen Teilprotokolle voneinander zu trennen.

Bei einem Schreibzugriff werden vom Host nacheinander Adressen und Daten auf den gemeinsamen **AD**-Bus gelegt. Beide Werte bleiben für jeweils zwei Takte gültig. Gleichzeitig mit den Adressen wird das Signal **HAS** aktiviert, gültige Daten werden durch das Signal **HDS** signalisiert. Jeweils zu Beginn des zweiten Taktes werden die Adressen bzw. Daten in die Register ZR geclockt. Zeitgleich zum Einlesen der Adressen wird vom FPGA das Signal **HBUSY** aktiviert. Dieses bleibt bis zum Ende des Zugriffs aktiv. Der jeweilige Inhalt von ZR wird mit einem Takt Zeitverzögerung in ein Zwischenregister im FPGA geclockt, wo er dann weiterverarbeitet wird.

Bei einem Lesezugriff läuft die Adressübermittlung genau wie beim Schreibzugriff ab. Liegen die gewünschten Daten im FPGA vor, so werden sie zuerst in ZR zwischengespeichert. Gleichzeitig wird durch Deaktivieren des Signals **HBUSY** dem Host angezeigt, daß die Daten ausgelesen werden können. Die Daten bleiben nun in ZR, bis der Host sie auslesen möchte. Dies zeigt er durch Aktivieren des **HDOE**-Signals über 2 Takte hinweg an. Im zweiten Takt wird der Inhalt von ZR auf den **AD**-Bus getrieben und wird vom Host übernommen.

Implementiert wurde das Hostprotokoll auf dem FPGA in Form eines Moore-Automaten [26], der durch ein Zustandsdiagramm definiert ist. Dieses ist in Abbildung 5.6 zu erkennen. Die Beschriftung der Pfeile gibt dabei die Bedingungen für den jeweiligen Zustandswechsel an, die kursiv gedruckten Begriffe

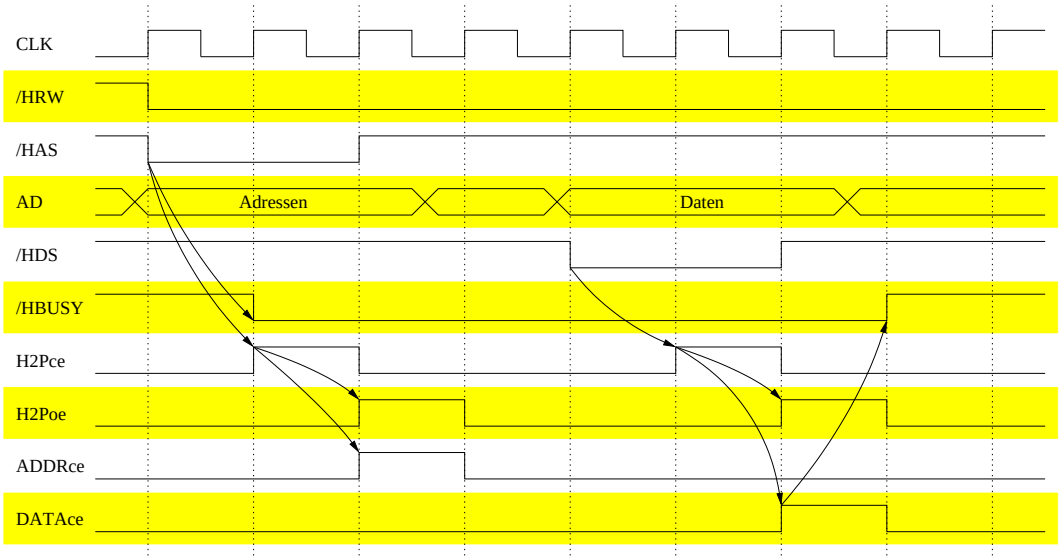

Abbildung 5.4: Schreibzugriff der Hostinterface

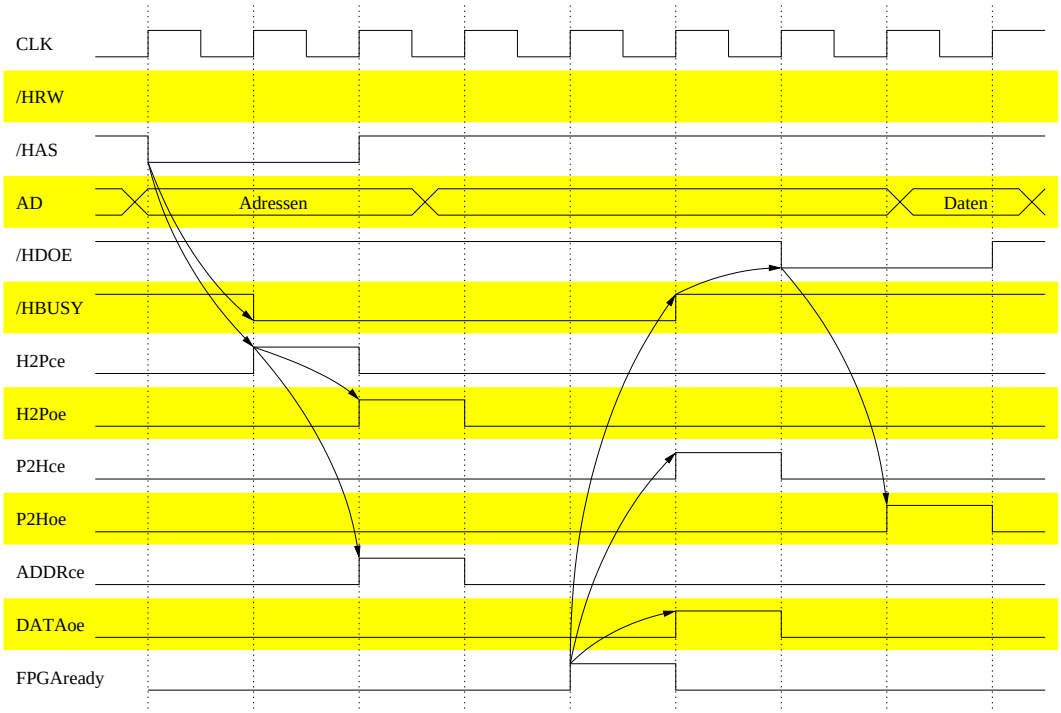

Abbildung 5.5: Lesezugriff des Hostinterface

innerhalb der einzelnen States geben in diesem Zustand aktivierten Kontrollsignale an. Die Signale **TA** und **FPGAready** kommen dabei aus Diagramm 5.11.

Während des kompletten Hostzugriffes werden dem Prozessor alle Berechtigun-

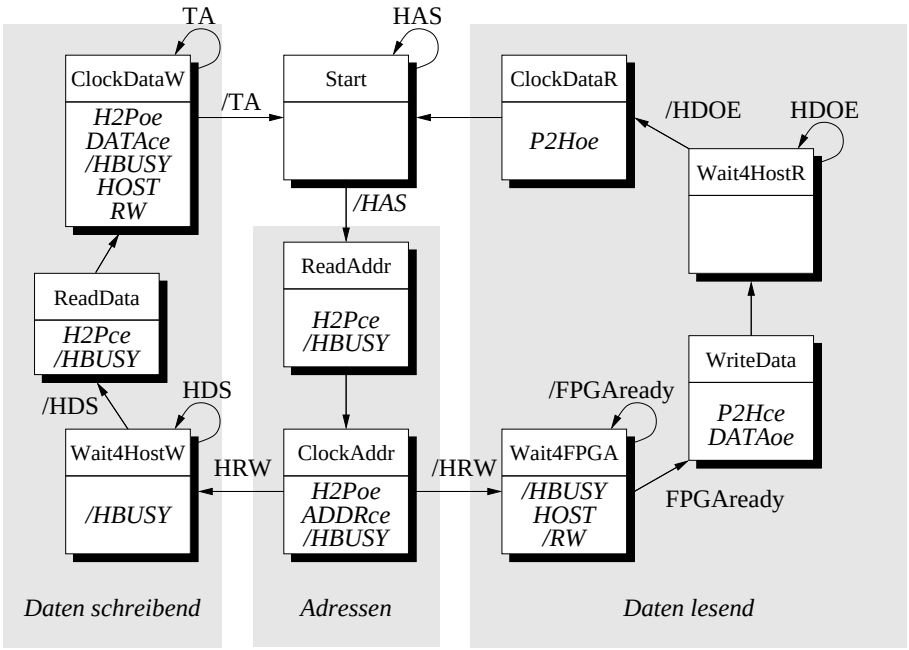

Abbildung 5.6: Zustandsdiagramm des Hostzugriffs

gen zum Schreiben auf den Adress- und den Datenbus der Platine entzogen (siehe Abschnitt 5.2.2). Dadurch werden Bus Contentions (gegeneinander treibende Bauteile auf dem gleichen Bus) vermieden. Da Hostzugriffe zu sehr großen Teilen *vor* dem Programmstart und dann erst wieder *nach* dem Programmende auftreten, hat diese Vorgehensweise nur wenig Wartezyklen für den Prozessor zur Folge und ist somit vollkommen ausreichend. Eine komplizierte Busarbitrierung wird überflüssig.

5.2 Die Prozessorumgebung

Als Prozessor wurde auf unserer Platine ein PowerPC des Typs 603E der Firma IBM eingesetzt. Es handelt sich dabei um einen in 4 Stufen gepipelineten Prozessor, der folgende Features besitzt:

- 32Bit Adressbus, wahlweise 64Bit oder 32Bit Datenbus
- bis zu 2 Instruktionen pro Takt ausführbar
- IEEE-754-kompatible Floating Point Einheit
- Einstellbarer Bus-Prozessor-Multiplikator
- 16K 4-Wege Programm-Cache
- 16K 4-Wege Daten-Cache
- Prefetch Queue, Branch Prediction, Stromsparmodus ...

Im folgenden werden das Prozessorinterface sowie die physikalische Einbettung des Prozessors beschrieben. Eine genauere Beschreibung der Möglichkeiten des Prozessors findet sich in [14].

5.2.1 Das Prozessor-Interface

Da es sich beim PowerPC um einen sehr komplexen Prozessor handelt, werden an dieser Stelle nur kurz die benötigten Teile des Interface erklärt. Außerdem wird auf die wichtigsten Signale und Busse eingegangen. Für eine genauere Beschreibung sei auf [15] verwiesen.

Der Prozessor besitzt die in Tabelle 5.1 aufgeführten externen Busse. Der Eintrag in der Spalte I/O gibt dabei an, ob es sich bei den betreffenden Signalen um Eingangssignale (I), Ausgangssignale (O) oder um bidirektionale Signale (B) handelt (jeweils von der CPU aus gesehen).

Bus	I/O	Beschreibung
A[0..31]	O	Adressbus für Speicherzugriffe
AP[0..3]	O	Paritysignale für den Adressbus
DH[0..31]	B	Untere Hälfte des Datenbusses
DL[0..31]	B	Obere Hälfte des Datenbusses
DP[0..7]	B	Paritysignale für den Datenbus
TT[0..4]	O	Transfer-Typ des Speicherzugriffs
PLLCFG[0..3]	I	Einstellung des internen Clock-Multiplikators

Tabelle 5.1: Prozessorbusse

Auf dem Bus TT wird der genaue Typ des Speicherzugriffs angezeigt (Burst oder Single Beat, Instruktion oder Daten...). Eine vollständige Auflistung der Kombinationen findet sich in [15]. Diese Informationen werden im Mehrprozessorbetrieb zur Kommunikation der Prozessoren untereinander benutzt. Auf der Platine wird nur das Signal TT1 verwendet, das anzeigt, ob es sich um einen Schreibzugriff (0) oder einen Lesezugriff (1) handelt. Die übrigen TT-Signale werden ignoriert.

Die Werte PLLCFG[0..3] können über Jumper direkt auf der Platine eingestellt werden. Die verschiedenen Clock-Multiplikatoren und ihre jeweiligen Einstellung sind im Anhang aufgeführt.

Zur Kommunikation zwischen Prozessor und Platine existieren von Seiten der CPU her die in Tabelle 5.2 aufgeführten Kotrollsignale. Alle aufgeführten Kontrollsignale sind activ low. Teilweise werden in [29] oder auch in den Schematics im Anhang andere Signalnamen benutzt als in der PowerPC-Dokumentation. In diesen Fällen sind beide Name aufgeführt. Die wichtigsten benutzten Signale werden im folgenden genauer beschrieben.

Signal	I/O	Beschreibung
ABR	O	Adress Bus Request: Fordert Reservierung des Adressbusses
ABG	I	Adress Bus Grant: Erteilt Reservierung des Adressbusses
ABB / AV	B	Adress Bus Busy: Ist aktiv wenn Adressen gültig
AACK	I	Adress Acknowledge: Adresstransfer war erfolgreich
ARTY	I	Adress Retry: Adressbus freigeben, später wieder versuchen
TS	O	Transfer Start: Fordert Reservierung des Datenbusses
DBG	I	Data Bus Grant: Erteilt Reservierung des Datenbusses
DBB / DV	B	Data Bus Busy: Ist aktiv wenn Daten gültig oder erwartet
TA / DACK	I	Transfer Acknowledge: Datentransfer war erfolgreich
DRTY	I	Data Retry: Übertragene Daten sind ungültig
TEA	I	Transfer Error Acknowledge: Busfehler, evtl. Checkstop
HRESET	I	Hard Reset: Prozessor-Reset, alle Ausgänge auf neutral
SRESET	I	Soft Reset: Software-Reset, Reset-Exception
CLK	I	System Clock: Globaler Clock-Eingang
TCLK	O	Test Clock: Interne (multiplizierte) Clock
TDI	I	Test Data Input: Dateneingang des JTAG-Testbusses
TDO	O	Test Data Out: Datenausgang des JTAG-Testbusses
TMS	I	Test Mode Select: Wahl des JTAG-Testmodus
TCK	I	Test Clock: JTAG-Clock

Tabelle 5.2: Kontrollsignale

5.2.2 Busprotokoll auf dem Prozessorbus

Der Pfad, der den Prozessor und das FPGA einerseits mit dem OCRAM und dem SRAM andererseits verbindet, wird Prozessorbus genannt. Abbildung 5.7 zeigt die Datenpfade des Prozessorbusses.

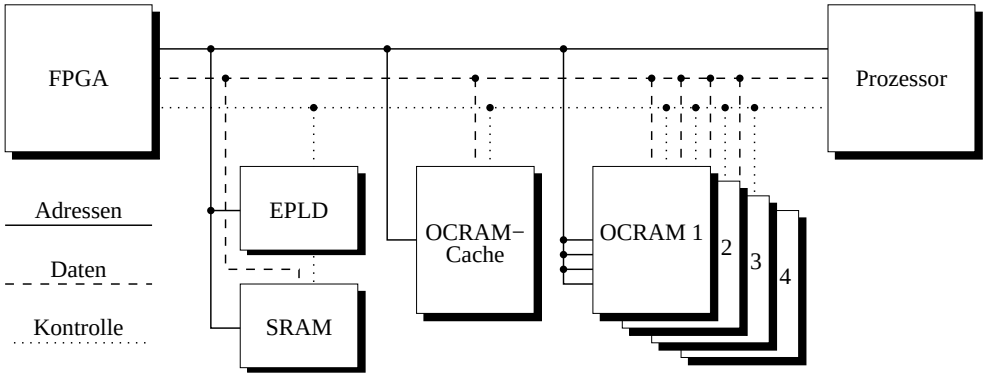

Abbildung 5.7: Datenpfade des Prozessorbus

Auf dem Prozessorbus wird für alle Arten von Zugriffen das gleiche Busprotokoll benutzt. Dieses Protokoll entspricht einer vereinfachten Version des PowerPC-Busprotokolls und wird in den folgenden Abschnitten genauer erklärt.

Das Busprotokoll des OCRAMs wurde bei dessen Entwicklung so konzipiert, daß es dem Protokoll des Prozessors entspricht. Um ohne Änderung des Protokolls neben den OCRAM-Chips auch auf den SRAM-Chip zugreifen zu können, wird im Falle eines SRAM-Zugriffs mittels eines dem SRAM vorgeschalteten EPLD das Protokoll des SRAM-Chips an das Prozessorprotokoll angepaßt. Die genaue Funktionsweise dieser Schaltung wird in Kapitel 5.3.2 erläutert.

Um über das FPGA auf den Speicher zugreifen zu können, wird auf dem FPGA das Protokoll des PowerPC simuliert. Die Implementierungsdetails dieser Zugriffsart werden in Kapitel 5.3.1 erklärt. Für den Speicher ist es dementsprechend unerheblich, ob der Zugriff vom Prozessor oder vom FPGA ausgeführt wird.

Abbildung 5.8 zeigt einen typischen Schreibzugriff, Abbildung 5.9 einen typischen Lesezugriff des Prozessors auf den Speicher.

Bevor der Prozessor Signale auf den Adress- bzw. Datenbus legt muß er zuerst den entsprechenden Bus anfordern. Dies geschieht im Falle des Adressbus durch das Signal **ABR**, im Falle des Datenbus durch das Signal **TS**. Die Kontrolle im FPGA erteilt die angeforderten Rechte mithilfe der Signale **ABG** für den Adress- und **DBG** für den Datenbus. Die Bus-Grant-Signale sind immer aktiviert, sofern nicht gerade ein Hostzugriff abgearbeitet wird. Hostzugriffe werden durch das Signal **HOST** angezeigt (siehe Abbildung 5.6). Ist **HOST** aktiv, so werden dem Prozessor die Bus-Grant-Signale entzogen und er muß warten. Eventuell laufende Speicherzugriffe werden durch das Signal **DRTRY** abgebrochen.

Jeder Speicherzugriff gliedert sich in eine Adress- und eine Datenphase. Die Adressphase ist für Lese- und Schreibzugriffe gleich. Lediglich in der Datenphase wird zwischen Lese- und Schreibzugriffen unterschieden.

In der Adressphase des Zugriffs zeigt der Prozessor durch das Signal **ABB** an, daß gültige Adressen auf dem Adressbus liegen. Dieses Signal und die Adressen bleiben solange aktiv, bis der Speicher das Signal **AACK** aktiviert und damit anzeigt, das er die Adressen eingelesen hat. Gleichzeitig mit den Adressen ist das Signal **TT1** gültig, welches anzeigt, ob es sich um einen Schreibzugriff (**TT1** = 0) oder einen Lesezugriff (**TT1** = 1) handelt.

Bei einem Schreibzugriff aktiviert der Prozessor das Signal **DBB**, um anzuzeigen, daß gültige Daten auf dem Datenbus liegen. Hat der Speicher die Daten erfolgreich gelesen, so zeigt er dies durch Aktivieren von **TA** an.

Im Falle eines Lesezugriff wird das Signal **DBB** von Seiten des Prozessors aktiviert, sobald dieser bereit ist, die Daten einzulesen. Der Speicher seinerseits zeigt durch aktivieren von **TA** an, daß die angeforderten Daten auf dem Datenbus bereitliegen.

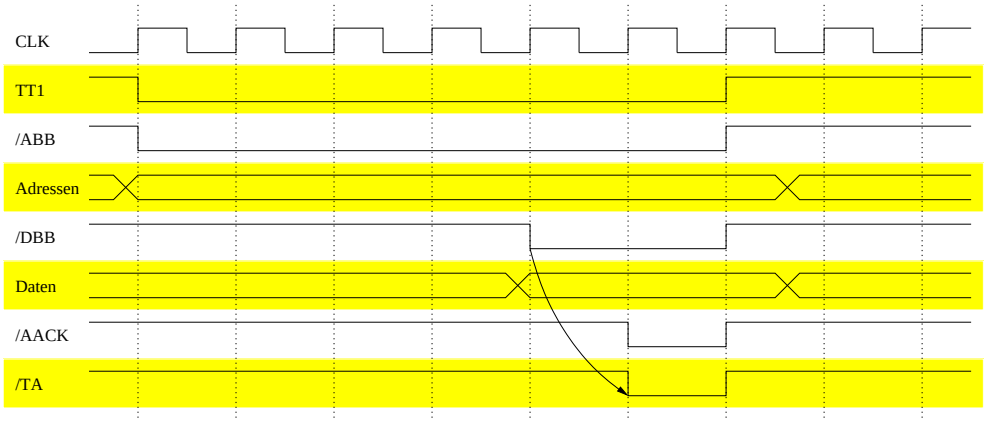

Abbildung 5.8: Schreibzugriff der Prozessorinterface

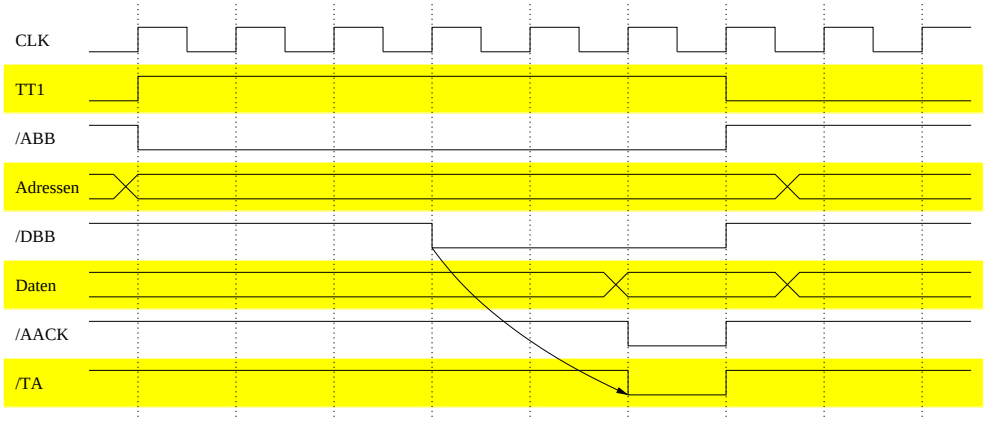

Abbildung 5.9: Lesezugriff der Prozessorinterface

Die Arbitrierung der Kontrollsignale **ABB**, **DBB**, **AACK** und **TA** geschieht wie folgt:
Solange ein Ausgangskontrollsignal von einem Baustein nicht aktiviert ist, setzt der Baustein den entsprechenden Ausgangspin auf *elektrisch neutral*. Treibt kein Baustein das Signal, so wird es durch einen *Pull-Up-Widerstand* auf inaktiv gezogen (siehe Abbildung 2.4). Diese Form der Schaltung wird auch als *Wired OR* bezeichnet.

5.2.3 Einschränkungen des PowerPC

Um die physikalische Integration des PowerPC zu erleichtern, wurden nicht benötigte Teile des Prozessors intern abgeschaltet oder nicht nach außen angeschlossen. Dies geschieht durch Negieren bestimmter Eingangssignale während eines Prozessor-Resets. Dadurch gibt es gegenüber den in [14] aufgeführten Spezifikationen des Prozessors folgende Einschränkungen:

- Der interne Cache des Prozessors wird durch Negieren des Signals **DRTRY**

ausgeschaltet. Dies ist notwendig, da der Prozessorcache nur über Burst-Zugriffen auf den Hauptspeicher zugreift, der OCRAM-Chip jedoch keine Burst-Zugriffe unterstützt. Ist der Cache hingegen ausgeschaltet, so werden vom Prozessor nur Einzelanfragen auf den Speicher ausgeführt.

- Da es sich bei der kompletten Platine um ein 32-Bit-System handelt (alle verwendeten Daten-Busse sind 32 Bit breit), werden die optional benutzbaren höheren Datenbits DL[0..31] im Prozessor durch Negieren des Signals **TLBISYNC** abgeschaltet und auch nach außen hin nicht angeschlossen. Eine Nutzung der vollen Datenbreite von 64 Bit wäre jedoch problemlos möglich, indem man jeweils zwei Speichermodule nebeneinandersetzt, von denen eines die unteren Datenbits DH beinhaltet und das zweite die oberen Datenbits DL. Beide Speicherbausteine könnten über die gleichen Kontrollsignale angesteuert werden.

- Der Prozessor wird durch Negieren des Signals **QACK** in den Reduced Pinout Mode geschaltet. Dadurch werden die Pins der Parity-Busse AP[0..3] und DP[0..7] sowie der höheren Datenbits DL[0..31] nach außen hin abgeschaltet. Dieser Modus wurde gewählt, da die genannten Signale sowieso nicht benutzt werden und der Modus stromsparender und damit weniger abwärmeintensiv ist.

5.3 Die Speicher-Umgebung

Im folgenden soll die Speicher-Umgebung auf der FiberLink-Platine beschrieben werden. Zuerst wird dabei der Zugriff auf die OCRAM-Chips erklärt. Diese sind jedoch noch nicht vorhanden. Sie können aber von SRAM und Speicher-EPLD simuliert werden. Dies wird in Abschnitt 5.3.2 näher erläutert.

5.3.1 Der OCRAM-Zugriff

In diesem Zusammenhang ist lediglich der Zugriff vom FPGA auf das OCRAM interessant, da die Prozessor-Speicherkommunikation – wie bereits beschrieben – weitestgehend unabhängig von der Platinenkontrolle abläuft.
In Abbildung 5.10 sind die Datenpfade und Kontrollsignale zu sehen, welche für den Zugriff vom FPGA auf die OCRAM-Chips benötigt werden.

Das verwendete Protokoll des OCRAM-Moduls ist genau auf das Protokoll des Prozessors angepaßt. Zum direkten Schreiben und Lesen des Speichers vom FPGA bietet es sich folglich an, das Prozessorprotokoll auf dem FPGA zu simulieren. Aufgrund folgender Voraussetzungen konnte das Protokoll für diese Zugriffe jedoch stark vereinfacht werden:

- Aufgrund eines Fehlers in der Kontrollogik des OCRAM-Chips [29] wird statt der zwei getrennten Acknowledge-Signale **AACK** und **TA** nur das Signal **TA** benutzt. Die Aktivierung von **TA** impliziert dabei die Aktivierung von **AACK**.

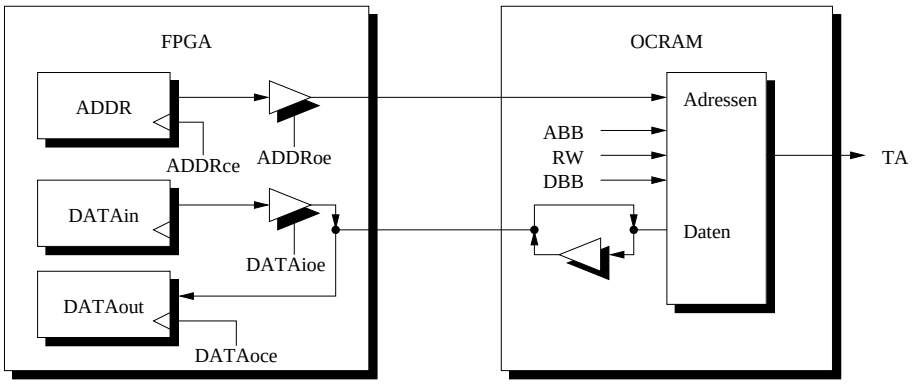

Abbildung 5.10: Datenpfade für den OCRAM-Zugriff

- Bei einem Schreibzugriff liegen die zu schreibenden Daten immer gleichzeitig mit den Adressen vor, da beides beim Hostzugriff in die entsprechenden Register geclockt wurde. So können gleichzeitig die Adressen und Daten an den OCRAM-Chip angelegt werden, bis dieser durch Aktivieren von **TA** das erfolgreiche Ende des kompletten Schreibvorgangs signalisiert.
- Auch bei einem Lesezugriff schadet es nicht, über den kompletten Zugriff hinweg die Adressen an den OCRAM-Chip anzulegen. Desweiteren ist der FPGA jederzeit bereit, die Daten aufzunehmen. Aus diesem Grunde kann auch das Signal **DBB** die ganze Zeit hindurch aktiviert bleiben.

Das vereinfachte Zustandsdiagramms wird in Abbildung 5.11 dargestellt. Die Signale **HOST** und **RW** stammen dabei aus Diagramm 5.6.

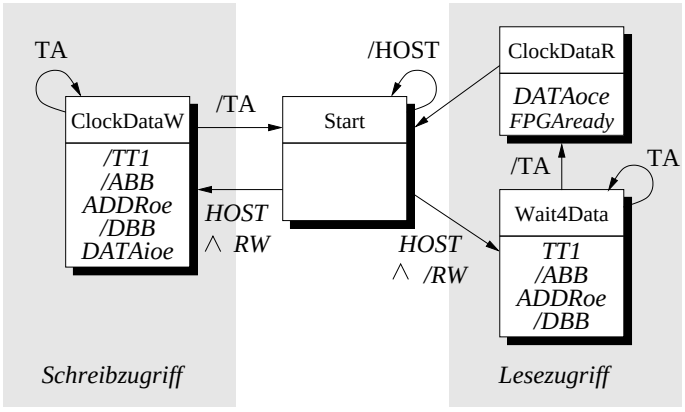

Abbildung 5.11: Zustandsdiagramm des simulierten Prozessor-Protokolls

5.3.2 Der SRAM-Zugriff

Um den Speicherbereich für Programme und Daten zu vergrößern wurden auf der Platine zusätzlich ein SRAM-Speicherchip sowie ein EPLD integriert. Außerdem kann mittels dieses Speicher-EPLD beim Zugriff auf das SRAM ein

OCRAM-Zugriff simuliert werden. Dazu muß das EPLD die entsprechenden Kontrollsignale des Prozessors verarbeiten bzw. erzeugen und mit dem Protokoll des SRAM-Chips abgleichen können.

Beim verwendeten SRAM [9] handelt es sich um Bausteine mit einer Zugriffszeit von 15ns. Das heißt für einen Lesezugriff, daß vom Anlegen von gültigen Adressen an die Adresspins bis zum Gültigwerden der gewünschten Daten maximal 15ns vergehen. Beim Schreibzugriff werden die anliegenden Daten innerhalb von 15ns an der angegebenen Stelle im Baustein gespeichert. Bei einer geplanten Taktfrequenz von 10MHz beträgt die Länge eines Taktes 100ns. Alle Zugriffe auf das SRAM werden also problemlos innerhalb eines Taktzyklus verarbeitet.

In Abbildung 5.12 sind die Datenpfade und Kontrollsignale zu sehen, welche für den Zugriff über das EPLD auf die SRAM-Chips benötigt werden. Bei den Signale **W** und **G** handelt es sich dabei um Eingangssignale des SRAM. Aktivierung des Signal **W** zeigt einen Schreibzugriff auf das SRAM an, das Signal **G** ist das globale Output-Enable des SRAM-Chips.

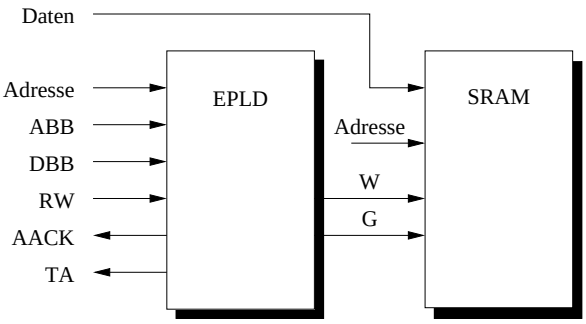

Abbildung 5.12: Datenpfade für den SRAM-Zugriff

Abbildung 5.13 zeigt das Zustandsdiagramm des Speicher-EPLD.

Im EPLD wird aufgrund der oberen Adressbits entschieden, ob der Zugriff auf das SRAM ausgeführt wird oder nicht. Da die Adressen über den gesamten Zugriff hinweg am SRAM anliegen müssen, wird die Aktivierung des Signals **AACK** bis zum Ende des kompletten Zugriffs hinausgezögert. Bei Neudesign oder Weiterentwicklung der Platine sollte an dieser Stelle ein Register eingefügt werden, in dem die Adresse zwischengespeichert werden kann, um das Signal **AACK** direkt aktivieren zu können.

Im Falle eines Schreibzugriffs wartet das EPLD nun auf die Aktivierung des Signals **DBB**. Ist diese erfolgt, so wird das Signal **TA** aktiviert. Zur Seite des SRAM-Bausteins hin wird das Schreibsignal **W** aktiviert. Damit werden die Daten innerhalb des nächsten Taktes in den Speicher geschrieben.

Handelt es sich um einen Lesezugriff, so wartet das EPLD ebenfalls auf das

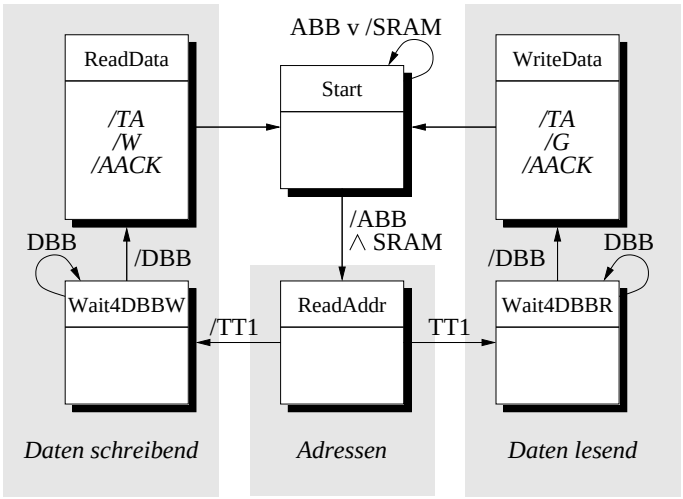

Abbildung 5.13: Zustandsdiagramm des simulierten OCRAM-Protokolls

Signal **DBB**. Sobald dieses aktiviert wurde wird das entsprechende Signal **TA** aktiviert und auf Seiten des SRAM mittels des Signals **G** das Output-Enable für die Datenpins gesetzt.

Auf diese Art und Weise verhält sich die Kombination SRAM-Baustein und Speicher-EPLD nach außen hin wie ein OCRAM-Baustein und kann somit problemlos auch direkt vom Prozessor angesprochen werden.

5.4 Die FPGA-Umgebung

Ein FPGA ist ein programmierbarer Logikbaustein, der aus frei konfigurierbaren Zellen besteht, den sogenannten Standardzellen. Diese können untereinander verbunden werden. Jede Standardzelle wird für eine bestimmte Aufgabe (z.B. logische Gatter, Register...) programmiert. Auf diese Weise können mittels eines FPGA komplexe Schaltungen realisiert werden.

Auf der FiberLink-Testplatine wird das FPGA für folgende Aufgabengebiete eingesetzt:

- Implementierung des Host-Interface und Bereitstellung von Zwischenregistern für Adressen und Daten (siehe Abschnitt 5.1.3).
- Implementierung der Busfreigabe für Prozessorzugriffe, wie in Abschnitt 5.2.2 erklärt.
- Implementierung der Simulation des Prozessor-Protokolls zum direkten Zugriff auf OCRAM und SRAM. Dies wurde in Abschnitt 5.3.1 erläutert.

Das FPGA ist mit diesen Schaltungen noch lange nicht komplett ausgefüllt. Es ist jedoch mit Absicht größer dimensioniert worden, um es dem Designer zu

erlauben, Workarounds für Fehler zu implemetieren oder fehlende bzw. fehlerhafte Teile zu simulieren.

Die Eingabe der entsprechenden Datenpfade und Zustandsdiagramme geschieht dabei in der XILINX-eignenen Design-Umgebung *Foundation* [40]. Diese besitzt ein CAD-Tool zur direkten Eingabe von Schaltungen auf Gatterniveau, ein Tool zur Eingabe von Hardware mittels der Hardwarebeschreibungssprachen *Verilog*, *VHDL* oder *ABEL* [4], sowie eine Oberfläche zur direkten Eingabe von Zustandsdiagrammen.

In *Foundation* wird dann aus der Hardwarebeschreibung ein Konfiguration des FPGA erzeugt, die eben diese Hardware implementiert. Zur Programmierung des FPGA mit dieser Konfiguration stehen zwei unterschiedliche Möglichkeiten zur Verfügung:

- Der Baustein kann direkt über den JTAG-Pfad mit Hilfe des Programms *JTL* von Torsten Brunklaus [7] programmiert werden. Da das FPGA die Programmierung jedoch ohne Stromversorgung nicht behält, muß dies bei jedem Einschalten der Platine aufs neue geschehen.
- Alternativ kann die Konfiguration auch in einem eigens dafür vorgesehenen EEPROM gespeichert werden [5]. Dieses ist speziell für die Speicherung von FPGA-Konfigurationen entworfen. Es wird dazu einfach an die dafür vorgesehenen Ausgänge des FPGA angeschlossen. Zur Programmierung des FPGA muß dann nur noch das Eingangssignal **PROGRAM** am FPGA aktiviert werden. Dies geschieht über die Benutzeroberfläche der Hostinterface-Steuerung *PRAMOS*.

5.5 Verschiedenes

In der Gruppe "Verschiedenes" sind die Teile des Designs zusammengefaßt, die zu klein sind, um als eigenständige Gruppen aufgefasst zu werden. Dazu gehören:

- Die Clockverteilung

  Herrscht in einer Schaltung eine bestimmte Hauptdatenflußrichtung vor (zum Beispiel in Pipelines), so sollte die Clockverteilung in Flußrichtung von hinten nach vorne erfolgen ([18], [26]). Im vorliegenden Design liegt jedoch keine solche Hauptrichtung vor. Aus diesem Grund wird im Sinne einer stabilen und gleichmäßigen Clockverteilung die Systemclock durch einen Treiberbaustein des Typs 49FCT805 verstärkt und parallel an die 4 oben genannten Funktionsgruppen weiterverteilt.

- Die Stromversorgung

  Neben dem Adapter zum Anschluß eines handelsüblichen ATX-Netzteils findet sich auf der Platine ein Schalter, mit dem sich das Netzteil von der Platine aus ein- und ausschalten läßt. Desweiteren existieren mehrere

Kontrolleuchten, die den Status der Stromversorgung (3.3V OK, 5V OK, ATX-Netzteil OK) erkennen lassen. Die Bedeutung der einzelnen Leuchten ist im Anhang erklärt.

- Filterung der ECL-Stromversorgung

  Da die in ECL-Technologie betriebenen Komponenten eine viel stabilere Spannungsversorgung brauchen als der CMOS-Teil, muß die Stromversorgung für die ECL-Bauteile gefiltert werden. Dies geschieht wie in [17] beschrieben durch 4 Induktivitäten der Stärke 4μH, die zwischen die beiden Versorgungslagen geschaltet sind. Sollten die OCRAM-Bausteine sehr viel Strom brauchen und deshalb zu viele Störungen auf der ECL-Versorgungslage verursachen, so werden eventuell noch weitere Filtermaßnahmen nötig.

- Der Lüfteranschluß

  Für den Fall, das eines der Bauteile (Prozessor oder OCRAM) im Betrieb zu heiß werden sollte und deshalb gekühlt werden muß, ist auf der Platine ein Stromanschluß für einen Standard 12V Lüfter vorgesehen. Auf der momentanen Platine wird dieser Anschluß nicht benutzt. In Abhängigkeit von der Abwärmeentwicklung des OCRAM-Chips wird es in der endgültigen Version der Platine eventuell nötig, noch weitere Lüfteranschlüsse vorzusehen.

- Der Testabgriff

  Um alle wichtigen Kontrollsignale bequem und gezielt abgreifen und kontrollieren zu können, wurden diese über einen 50-poligen Steckverbinder nach außen geführt. Auf diese Weise ist eine einfache Überwachung des Platinenbetriebs mithilfe eines Logikanalysators möglich. Die Pinbelegung dieses Testabgriffes ist im Anhang A aufgeführt.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel wird der Inhalt dieser Arbeit noch einmal kurz zusammengefaßt. Im Anschluß daran wird ein Ausblick gegeben, was noch im Projekt zu tun ist und was zu tun wäre, um den Prototypen zu einem kommerziellen System weiterzuentwickeln.

6.1 Zusammenfassung

Bei heutigen Computersystemen stellt die beschränkte Bandbreite der elektrischen Datenübertragung ein ernsthaftes Problem dar. Eine Erhöhung der Bandbreite über elektrische Leitungen ist durch die auftretenden physikalischen Seiteneffekte enorm schwierig. Glasfaserverbindungen jedoch versprechen eine Schub in der erreichbaren Datenrate im Bereich von mehreren Größenordungen.

Bisher bot sich die optische Datenübertragung nur für die Kommunikation zwischen weit voneinander entfernten Systemen an. Das liegt hauptsächlich daran, das die elektro-optische Umwandlung nicht "on Chip", sondern in externen Komponenten (zum Beispiel ATM-Adaptern) vorgenommen wird. Dies wiederum findet seine Begründung in der Größe der optischen Sender- und Empfängereinheiten. Die Probleme der elektrischen Übertragung werden nicht gelöst, sondern nur auf das Teilstück zwischen Chip und Umwandler reduziert.

Durch die Entwicklung hinreichend kleiner optischen Bausteine ist es inzwischen möglich, die Chiplogik sowie optische Sender- und Empfängereinheiten in einem gemeinsamen Gehäuse unterzubringen. Im Rahmen eines gemeinsamen Pilotprojektes des Lehrstuhls für Rechnerarchitektur an der Universität Saarbrücken und des Institutes für Technische Physik am Deutschen Zentrum für Luft- und Raumfahrt DLR wird ein kombinierter Cache- und Memorychip entwickelt, in dem solch ein optisches Kommunikationsinterface direkt integriert ist.

Ziel dieser Diplomarbeit war der Entwurf und die Realisierung einer Testumge-

bung für den Chipprototypen. Dies erfolgte in Form einer Prozessorplatine, auf der ein Prozessor des Typs IBM PowerPC auf den neu entwickelten Cachechip zugreifen kann. Der Cache wiederum kommuniziert über Glasfaserkabel mit 4 Speicherchips.

Bisher wurde eine Testplatine ohne die neuen Chips gebaut. Dies hat folgende Gründe:

- Im Sinne eines einfachen und effektiven Debuggings soll der Rest der Platine schon einmal getrennt auf Fehler untersucht werden können. Der Chip kann dann in eine getestete Umgebung eingesetzt werden.
- Entwurf, Entwicklung und Produktion der Chips erwiesen sich aus unterschiedlichsten Gründen als extrem zeitintensiv. Der Chip ist noch immer nicht fertig. Im Moment befindet sich der Logik-Die in der Fertigungskontrolle, bevor er zum Einbau in sein Gehäuse zur DLR geschickt wird.

6.2 Zukunft des Projektes

Der nächste Schritt im Rahmen des FiberLink-Projektes ist die Implementierung des OCRAM-Chips in das Platinendesign. Dies kann geschehen, sobald das endgültige Aussehen des Chips (Größe, Pinout, ...) feststeht. Da aus verschiedenen Gründen kein JTAG auf dem Chip implementiert werden konnte, müssen alternative Methoden entwickelt werden, um Fehler auf dem Chip effizient finden zu können.

Beim FiberLink-Projekt handelt es sich um eine reine Machbarkeitsstudie. Um den Chip kommerziell einsetzen zu können, müßten folgende konzeptuellen Änderungen vorgenommen werden:

- Die erreichten Datenübertragungsraten stellen keine Verbesserung zu denen dar, die in handelsüblichen Systemen auf elektrischem Wege erreicht werden. Dazu sind sowohl die Datenraten von nur 320 MHz als auch die beschränkte Anzahl von lediglich 10 optischen Leitungen viel zu tief angesetzt. Im Gegensatz zur elektrischen Übertragung ist allerdings bei der optischen Übertragung jeweils eine Erhöhung um ganze Größenordnungen denkbar.
- Bei einem Aufbau als Multichip-Modul erweist sich insbesondere eine Erhöhung der Anzahl der optischen Leitungen als äußerst schwierig. Um eine weitere signifikante Verkleinerung der Laser und Empfänger zu erreichen, können diese inklusive der nötigen Verstärker direkt auf einem GaAs-Die gefertigt werden [11]. Beim Flip-Chip-Bonding wird dieser Die direkt auf den Logik-Die aufgelegt und kann auf der ganzen Fläche durch Lötkontakte verbunden werden [12]. Diese Methode wird jedoch weltweit von nur wenigen Unternehmen beherrscht [23] und ist dementsprechend noch enorm kostspielig.

- Das optische Kommunikationsinterface muß direkt auf dem Prozessor integriert werden. Die in unserem Projekt vorhandene elektrische Lücke zwischen Prozessor und Cache muß geschlossen werden. Desweiteren ist der verfügbare Speicher auf den OCRAM-Chips viel zu klein. Das Interface sollte auf kommerziellen DRAM-Bausteinen implementiert werden.

Um aus dem Prototypen ein kommerzielles Produkt herstellen zu können, ist eine enge Zusammenarbeit mit der Industrie sowohl aus finanziellen als auch aus technologieabhängigen Gründen unabdingbar.

6.3 Weitere Einsatzgebiete

Das verwendete Interface kann prinzipiell überall dort eingesetzt werden, wo hohe Bandbreiten wichtig sind. Beispiele hierfür sind die Verbindungen zwischen:

- Haupspeicher und I/O-Puffer
- verschiedenen I/O-Puffern
- Hauptspeicher und Grafikkarte
- verschiedenen Cache-Levels

Dabei läßt sich das verwendete Interface problemlos auf diese Anwendungsgebiete portieren. Darüberhinaus ist es möglich, eine Hardwarekomponente direkt mit mehreren diese Interfaces auszustatten (zum Beispiel einen Cache sowohl in Richtung des nächsthöheren als auch des nächstniedrigeren Levels). Man müßte lediglich in Zusammenarbeit mit den entsprechenden Hardwareherstellern einen Standard schaffen und die Technologie direkt in den entsprechenden Produkten implementieren.

Anhang A

Anzeigen und Anschlüsse auf der Platine

Leuchtdiode	Aussage
D2	5V Stromversorgung vorhanden
D3	3.3V Stromversorgung vorhanden
D5	ATX-Netzteil angeschlossen und bereit

Tabelle A.1: Leuchtdioden auf der Platine

Anschluß	Signal
J1	geviertelte Host-Clock
J2	Host-Clock
J3	Testabgriff
JP4	12V-Anschluß für eventuellen Lüfter
JP5	ATX-Netzteil
U16	Host-Connector

Tabelle A.2: Platinenanschlüsse

PLL-Config	Bus-to-Core Multiplikator
0000	1 x
1100	1.5 x
0100	2 x
0110	2.5 x
1000	3 x
1110	3.5 x
1010	4 x

Tabelle A.3: JP3: Einstellung des Clock-Multiplikators am Prozessor

Pin	Abgriff
1	3.3V
2	GND
3	PowerPC Clock
4	PPC Adress Bus Request
5	PPC Adress Bus Grant
6	PPC Adress Bus Busy
7	PPC Transfer Start
8	PPC Data Bus Grant
9	PPC Data Bus Busy
10	GND
11	PPC Data Bus Disable
12	PPC TT1 (Read/Write)
13	PPC Transfer Size 0
14	PPC Transfer Size 1
15	PPC Transfer Size 2
16	PPC TBST
17	PPC TC0
18	PPC TC1
19	GND
20	PPC Adress Acknowledge
21	PPC Adress Retry
22	PPC Transfer Acknowledge
23	PPC Data Retry
24	PPC Transfer Error Acknowledge
25	3.3V

Pin	Abgriff
26	Host Hard Reset
27	Host Soft Reset
28	Systemclock
29	PPC Checkstop In
30	PPC Checkstop Out
31	Host UD1
32	Host UD2
33	GND
34	Host Output Enable H2P
35	Host Output Enable P2H
36	Host Clock Enable H2P
37	Host Clock Enable P2H
38	Host Read / Write
39	Host Adress Strobe
40	Host Data Strobe
41	Host Busy
42	GND
43	Host UD3
44	FPGA Init
45	EEPROM LDC
46	EEPROM Serial Enable
47	EEPROM Device Select
48	EEPROM Clock
49	GND
50	3.3V

Tabelle A.4: Pinout des Testabgriffs

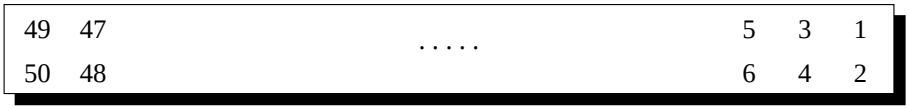

Abbildung A.1: Durchnummerierung der Pins am Testabgriff

Anhang B

Schematics

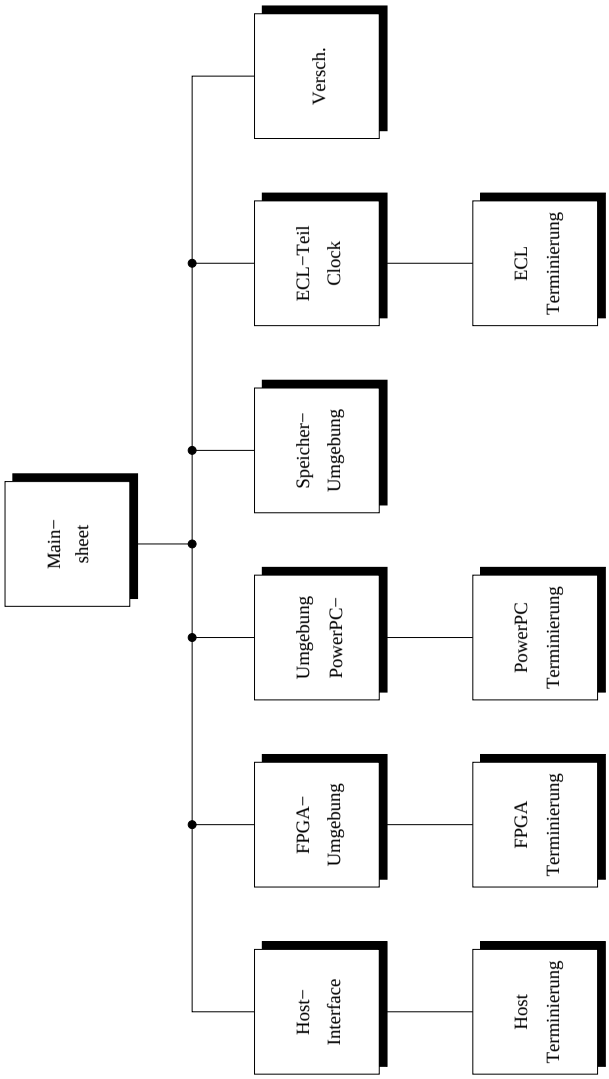

Abbildung B.1: Hierarchischer Aufbau der Platine

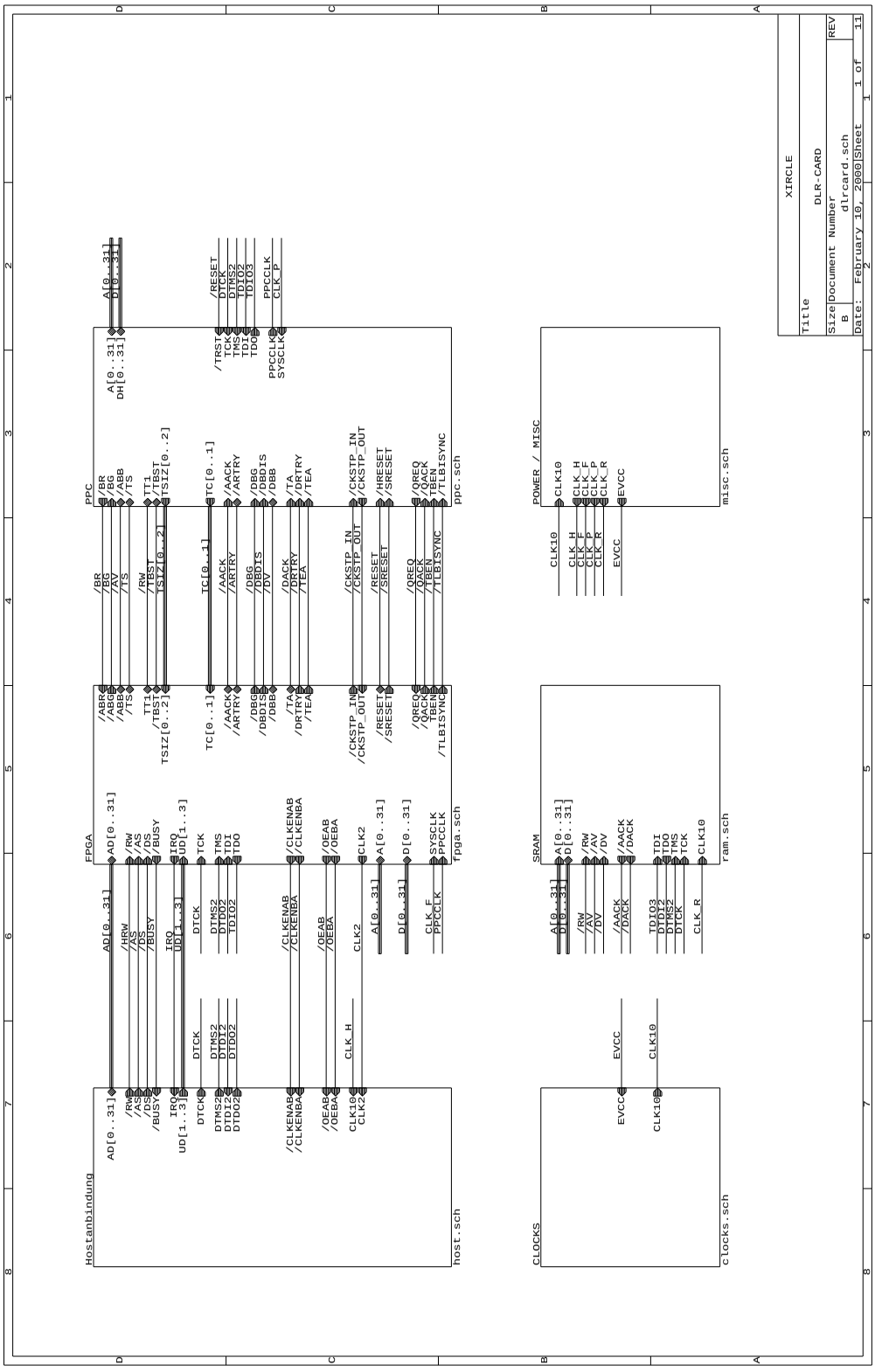

Abbildung B.2: DLR-CARD : Mainsheet

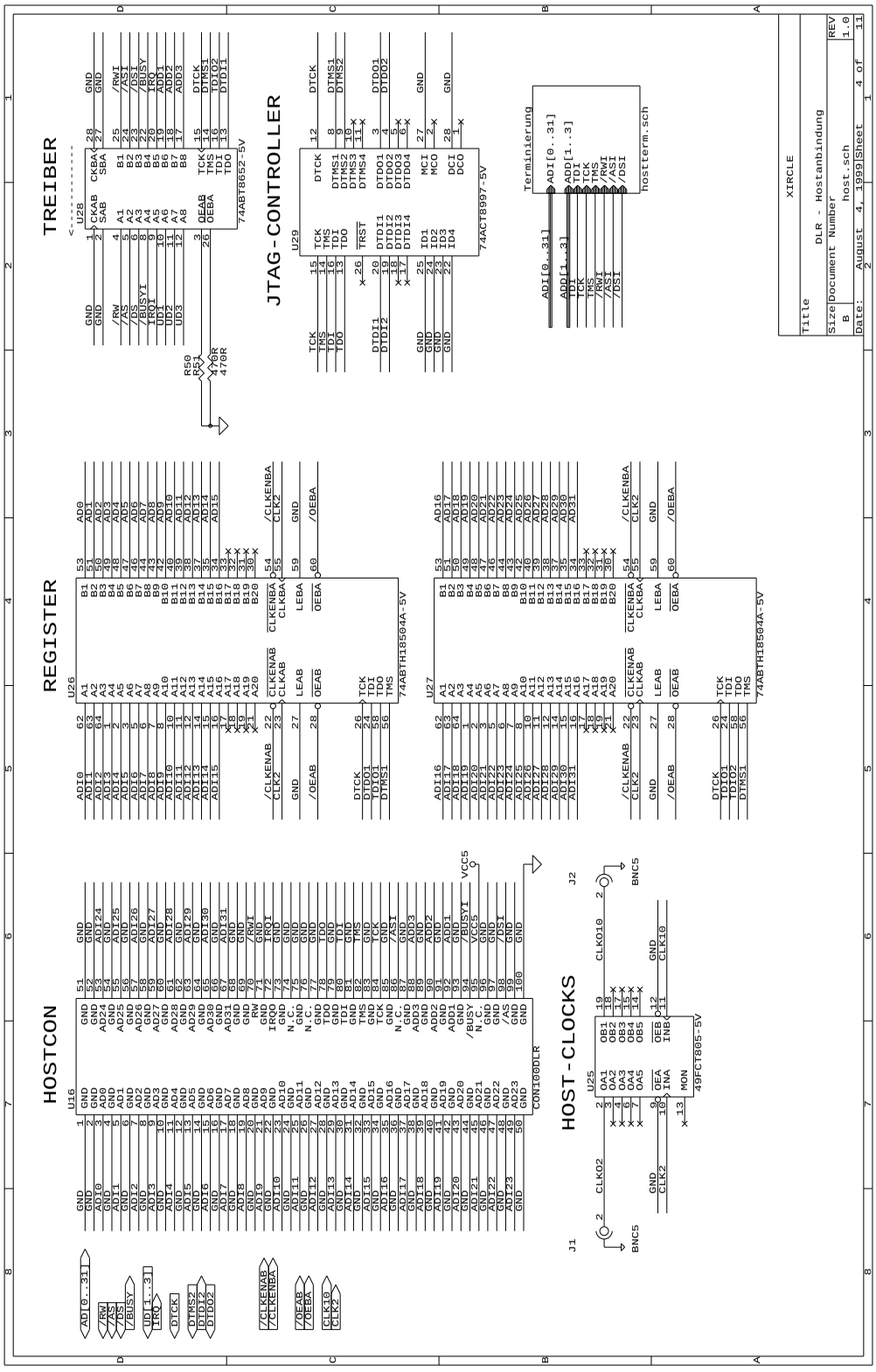

Abbildung B.3: DLR-CARD : Host-Interface

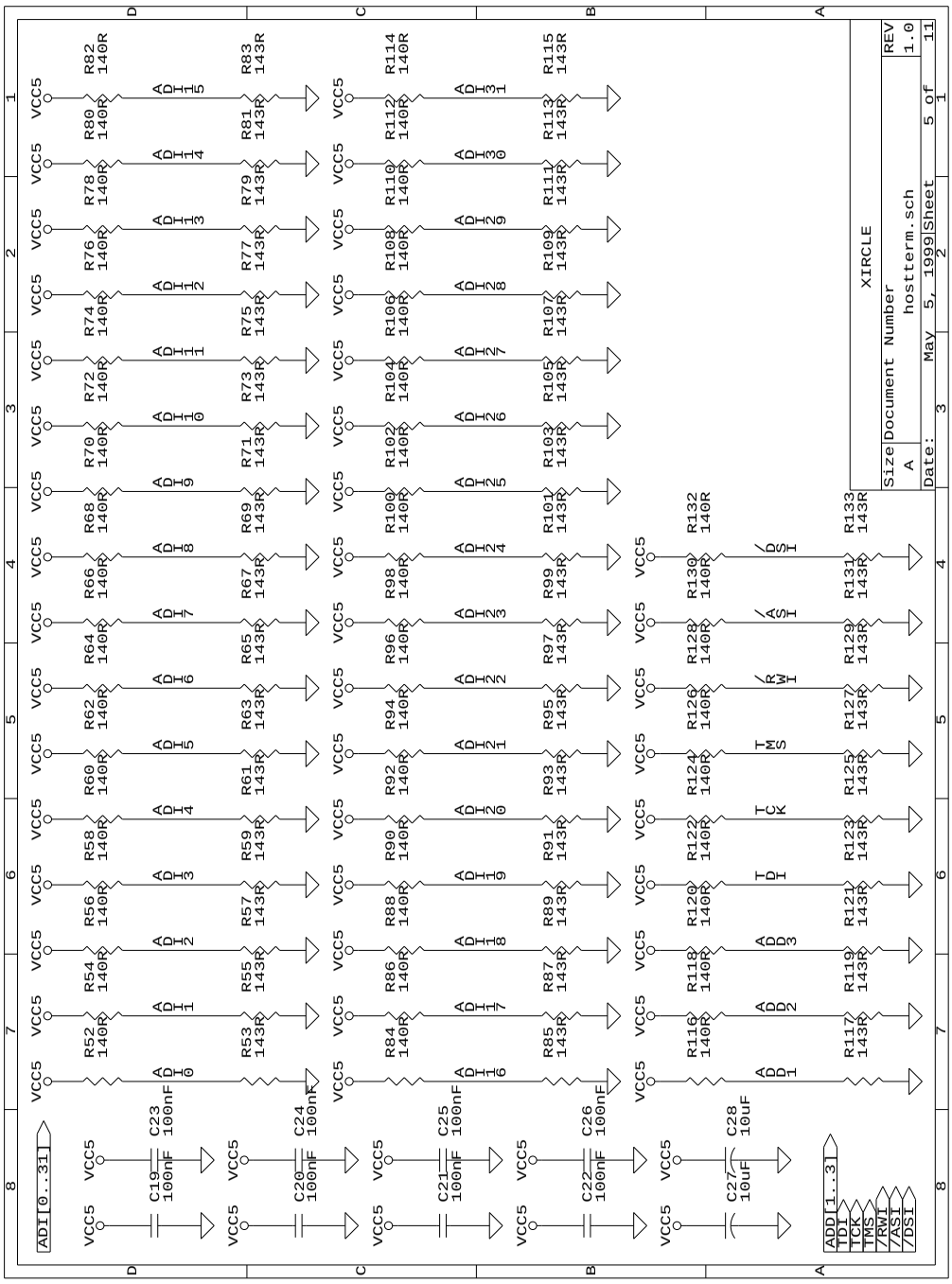

Abbildung B.4: DLR-CARD : Host Terminierung

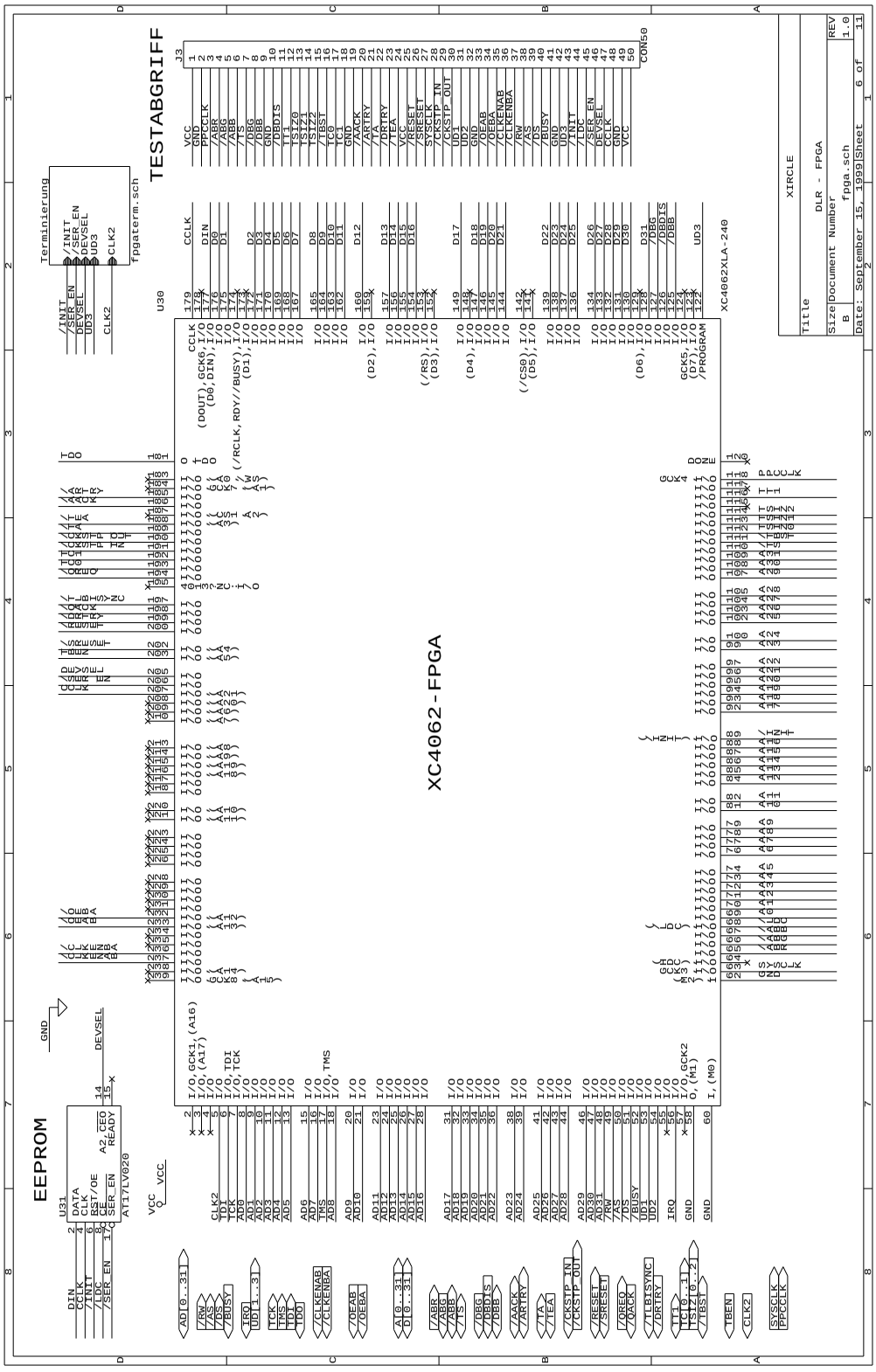

Abbildung B.5: DLR-CARD : FPGA-Umgebung

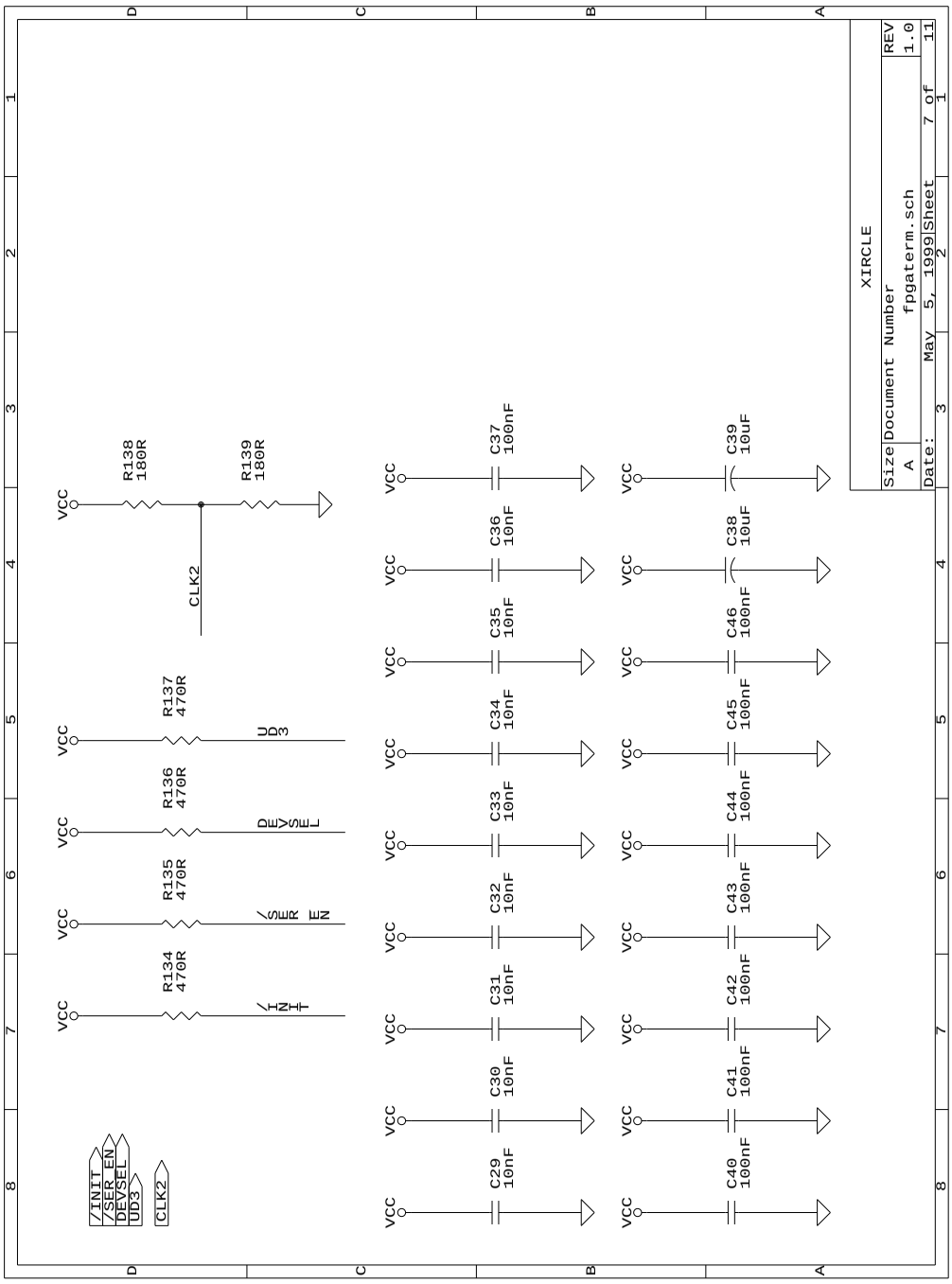

Abbildung B.6: DLR-CARD : FPGA Terminierung

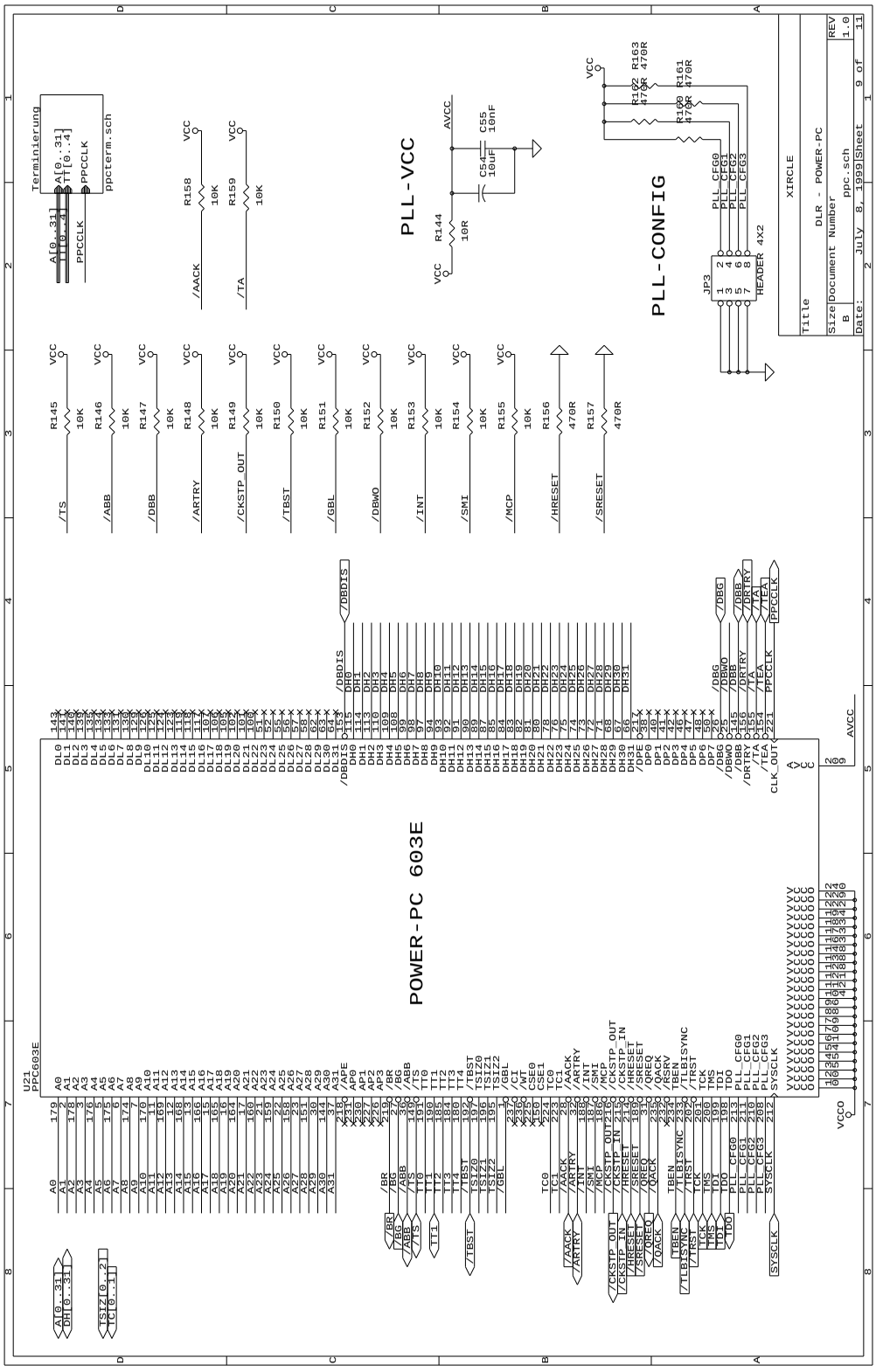

Abbildung B.7: DLR-CARD : PowerPC-Umgebung

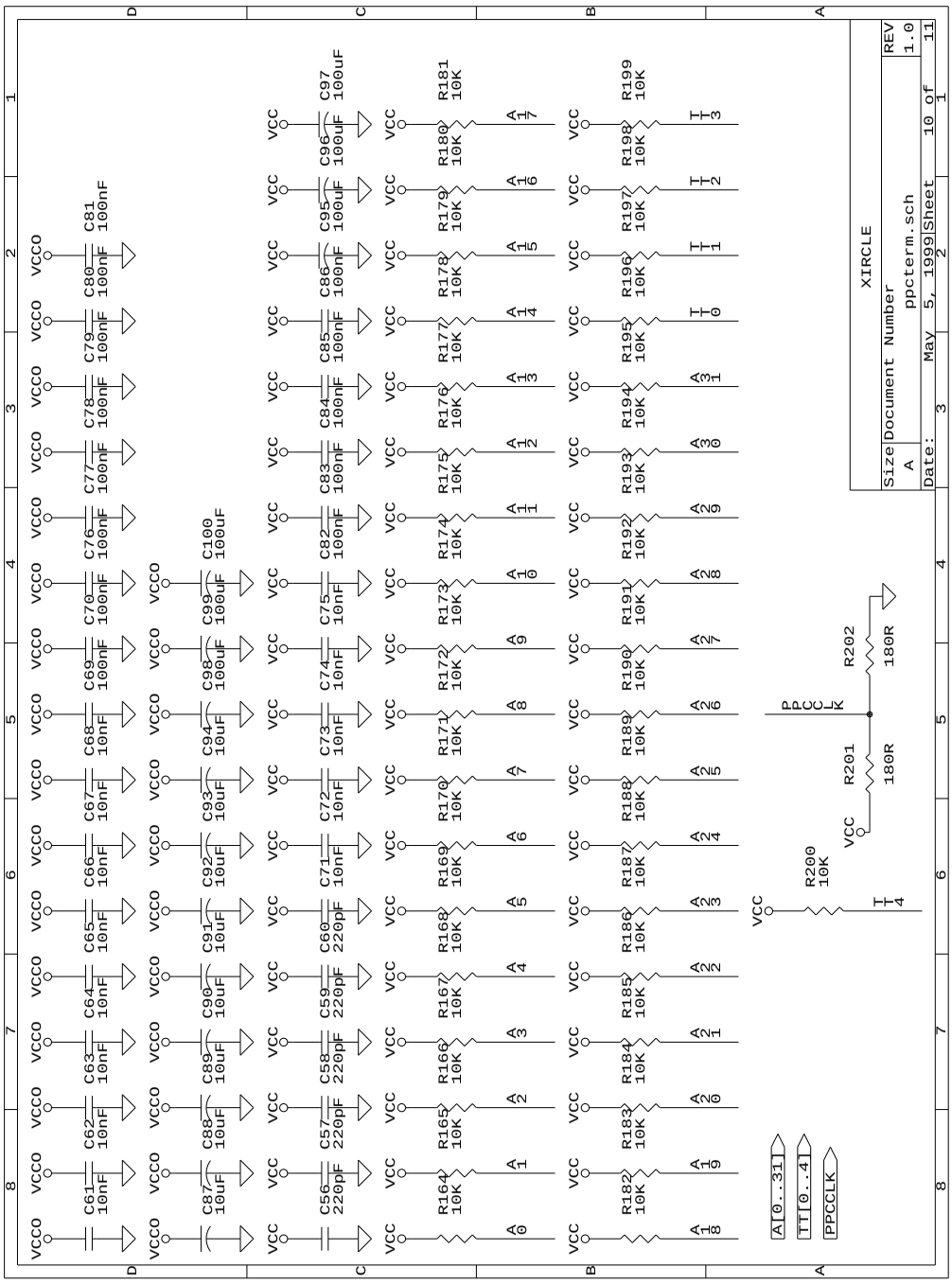

Abbildung B.8: DLR-CARD : PowerPC Terminierung

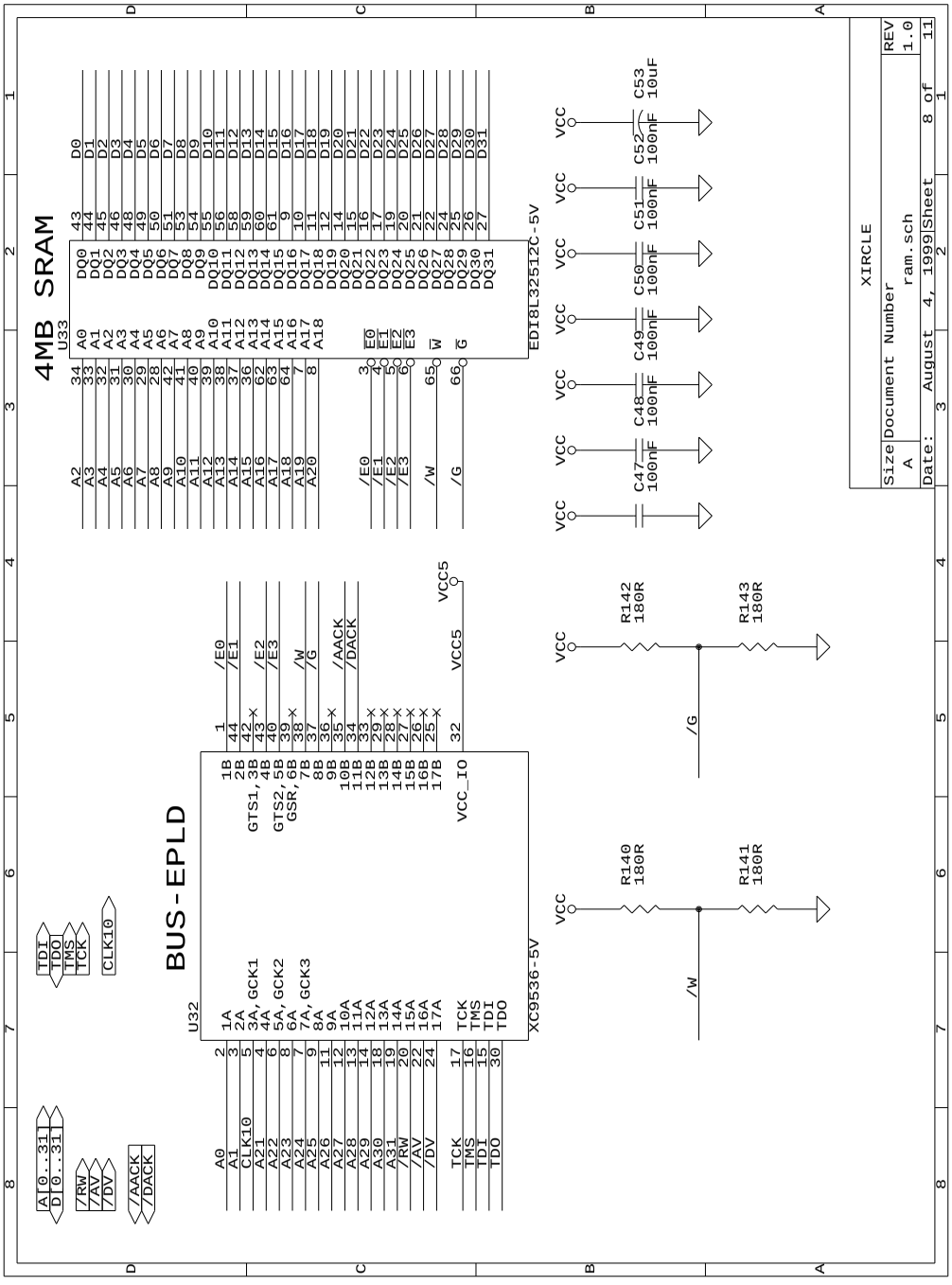

Abbildung B.9: DLR-CARD : Speicherumgebung

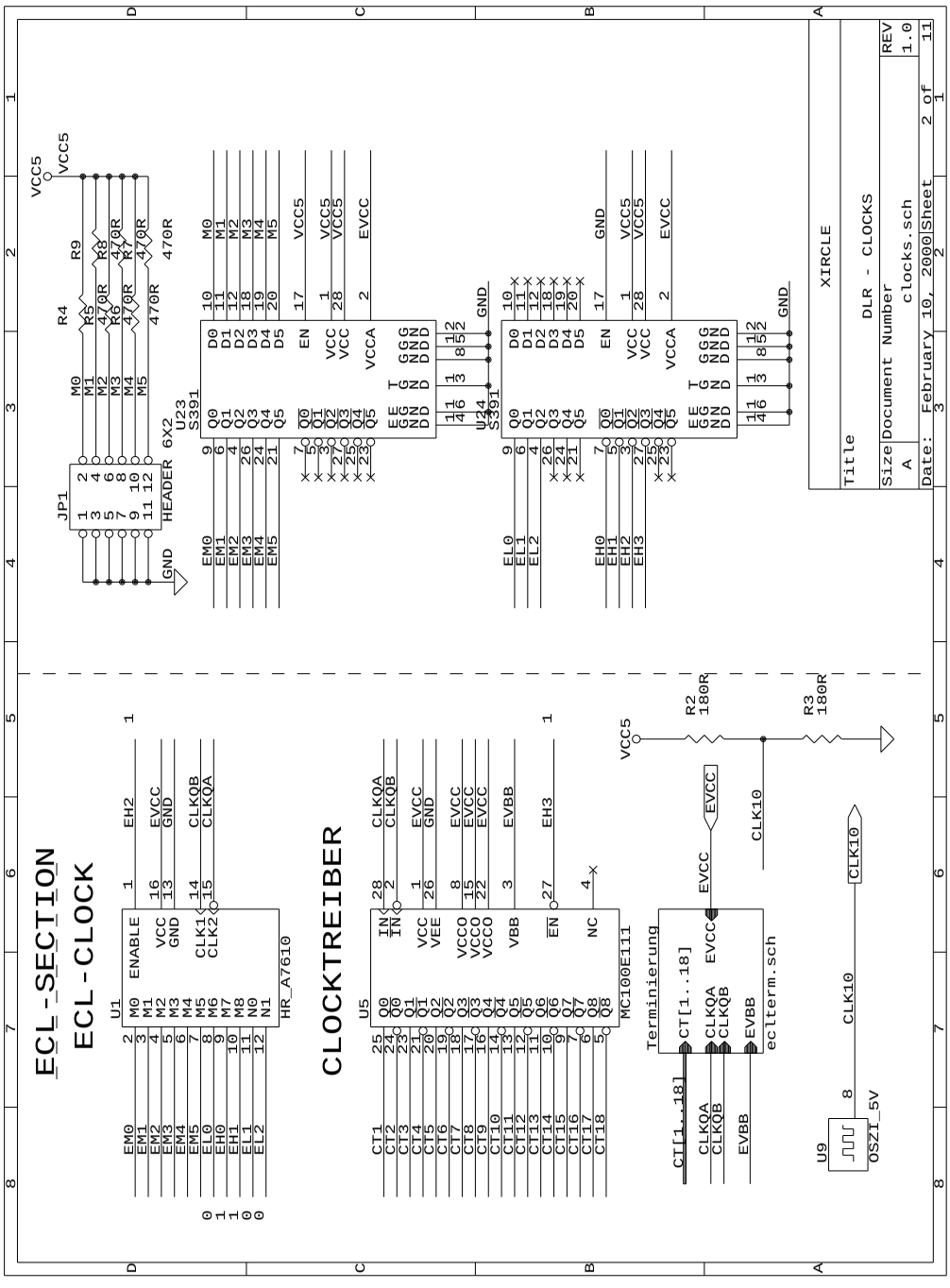

Abbildung B.10: DLR-CARD : ECL-Teil und Clockverteilung

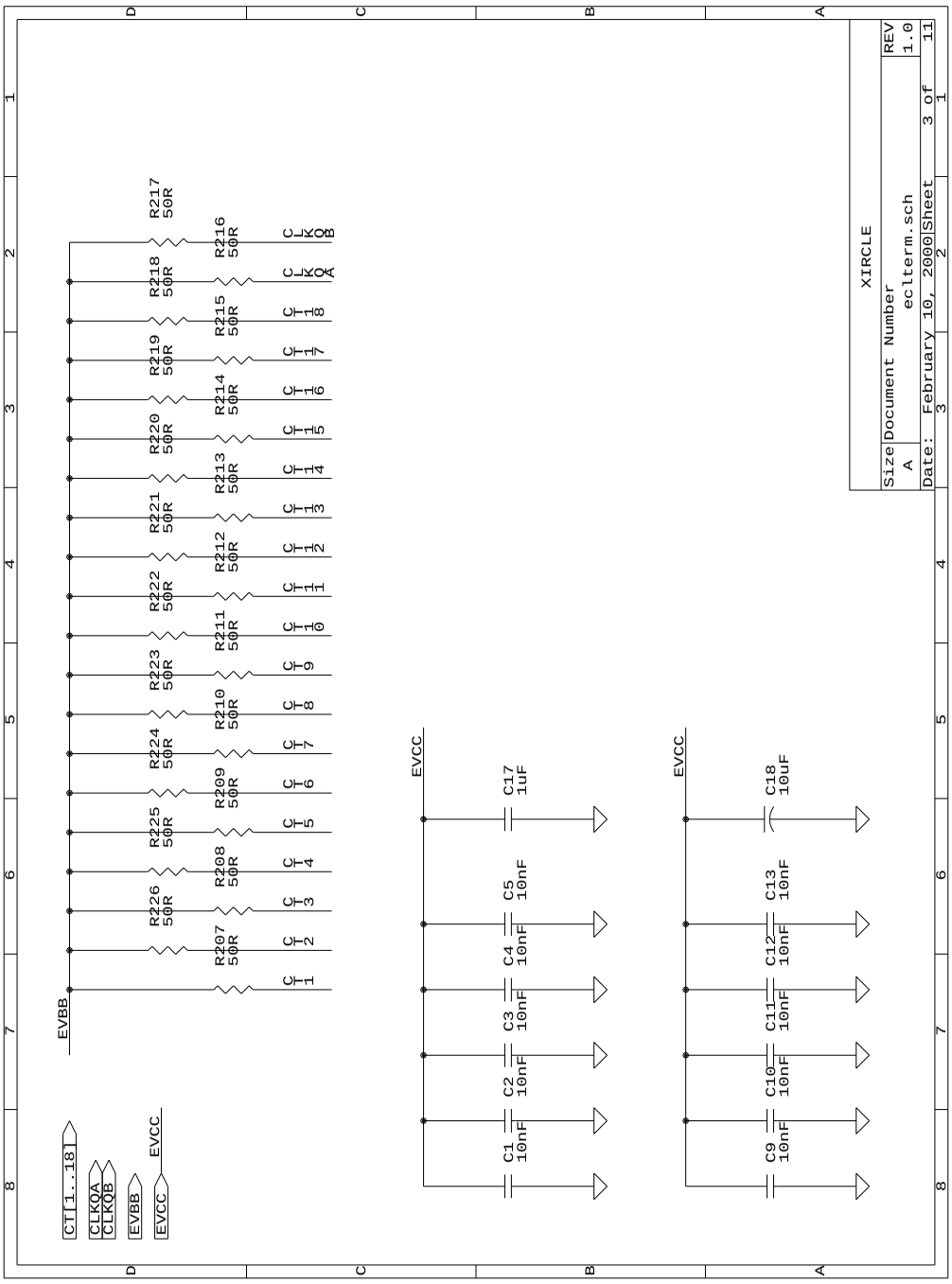

Abbildung B.11: DLR-CARD : ECL-Teil Terminierung

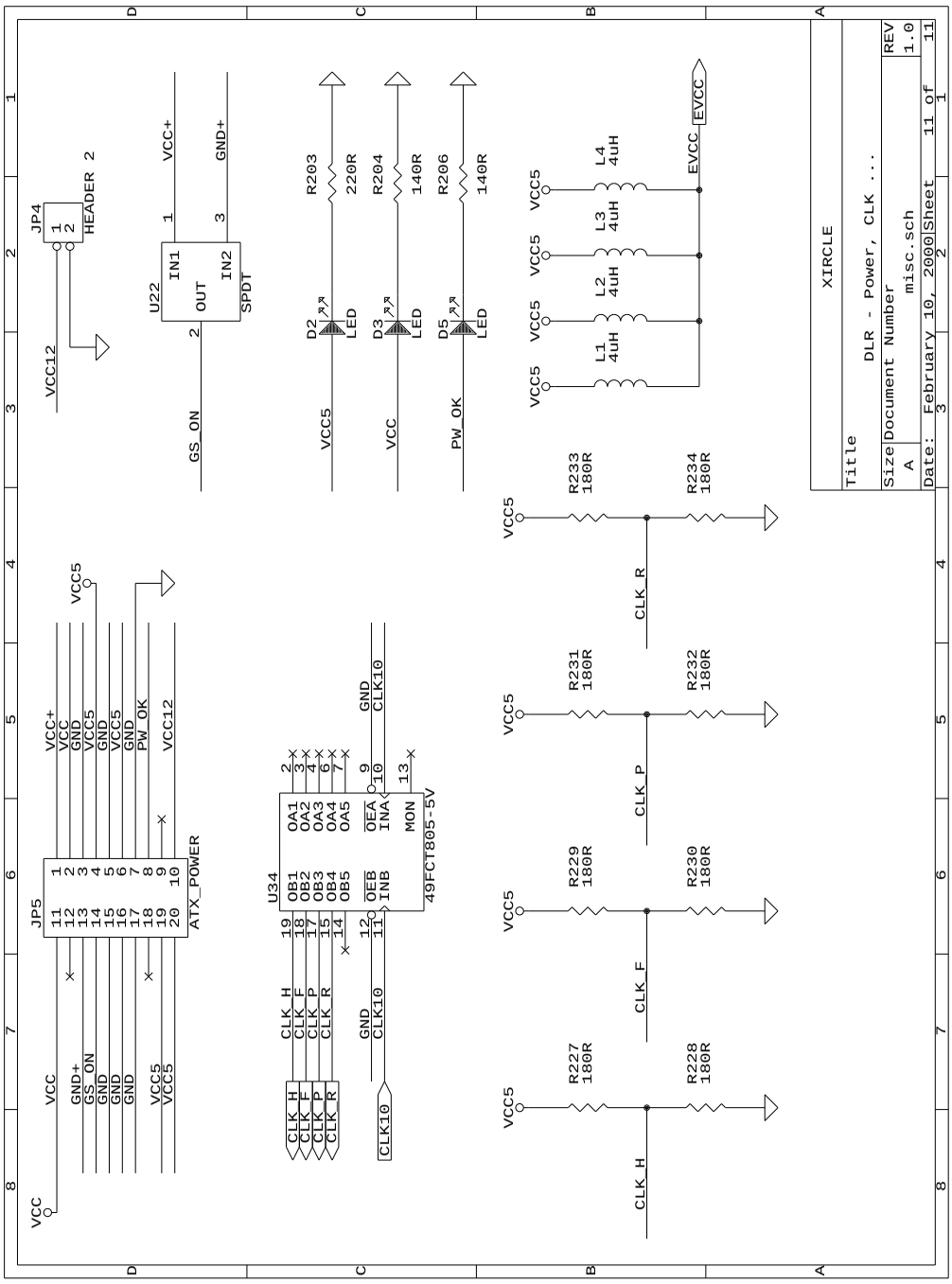

Abbildung B.12: DLR-CARD : Verschiedenes

Literaturverzeichnis

[1] Academy for Advanced Telecommunications. A Brief Introduction to ATM.
`http://academy.tamu.edu/atm/intro.html`.

[2] ACTEL. The Antifuse Advantage in FPGAs.
`http://www.actel.com/products/antifuse/`.

[3] ALDI Süd Supermarktkette. Sonderangebot, 2001.

[4] Peter Ashenden. *The Designer's Guide to VHDL.* Morgan Kaufmann, 1996.

[5] ATMEL. *FPGA Configuration EEPROM Programming Specifications*, 1999.

[6] Peter Bach. Schnelle Fertigungsfehleranalyse am Beispiel der Prozessorplatine CPULIGHT. Dissertation, Universität des Saarlandes, FR. 6.2 - Informatik, 2000.

[7] Torsten Brunklaus. Entwurf und Implementierung eines Interpretierers für JTAG-Testumgebungen. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1998.

[8] Deutsches Zentrum für Luft- und Raumfahrt. Institut für Technische Physik, Homepage.
`http://www.dlr.de/Stuttgart/institute/physik.htm`.

[9] Electronic Designs, Inc. *EDI8L32512C Data Sheet*, 1999.

[10] Jörg Fischer. Die Netzwerkplatine der SB-PRAM. Diplomarbeit, Universität des Saarlandes, FR. 6.2 - Informatik, 1999.

[11] R. Geels, S. Corzine, and L. Coldren. InGaAs vertical-cavity surface emitting lasers. *IEEE Quantum Electron.*, 27:1359–1367, 1991.

[12] K. Goossen, J. Walker, L. D'Asaro, et al. GaAs MQW Modulators Integrated with Silicon CMOS. *IEEE Phot. Tech. Lett.*, 7:360–362, 1995.

[13] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[14] IBM. *PowerPC 603e RISC Microprocessor Data Sheet*, 1997.

[15] IBM. *PowerPC 603e Embedded Hardware Specification: Advance Information*, 1998.

[16] Institute of Electrical and Electronics Engineers. *IEEE Standard 1149.1-1990, Standard Test Access Port and Boundary Scan Architecture*, 1993.

[17] Howard W. Johnson and Martin Graham. *High Speed Digital Design.* Prentice-Hall, 1993.

[18] Jörg Keller and Wolfgang J. Paul. *Hardware Design*, volume 15 of *Teubner-Texte zur Informatik.* Teubner, 1995.

[19] Michael Klein. JTAGDLX - Der Parser. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1998.

[20] Michael Klein, Wolfgang Paul, Jochen Preiß, Günther Renz, and Markus Scholl. Optical interconnect between cache and main memory. *LaserOpto*, 2001. to be published.

[21] Daniel Kröning. Cache-Simulationen für einen 32-bit RISC-Prozessor bei einer Standard-SPEC-Workload. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1997.

[22] Lehrstuhl für Rechnerarchitektur der Universität des Saarlandes. Fiberlink – Mainpage. `http://www-wjp.cs.uni-sb.de/projects/fiberlink`.

[23] Lucent Technologies. Process for Fabricating OE/VLSI Chips. `http://www.lucent.com/oevlsi`.

[24] Lennart Meier. JTAGDeLuXe - Beschreibung der BSDL-Funktionen. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1998.

[25] Mind Share, Inc. *PCI System Architecture*, 1995.

[26] Silvia M. Müller and Wolfgang J. Paul. *The Complexity of Simple Computer Architectures.* Springer, 1995.

[27] PLX Technology. *PCI Bus Interface and Clock Distribution*, 1996.

[28] Jochen Preiß. JTAGDLX - Einlesen und Verarbeitung der Netzlisten. FoPra-Dokumentation, Universität des Saarlandes, FB. Informatik, 1998.

[29] Jochen Preiß. Entwicklung eines optisch gekoppelten Cache. Diplomarbeit, Universität des Saarlandes, FR. 6.2 - Informatik, 2001.

[30] Steven A. Przybylski. *Cache and Memory Hierarchy Design.* Morgan Kaufmann Publishers, Inc., 1990.

[31] System Office of Telecommunication Services. The Gigabit Ethernet. `http://www.ots.utexas.edu/ethernet/gigabit.html`.

[32] Texas Instruments. JTAG Tutorial.
`http://www.ti.com/sc/docs/jtag/seminar1.pdf`.

[33] Texas Instruments. *49FCT805 product description*, 1996.

[34] Texas Instruments. *SN74ABT8652 product description*, 1996.

[35] Texas Instruments. *SN74ABTH18504A product description*, 1996.

[36] Texas Instruments. *SN74ACT8997 product description*, 1996.

[37] XILINX. The Home Page for Programmable Logic.
`http://www.xilinx.com`.

[38] XILINX. XC9536XV High-performance CPLD.
`http://www.xilinx.com/partinfo/ds053.pdf`.

[39] XILINX. *XC4000XLA/XV Field Programmable Gate Arrays*, 1999.

[40] XILINX. *The Foundation Manual*, 2000.