Universität des Saarlandes
Prof. Dr.-Ing. L. Thiele
Lehrstuhl für Mikroelektronik

Konzept und Realisierung eines zweidimensionalen programmierbaren Rechenfeldes mit XILINX FPGAs

Diplomarbeit von Tobias Blickle

Betreuer: Dipl.-Inform. Joachim König

Eingereicht im März 1993

Inhaltsverzeichnis

Bilderverzeichnis

Tabellenverzeichnis

Hiermit versichere ich an Eides statt, daß ich diese Arbeit selbst und nur mit den angegebenen Hilfsmitteln angefertigt habe.

Saarbrücken, den 17.3.1993

(Tobias Blickle)

1 Einführung

Viele Probleme der Datenverarbeitung, besonders im Bereich der Signalverarbeitung, erfordern heute Datenraten und Rechenleistungen, die von herkömmlichen sequentiellen Rechnern nicht bereitgestellt werden können. Zwar wurden in letzter Zeit immer wieder Leistungssteigerungen durch Vergrößern der Packungsdichte und Erhöhung der Taktfrequenz erreicht, aber hier sind bereits die physikalischen Grenzen der Miniaturisierung und Frequenzsteigerung absehbar. Selbst bei weiteren Leistungssteigerungen wären die Rechenleistungen für Echtzeitanwendungen wie Bilddatentransformation nicht ausreichend. Deshalb sucht man seit längerem nach anderen Möglichkeiten, die Leistungsfähigkeit von Rechnern zu erhöhen. Den Schlüssel dazu liefert die Parallelisierung, denn dadurch können von mehreren Prozessoren Teilaufgaben gleichzeitig gelöst werden, was eine deutliche Leistungssteigerung mit sich bringen kann.

Dabei werden zwei unterschiedliche Wege in der Ausführung paralleler Programme beschritten.

Eine Möglichkeit, die parallelisierten Probleme berechnen zu lassen, stellen frei programmierbare Parallelrechner dar. Dabei sind mehrere Prozessoren durch ein festes Netz verbunden. Die Prozessoren selbst arbeiten ein Programm sequentiell mit Hilfe eines Befehlssatzes ab, der auf die Probleme der parallelen Verarbeitung zugeschnitten ist. Spezielle Compiler passen den Algorithmus an die Hardware an. Sie sind dafür verantwortlich, wie gut die vorhandene Hardware genutzt wird. Diese Parallelrechner sind wegen ihrer freien Programmierbarkeit für unterschiedliche Probleme leicht anpaßbar. Ihr Parallelitätsgrad ist allerdings durch die Anzahl der vernetzten Prozessoren beschränkt.

Ein anderer Weg wird bei der Herstellung von algorithmisch spezialisierten parallelen Schaltungen gegangen. Dabei wird für jedes Problem ein spezieller Chip entwickelt. Dadurch lassen sich im allgemeinen noch höhere Verarbeitungsgeschwindigkeiten als bei Parallelrechnern erreichen, denn es können nicht nur die einzelnen Teilprozessoren parallel arbeiten, sondern jeder Prozessor kann intern parallel aufgebaut sein. Außerdem kann die Verbindungsstruktur der Prozessoren auf dem Chip für jeden Algorithmus angepaßt werden.

Beiden Verfahren ist zu eigen, daß die nötigen Umformungen am Algorithmus, um ihn an einen Parallelrechner oder an einen Chip anzupassen, sehr komplex sind. Wegen der einfachen Programmierbarkeit haben die Parallelrechner den Vorteil, daß das Ergebnis der Umformung schnell auf einem solchen Rechner verifiziert werden kann. Bei algorithmisch spezialisierten Schaltungen ist dagegen erst die Fertigung eines Prototyps nötig, wenn man nicht auf Simulationen

angewiesen sein will. Eine solche Chipherstellung dauert mehrere Wochen und ist sehr kostspielig.

Besonders im Bereich der Forschung und Entwicklung von parallelen Schaltungen ist deshalb ein System von Interesse, das wie die Parallelrechner einfach und beliebig oft programmiert werden kann, dessen Prozessoren sich aber selbst noch spezifizieren lassen, um die Parallelität optimal auszunutzen. So lassen sich die langen Wartezeiten und hohen Kosten, die durch die Fertigung eines Chip-Prototypen entstehen, umgehen. Außerdem wird es möglich, einen schnellen Überblick über die entwickelte Schaltung zu bekommen und sie frühzeitig in der Anwendungsumgebung zu testen.

Ein solches System stellen programmierbare Rechenfelder dar. Sie basieren auf FPGAs (Field-Programmable Gate-Array), die eine Weiterentwicklung der bekannten PLAs (Programmable Logic Array) darstellen. FPGAs sind nicht nur in der Lage, kombinatorische Schaltungen zu realisieren, sondern stellen auch intern Flipflops, programmierbare Verdrahtung und programmierbare Ein-/Ausgabepins zur Verfügung. Außerdem sind sie wie ein RAM beschreibbar, d.h. die momentane Konfiguration kann einfach überschrieben werden, ohne daß ein Löschen wie bei einem EPROM nötig wäre.

Mehrere dieser FPGAs sind bei einem programmierbaren Rechenfeld zu einem Netz verbunden. Die Verbindungen der Teilprozessoren (FPGAs) sind somit zwar starr vorgegeben, aber die FPGAs selbst lassen intern eine Parallelisierung und Spezialisierung zu.

Diese Vorteile werden bereits in mehreren Architekturen ausgenutzt, die unter dem Schlagwort “Rapid Prototyping” zusammengefaßt werden. Beispiele hierfür sind Splash [4], Anyboard [9] oder ein Prototyping-System für Steuerungsaufgaben [6].

Da der Lehrstuhl sich mit der Synthese algorithmisch spezialisierter Schaltungen befaßt, besteht ein Interesse an einem programmierbaren Rechenfeld. Die Entwicklung und Realisierung eines solchen Rechenfeldes erfolgt im Rahmen des Projekts PAr^2 (**P**rototyping **Ar**ray for **P**arallel **Ar**chitecture). Es soll an das Entwurfssystem COMPAR angeschlossen werden, das zur Abbildung von Algorithmen auf Schaltungen dient. Die damit erzeugten Algorithmen sollen auf diesem Rechenfeld abgearbeitet und getestet werden können.

Neben der Hardware des eigentlichen Rechenfeldes wird für den Einsatz eines solchen Systems ein Interface zum Anschluß an einen Hostrechner, sowie Software benötigt, die eine Programmierung des Rechenfeldes gestattet. Die Entwicklung und Realisierung der Hardware für PAr^2 war Aufgabe dieser Diplomarbeit.

2 Einbettung

Dieses Kapitel bettet das Projekt PAr^2 und diese Diplomarbeit in die Lehrstuhlarbeit ein.

Wie in der Einleitung kurz erwähnt, ist der Entwurf eines algorithmisch spezialisierten Chips sehr komplex. Es müssen diverse Transformationen durchgeführt werden, die den Algorithmus näher an eine Schaltung heranführen, die Semantik aber nicht verändern. Um einen Entwurf effizient zu halten, ist eine Computerunterstützung sinnvoll.

Normalerweise ist die Formulierung eines Algorithmus in sequenzieller Form gegeben. Ein erster Schritt, einer Schaltungsrealisation näherzukommen, ist deshalb die Parallelisierung des Algorithmus. Die Probleme, die mit einer Automatisierung dieser Parallelisierung zusammenhängen, sind ein Teil der Lehrstuhlarbeit.

Ein weiterer Schwerpunkt der Arbeiten ist, für eine Algorithmenklasse ein Entwurfssystem zu erstellen, das die notwendigen Schritte zur Abbildung eines (bereits parallelisierten) Algorithmus auf eine Schaltung durchführt, indem es dem Entwickler eine Reihe von Transformationstools zur Verfügung stellt.

Dabei beschränkt man sich auf eine spezielle Klasse von Algorithmen, die Problemstellungen aus dem Bereich der linearen Algebra behandeln, den sogenannten *stückweisen linearen Algorithmen* (Piecewise Linear Algorithms). Zu ihnen zählen Algorithmen der Signalverarbeitung wie FIR-Filter (Finite Response Filter), der Bildverarbeitung (z.B. Bilddatenkompression) oder direkte Anwendungen aus der linearen Algebra wie Matrizenmultiplikation.

Um die parallele Form des Algorithmus beschreiben zu können, wurde für diese Klasse von Algorithmen die Sprache LIRAN entwickelt, die auf der von Chandy und Misra entwickelten parallelen Programmnotation UNITY [2] aufbaut. Formal lassen sich diese Algorithmen wie folgt charakterisieren:

- Es existieren nur Anweisungen zwischen linear indizierten Variablen.
- Jeder dieser Anweisungen ist ein Gültigkeitsraum im Indexraum zugeordnet.
- Jede Variable steht nur einmal auf der linken Seite einer Gleichung (*Single Assignment* Bedingung).
- Es existiert eine zeitliche Reihenfolge der Anweisungen, so daß jeder Variablen, die auf der rechten Seite einer Anweisung auftaucht, vorher schon einmal ein Wert zugewiesen wurde (*Berechenbarkeit*).

2.1 COMPAR

Das Projekt COMPAR (Compiler for Massive Parallel Architectures) ist ein Entwurfssystem, um stückweise lineare Algorithmen auf Schaltungen abzubilden [1].

Ausgehend von einer Beschreibung des Algorithmus in LIRAN soll der Algorithmus aus der anfänglichen hardwarefernen Darstellung in eine hardwarenahe Darstellung gebracht werden. Dazu sind eine ganze Reihe von Transformationen nötig, die aber das Ein-/ Ausgangsverhalten des Algorithmus nicht verändern dürfen. Das Lokalisierungstool beispielsweise dient dazu, globale Datenabhängigkeiten in lokale umzuwandeln [10]. Globale Abhängigkeiten entsprächen langen Verbindungen auf dem Chip, die lange Signallaufzeiten und Platzprobleme mit sich brächten und deshalb unerwüscht sind. Weitere Beispiele für Transformationen sind die Generierung der Steuerung [12], die Zuordnung einer Zeitrichtung (*Scheduling*) oder die Partitionierung [13]. Die Parititionierung ist besonders im Hinblick auf das Rechenfeld von Interesse, denn sie ermöglicht es, den Algorithmus an die vorgegebenen Ressourcen anzupassen. Hierauf wird genauer in Kapitel 3.3 und 4.4 eingegangen.

Wurden alle erforderlichen Transformationen durchgeführt, ist beispielsweise eine Ausgabe des Algorithmus in einer hardwarenahen Beschreibungssprache (VLSI-OCCAM) möglich. Die Ausgabe in VLSI-OCCAM ermöglicht die Simulation des Algorithmus auf einem Transputer. Sie dient aber auch als Ausgangspunkt, um die endgültige Realisierung auf einem Chip zu ermöglichen. Die nötigen Anpassungen werden dabei von dem ebenfalls am Lehrstuhl entwickelten Transformationstool O2H (Occam to Hardware) vorgenommen.

Die Anbindung an das programmierbare Rechenfeld PAr^2 soll ebenfalls über eine hardwarenahe Beschreibungssprache erfolgen. Allerdings ist dazu ein weiterer Schritt notwendig, der den Algorithmus auf die Register-Transfer-Ebene (Register-Transfer-Level, RTL) umformt.

Da COMPAR zur Zeit noch in Entwicklung ist, sind noch nicht alle angesprochenen Transformationen verfügbar. Implementiert sind bisher nur die Datenein- und -ausgabe in LIRAN, die Occamausgabe und die Lokalisierung.

2.2 PAr^2

Das Projekt PAr^2 soll ein programmierbares Rechenfeld zur Verfügung stellen, um die mit COMPAR erzeugten Algorithmen in Echtzeit ablaufen lassen zu können.

Dazu muß auf der Hardwareseite eine ausreichende Rechenkapazität - also genügend Gatteräquivalente bei möglichst hoher Taktfrequenz - vorhanden sein.

Die Hardware muß schnell, einfach und beliebig oft programmierbar sein. Debuggingmöglichkeiten sollten vorgesehen sein. Ebenso sollte auf dem Rechenfeld lokal ausreichend RAM vorhanden sein, um speicherintensive Algorithmen realisieren zu können, ohne Zwischenergebnisse zwischen Host und Rechenfeld hin- und hertransportieren zu müssen.

Neben dem eigentlichen Rechenfeld muß eine Anbindung an das Rechnersystem (Sparcstations) über ein Interface erfolgen. Dazu dient das PSI-Interface (**P**rogrammable **S**Bus **I**nterface), das hostseitig an den SBus der Sparcstations angeschlossen wird. Die andere Seite des Interfaces stellt eine beliebig programmierbare Schnittstelle zu Verfügung, die nicht nur zum Anschluß von PAr^2 verwendet werden kann, sondern auch für andere Projekte des Lehrstuhls verwendet werden soll (siehe [8]). Die Programmierung dieser Schnittstelle für PAr^2 wurde in der Diplomarbeit von Patrik Knapp ausgeführt [7].

Als weiterer Schritt zur vollständigen Anbindung eines Rechenfeldes an COMPAR muß softwareseitig die Lücke zwischen RTL-Beschreibung des Algorithmus und der Gatterebene geschlossen werden, auf der PAr^2 arbeitet.

Ist die vollständige Anbindung des Rechenfeldes an COMPAR gelungen, so stellt PAr^2 eine flexible Hardware zur Verfügung, um parallele Algorithmen abzuarbeiten. Neben der Verifizierung von Chipentwürfen wäre in einem nächsten Schritt die Einbindung dieser Hardware über Funktionsaufrufe am Host denkbar, die automatisch den entsprechenden Algorithmus im Rechenfeld programmieren und abarbeiten. Auf diese Weise ließe sich PAr^2 als ein "Coprozessor für parallele Algorithmen" an konventionellen Rechnersystemen einsetzen.

2.3 Aufgabenstellung der Diplomarbeit

Im folgenden Kapitel wird zunächst das Grundkonzept von PAr^2 vorgestellt, das in enger Zusammenarbeit mit Prof. Dr.-Ing. Lothar Thiele, Dipl.- Inform. Joachim König sowie Patrik Knapp entstanden ist.

In Kapitel 4 wird der erste Themenschwerpunkt der Diplomarbeit erläutert, der sich mit den gewählten Kommunikationsmethoden des Rechenfeldes befaßt. Neben der Beschreibung dieser Kommunikation wird ein Registermodell der Hardware vogestellt. Dieses Modell ist besonders für COMPAR wichtig, da es die Restriktionen vorgibt, die von jedem Algorithmus erfüllt werden müssen, damit er auf der Hardware lauffähig ist. Mit Hilfe dieses Registermodells und der Einführung von einheitlichen Signalbezeichnungen innerhalb der FPGAs wird es auch möglich, eine Beschreibung der Kommunikation abstrahiert von der tatsächlichen physikalischen Realisation zu finden.

Der zweite Themenschwerpunkt wird in Kapitel 5 behandelt und stellt die Hardware von PAr^2 vor und geht auf die Probleme bei der Realisation ein.

3 Konzept von PAr^2

In diesem Kapitel soll nun das Konzept von PAr^2 dargelegt werden. Nach kurzer Vorstellung der verwendeten FPGA-Bausteine und des Interface werden die Anforderungen an das Rechenfeld zusammengefaßt. Im Anschluß wird daraus das Konzept von PAr^2 entwickelt.

3.1 Xilinx FPGAs

Das gesamte Konzept von PAr^2 basiert auf den XILINX FPGAs der XC4000 Serie [15]. FPGA steht für Field-Programmable Gate Array, frei übersetzbar mit "programmierbarem Logikchip". Mit ihnen lassen sich die Vorteile von VLSI Schaltungen nutzen, ohne jedoch für jede Anwendung einen speziellen Chip konventionell herstellen zu müssen. Die Chips werden programmiert, indem man die Konfigurationsdaten in den internen Speicher lädt. Dies kann entweder vom Chip selbständig aus einem EPROM erfolgen oder durch Laden eines Bitstroms von einem Host aus. Sind die Daten geladen, kann die Schaltung mit einer Taktfrequenz von bis zu 50-60 MHz betrieben werden. Dies gilt allerdings nur bei geringer Auslastung des FPGAs und entsprechend optimaler Verdrahtung; Praxiswerte bei hoher Chipauslastung und komplexen Routing liegen eher bei 10–20 MHz. Da ein FPGA beliebig oft programmiert werden kann, stellen sie die idealen Bausteine zur Realisierung eines programmierbaren Rechenfeldes dar.

Die FPGAs lassen sich in drei Grundblöcke aufteilen: Ein- /Ausgabe-Blöcke (Input-Output-Blocks, IOBs), konfigurierbare Logikblöcke (Configurable Logic Blocks, CLBs) und programmierbare Verbindungen (programmable Interconnect). Die CLBs sind regelmäßig in einem quadratischen Feld auf dem Chip verteilt. An den Rändern des Chips befinden sich die IOBs. In den Zwischenräumen liegen die Verdrahtungskanäle mit den Verbindungsleitungen, an die die CLBs und IOBs angeschlossen werden können (Bild 1).

Die IOBs bilden die Schnittstelle zwischen Chip und Außenwelt. Jedem frei verfügbarem Datenpin ist ein IOB zugeordnet, der den Pin als Eingang oder Ausgang konfigurieren kann. Da der Ausgang über einen Tristate-Treiber hochohmig geschaltet werden kann, sind auch bidirektionale Pins realisierbar. Außerdem können sowohl im Eingangs-, als auch im Ausgangspfad ein Register zwischengeschaltet werden (vgl. Bild 2).

Die CLBs stellen die eigentliche Schaltungslogik zur Verfügung. Sie besteht im wesentlichen aus zwei programmierbaren Funktionsgeneratoren F und G mit jeweils vier Eingängen und einem Ausgang (vgl. Bild 3). Sie bestimmen ihren Funktionswert über eine Look-Up-Table, d.h. über ein vier Bit breites und

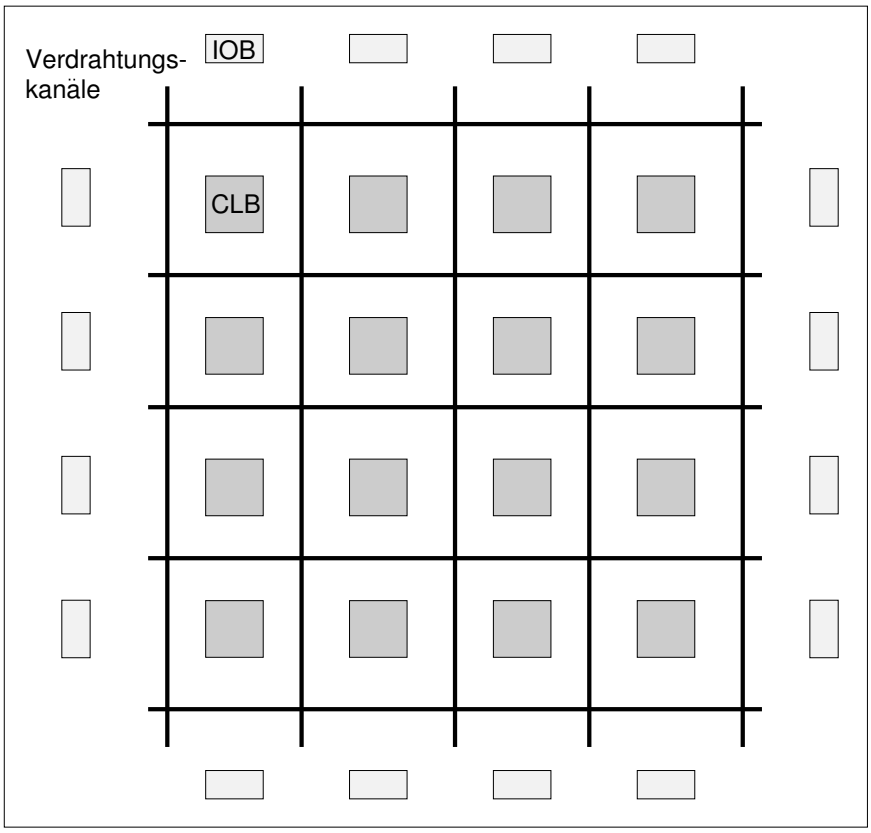

Bild 1 Schematischer Aufbau eines XILINX FPGA

ein Bit tiefes ROM, in dem der Funktionswert zu jeder Eingangskombination gespeichert ist. Die Ausgänge der Funktionsgeneratoren können zusammen mit einem weiteren externen Signal als Eingänge für den Funktionsgenerator H dienen. Außerdem befinden sich in jedem CLB zwei Flipflops, deren Eingänge von den Ausgängen der Funktionsgeneratoren F, G oder H oder direkt von dem externen Signal C2 gespeist werden können. Als Ausgänge eines CLBs existieren die Ausgänge der Flipflops und zwei weitere Leitungen, die mit den Ausgängen der Funktionsblöcke F, G und H verbunden werden können.

Alle Flipflops der FPGA übernehmen die Daten mit steigender Taktflanke.

Neben dieser Grundfunktion verfügt jeder CLB über eine Fast Carry Logic, sowie über eine Möglichkeit, die Funktionsgeneratoren F und G als 16*1 Bit RAM zu verwenden. Auf diese Möglichkeiten soll hier nicht weiter eingegangen werden, da sie für das Verständnis der Arbeitsweise von PAr^2 nicht erforderlich sind. Genauere Informationen lassen sich in [15] nachlesen.

Das dritte Grundelement der FPGAs stellen die programmierbaren Verbindungen dar. Es existieren dabei unterschiedliche Typen von Verbindungsleitungen: Die "Single-Length Lines" verbinden direkt benachbarte CLBs, während "Double-Length Lines" immer ein CLB überspringen. Für größere Entfernung

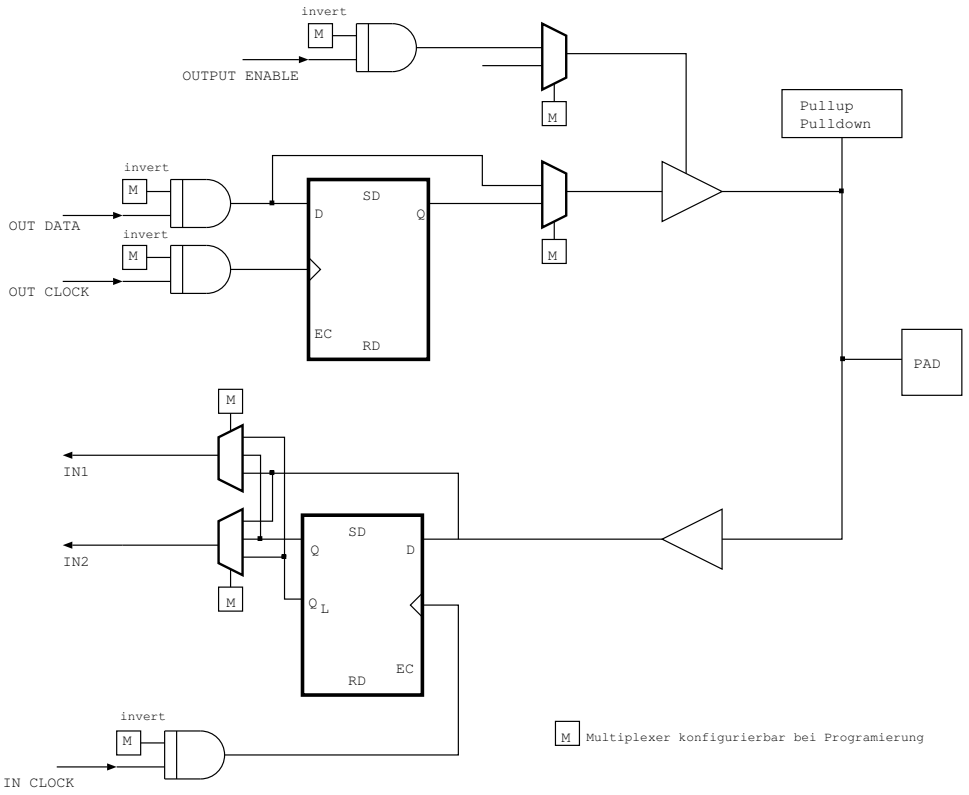

Bild 2 Vereinfachter Input-/Output Block (IOB) der FPGAs.

sind die "Global Longlines" prädestiniert, die mit Tristate-Treibern angesteuert werden können. “Global Nets” sind für die Verteilung von Taktsignalen konzipiert. Sie sind mit besonderen internen Treibern versehen (Global Primary Buffer und Global Secondary Buffer) und erlauben es, zeitkritische Signale auf dem Chip zu verteilen, da sie zu jedem Flipflop im Chip eine garantierte Laufzeit unter fünf Nanosekunden haben. Insgesamt stehen acht solcher Leitungen pro Chip zur Verfügung; eine dieser Leitungen wird zum Verteilen des globalen Taktes verwendet. Die Verbindungen zwischen den unterschiedlichen Leitungen erfolgen über sogenannte "Switch Matrices".

Der Programmiervorgang besteht nun darin, die Funktion der einzelnen Funktionsblöcke der CLB festzulegen, sowie die Art und Weise wie ihre Ausgänge verschaltet sind. Ebenso werden beim Programmieren die Konfiguration der IOBs geladen und die Verbindungen zwischen den CLBs festgelegt, d.h. die Switch Matrices programmiert.

Als Besonderheit bieten die XILINX FPGAs die Möglichkeit, den internen Zustand auszulesen. Dies geschieht über das READBACK. Zu einem beliebigen Zeitpunkt kann mittels des Signals RDBK.TRIG das FPGA veranlaßt werden, die Daten mit der durch RDBK.CLK vorgegebenen Geschwindigkeit über den Pin RDBK.DATA auszugeben. Dabei werden nicht nur die Daten über die

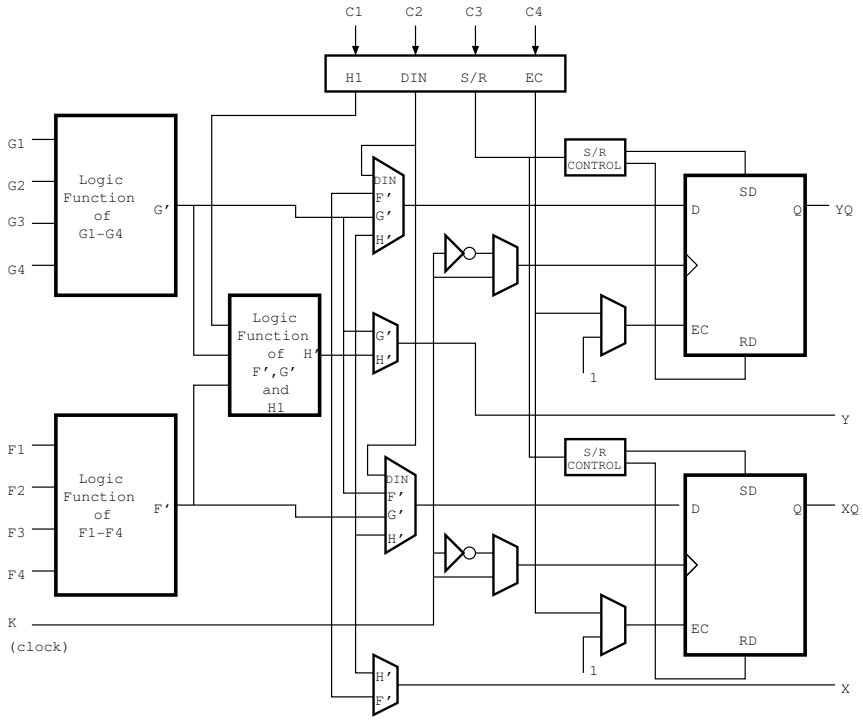

Bild 3 Configurable Logic Block (CLB) der FPGAs.

programmierte Funktion übertragen, sondern auch der Zustand der Register in den IOBs und CLBs.

Beim Aufbau von PAr^2 wurden FPGAs vom Typ XC4005 verwendet. Sie verfügen über eine Gatteräquivalent von ca 5000 in 14 * 14 = 196 CLBs. In einem 156poligen Grid-Array stellen sie 112 frei verfügbare IOPins und ebensoviele IOBs zur Verfügung. Insgesamt befinden sich in den CLBs und IOBs zusammen 616 Flipflops. Durch Ausnutzung der Funktionsgeneratoren als Speicher können maximal 6272 Bit RAM auf einem Chip realisiert werden.

3.2 Interface

Das Interface, das am Lehrstuhl entwickelt wird, soll die Möglichkeit bieten, unterschiedlich parallele algorithmische und semialgorithmische Prozessoren an SPARC-Stations anzuschließen. Bild 4 zeigt das Prinzipschaltbild. Über einen Buscontroller ist ein FPGA an den SBus der SPARC-Station angeschlossen, das die Kommunikation mit dem Host steuert. Zwei getrennte 32 Bit breite FIFOs — eins für die Lese-, eins für die Schreibrichtung — puffern die zu übertragenden Daten, die an zwei weitere FPGAs geführt werden. Von jedem dieser FPGAs sind 60 Pins auf eine Steckerleiste gelegt, deren Bedeutung frei programmierbar ist, je nach der in den FPGAs realisierten Schaltung. Es stehen also insgesamt 120 Pins zur Verfügung.

In seiner momentanen Realisierung erreicht das Interface eine Übertragungsrate von maximal 8 MByte/s.

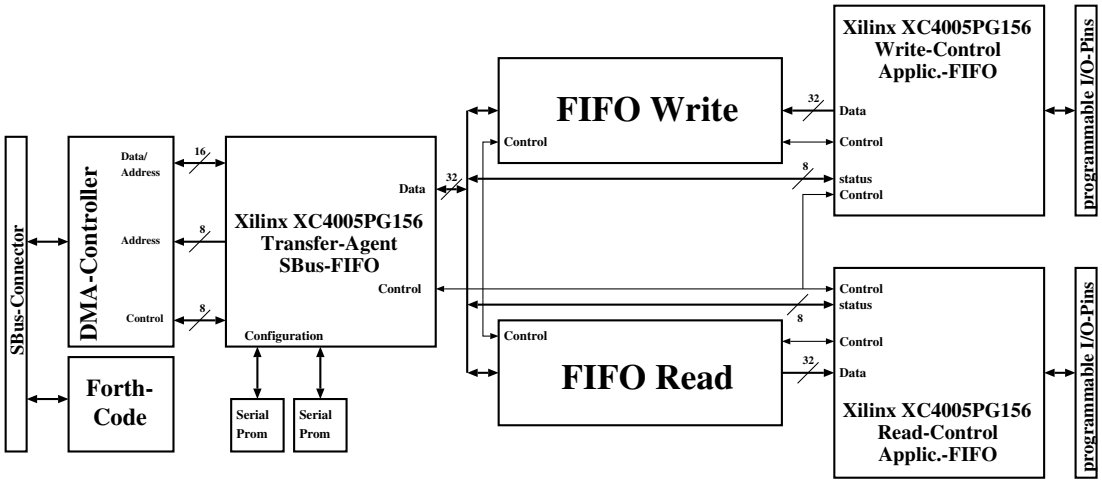

Bild 4 Blockschaltbild des PSI-Interface

3.3 Anforderungen

Das Rechenfeld soll für den *implementierten Algorithmus*[1] sozusagen das Gehäuse zur Verfügung stellen, d.h. mehrere FPGAs und das RAM sowie die Datenein- und -ausgabe so verfügbar machen, daß möglichst wenig Ressourcen der FPGAs verloren gehen. Die Möglichkeiten, die sich durch die Transformationstools in COMPAR ergeben, sollten gewinnbringend in der Konzeption eingesetzt werden.

Insgesamt ergeben sich folgende Anforgerungen an das Rechenfeld:

- Es muß eine ausreichende Anzahl von Gatteräquivalenten zur Verfügung stellen.
- Die Anordnung der FPGAs als Teilprozessoren soll regelmäßig sein. Ziel der Chipentwicklung ist es, eine regelmäßige Realisierung zu erreichen.
- Die einzelnen FPGAs sind über lokale Busse miteinander verbunden. Lokale Verbindungen direkt benachbarter FPGAs sind ausreichend, da mit dem Lokalisierungstool von COMPAR globale Datenabhängigkeiten im *implementierten Algorithmus* aufgelöst werden können. Über diese Verbindungen gelangen sowohl die Rechendaten als auch die Steuerdaten, die Information über den momentanen Zustand im *implementierten Algorithmus* repräsentieren. Zwischen diesen Daten besteht kein qualitativer Unterschied, denn durch

[1] Im weiteren bezeichnet der kursiv gedruckte Ausdruck *implementierter Algorithmus*, die Schaltung (Algorithmus), die auf dem Rechenfeld ablaufen soll.

die Generierung einer Steuerung mittels COMPAR lassen sich Steuersignale wie normale Daten behandeln.

- Die Kommunikation mit dem Host erfolgt am Rand des Rechenfeldes. Dies ergibt sich praktisch direkt aus der Lokalität des Algorithmus. Außerdem erlaubt eine Chiprealisierung des Algorithmus nur Anschlüsse am Rand der Chipfläche. Entsprechend ist es sinnvoll, bei einem Rechenfeld analog zu verfahren.
- Es sollte möglichst viel RAM auf dem Rechenfeld vorhanden sein, um Daten speichern zu können, ohne den Umweg über den Host nehmen zu müssen, was viel Zeit in Anspruch nehmen würde. Die von den FPGAs von XILINX zur Verfügung stehenden ca. 6k RAM sind nicht ausreichend für speicherintensive Algorithmen. Außerdem ist dieses RAM nur gegen Verzicht auf Logikblöcke erhältlich, d.h. es bleiben für die Kombinatorik der Schaltung weniger Gatteräquivalente übrig.
  Hauptverwendungszweck des RAMs wird das Speichern von Daten sein, die durch die Partitionierung anfallen. Eine Partitionierung ist deshalb nötig, weil der *implementierte Algorithmus* in den seltensten Fällen genau auf das Rechenfeld "paßt", d.h. der *implementierte Algorithmus* muß auf die durch das Rechenfeld vorgegebenen Ressourcen abgebildet werden [13].
  Die Partitionierung besteht in einem Aufteilen des Algorithmus in mehrere Teilalgorithmen, wobei zwei grundsätzliche Methoden unterschieden werden können. Arbeiten alle Prozessoren des Rechenfeldes parallel an einem Teilalgorithmus, und die Teilalgorithmen werden sequentiell nacheinander abgearbeitet, spricht man von einer Partitionierung nach dem LPGS-Verfahren (locally parallel, globally sequential). Hierbei wird wird zum Speichern der Zwischenergebnisse Speicher am Rand des Prozessorfeldes benötigt, um Daten über mehrer Teilalgorithmen hinweg auszutauschen. Die zweite Methode ist die LSGP-Partitionierung (locally sequential, globally parallel): Hier arbeitet jeder Prozessor einen Teilalgorithmus sequentiell ab, alle Teilalgorithmen aber werden parallel ausgeführt. Dabei benötigt jeder Prozessor seinen eigenen Speicher, um die Daten des Teilalgorithmus zu speichern.
  Für eine LSGP-Partitionierung müßte jedem FPGA eigenes RAM zur Verfügung gestellt werden. Damit würden viel Ressourcen des FPGAs verbraucht, denn es werden sowohl viele Pins für die Adressierung und den Datentransfer zum RAM benötigt, als auch CLBs für eine Ansteuerschaltung des RAMs. Da die Kommunikation mit dem Host sowieso eine Sonderbehandlung am Rand des Rechenfeldes erfordert, geht man davon aus, daß nach dem LPGS-Verfahren partitioniert wird und somit der Speicher ebenfalls an den Rand gelegt werden kann.
- Das Rechenfeld soll zweidimensional sein. Diese Vorgabe liegt hauptsächlich in der dadurch erreichbaren höheren Parallelität begründet. Außerdem

sind zweidimensionale Rechenfelder seltener vertreten und somit weniger gut untersucht als eindimensionale Anordnungen.

- Die Kommunikation erfolgt synchron. Der Grund hierfür ist der wesentlich geringere Hardwareaufwand einer synchronen Kommunikation gegenüber einer asynchronen, die zusätzliche Quittierungsleitungen benötigt. Die Nachteile der synchronen Kommunikation sind die Taktverschiebung bei großen Entfernungen, sowie die Stromspitzen durch das gleichzeitige Schalten der Elemente.
- Selbstverständlich soll das Rechenfeld beliebig oft programmierbar sein.
- Debuggingmöglichkeiten sollten so weit wie möglich genutzt werden, da bei einem experimentellen System jede Form der Überprüfbarkeit willkommen ist.
- Wünschenswert ist weiterhin eine weitestgehende Modularität der Hardware, um sich die Option einer späteren Erweiterbarkeit des Systems offen zu halten.

3.4 Grundüberlegungen

Aus diesen Anforderungen an PAr^2 ist das folgende Grundkonzept entstanden.

Das Rechenfeld besteht aus einem quadratischen 3x3 Feld von XILINX XC4005 FPGAs, die zusammen eine Gatterkapazität von 45000 Gattern haben. Die FPGAs sind lokal über Busse miteinander verbunden und am Rand torusmäßig geschlossen. Bei 112 I/O Pins der FPGAs und vier benötigten Bussen ergibt sich eine theoretische Busbreite von 28 Bit. Da SRAMS in Datenbusbreiten von 8 Bit oder Vielfachen davon üblich sind, wurde als Busbreite 24 Bit gewählt.

Die Verwaltung des benötigten Speichers am Rand sowie die Kommunikation mit dem Host erfolgt über FPGAs mit besonderen Steueraufgaben, die in die Rückführungen der Rechenfeldverbindungen eingeschleift sind (*Ramcontroller*-FPGAs). Sie haben damit die Möglichkeit, die Daten, die das Rechenfeld austauscht, "mitzuhören" und zu manipulieren. Da in diesen Daten auch Steuerinformationen enthalten sind, können sie so eine gewünschte Aktion ausführen, also z.B. Daten an den Host schicken bzw. vom Host holen oder Daten ins RAM schreiben bzw. vom RAM lesen.

Die Aufgaben des Ramcontrollers sollen näher untersucht werden.

Um Daten mit dem Host auszutauschen, ist an jedes Ramcontroller-FPGA das Interface über den IFBUS (Interface-Bus) angeschlossen, der sowohl eine Lese-, als auch eine Schreibrichtung hat. Dies liegt darin begründet, daß das Interface bereits getrennt Richtungen für Lesen und Schreiben vorsieht. Da es pro Rechenfeldzeile und Rechenfeldspalte jeweils ein Ramcontroller FPGA gibt, sind an den IFBUS mehrere FPGAs parallelgeschaltet. In Leserichtung (vom Interface zu den Ramcontrollern) ergibt sich daraus kein Problem, da nur das

Interface schreibend auf diesen Teilbus zugreift. In Schreibrichtung sieht das anders aus, da hier jedes FPGA einen möglichen Datensender darstellt. Dadurch wird es nötig, eine Buskontrolle einzuführen, die garantiert, daß immer nur ein Teilnehmer auf den Bus zugreift.

Dafür wurden die unterschiedlichsten Verfahren in Erwägung gezogen. So ist z.B. eine übergeordnete Buskontrolle durch das Interface denkbar, das die Ramcontroller der Reihe nach abfrägt und ihnen so die Möglichkeit gibt, ihre Daten abzusenden (Polling). Eine andere Idee wäre, jedem Ramcontroller eine "Request-Leitung" zum Interface zu legen, mit dem es ihm anzeigt, daß es Daten senden möchte. Nach Zuteilung der Erlaubnis kann der Ramcontroller den Bus belegen und seine Daten zum Interface schicken. Dieses Verfahren benötigt so viele Leitungen am Interface wie es Ramcontroller gibt.

Der Nachteil an diesen Verfahren ist aber, daß der Ramcontroller so seine Daten zwischenspeichern muß, da einige Zeit vergehen kann, bis er den Bus zugeteilt bekommt. Ein Zwischenspeichern könnte nur entfallen, wenn das Interface innerhalb eines Taktzyklusses des Ramcontrollers den "Sender" ausmachen und die Daten übertragen kann. Dies wiederum setzt voraus, daß das Interface (bei sechs Ramcontrollern) mindestens sechsmal schneller läuft als der Ramcontroller. Zusätzlich müßte garantiert werden, daß nicht ein zweiter Ramcontroller zur selben Zeit Daten sendet, sonst müßte einer von beiden wiederum seine Daten speichern.

Würde man sich also notgedrungen auf eine Zwischenspeicherung der Daten einlassen und eine gewisse Zwischenspeichergröße festlegen, müßte man sich immer noch überlegen, was passierte, wenn dieser Speicher voll wäre. Dann müßte der Ramcontroller das Rechenfeld anhalten können, bis wieder Platz in seinem Speicher wäre. Damit wären zwei Taktleitungen zu den Ramcontrollern nötig: eine, die die Kommunikation mit dem Rechenfeld steuern würde und angehalten werden könnte und eine weitere, die die Kommunikation zwischen Ramcontroller und Interface regeln würde.

Man sieht, daß diese Überlegungen zu immer neuen Problemen führen. Deswegen wurde ein anderes Konzept gewählt, das die Möglichkeiten von COMPAR ausnutzt: Um die aufwendige Busarbitrierungsmaßnahmen zu vermeiden, wird davon ausgegangen, daß jeder Ramcontroller den Zeitpunkt "weiß", wann er den Bus benutzen darf. Dieses Wissen ist in COMPAR bereits vorhanden, denn die Arbeitszeitpunkte der Prozessoren sind bekannt und somit auch die Zeitpunkte, an denen die Ergebnisse der Berechnungen anfallen. Durch Wahl einer Zeitrichtung in COMPAR (*Schedule*) ist eine Transformation des Algorithmus so möglich, daß immer nur ein Ramcontroller zu einer Zeit den Bus benutzt. Dadurch ist die Buskontrolle in eine höhere Ebene delegiert. Es genügt so, jedem Ramcontroller-FPGA ein Output-Enable Signal für seine Bustreiber zur Verfügung zu stellen. Auf der Gegenseite muß das Interface wissen, wann Daten auf dem Bus gültig

sind. Denn es ist durchaus denkbar, daß mehrere Taktzeitpunkte kein Ramcontroller Daten zu senden hat. Dieses Problem wird in [7] diskutiert. Prinzipiell wäre es auch möglich, alle Daten einfach an den Host weiterzugeben, und diesen erst die Unterscheidung treffen zu lassen, ob die Daten gültig oder ungültig sind. Dies ist aber eine ungünstige Lösung, da so viele unnötige Daten zum Host transportiert würden und der Bus zwischen Interface und Host, der sowieso schon den Flaschenhals in der Kommunikation darstellt, zusätzlich belastet würde.

Man benötigt nun zwar immer noch einen Mechanismus, um das gesamte Rechenfeld mit Ramcontroller anhalten zu können, aber jetzt kann dies das Interface allein entscheiden. Denn der einzig denkbare Auslöser dafür ist das Überlaufen des Schreib-FIFOS bzw. das Leerlaufen des Lese-FIFOs im Interface. Außerdem besteht bei dieser Lösung die Möglichkeit, andere Arbitrierungsverfahren nachträglich durch Programmieren der FPGAs zu realisieren, indem man einige Leitungen des IFBUS für diese Zwecke reserviert.

Das RAM ist ebenfalls an den Ramcontroller angeschlossen und soll pro Taktzyklus sowohl einen Lese- als auch einen Schreibzugriff gestatten. Dies erlaubt es beispielsweise gleichzeitig Daten vom RAM zum Interface zu senden und Daten vom Rechenfeld in das RAM zu schreiben.

Insgesamt ergibt sich aus diesen Überlegungen das in Bild 5 am Beispiel eines 2x2–Feldes gezeigte Prinzipschaltbild.

Wie dort zu ersehen ist, müssen an ein Ramcontroller-FPGA die Busse zweier Rechenfeld-FPGAs, Adreß- und Datenbus des RAMs sowie der IFBUS zur Kommunikation mit dem Interface angeschlossen werden. Die verwendeten XC4005 FPGAs stellen wie erwähnt 112 I/O-Pins zur Verfügung. Die Ramcontroller benötigen bei einer Busbreite von 24 Bit und 64k Adreßraum insgesamt 6*24+2*16=176 Pins. Es sollten aber keine größeren Chips der XC4000–Serie verwendet werden, da diese erheblich teurer sind. Der Preis steigt hierbei etwa quadratisch mit der Anzahl der zur Verfügung stehenden CLBs. Nicht unerheblich ist auch das geometrische Problem, das im Layout entsteht, wenn so viele Leitungen gezogen werden müssen. Es wird auch zunehmend schwieriger, ein Design zu routen, je größer der Chip ist, da die Ressourcen an Verbindungsleitungen zwischen den CLBs konstant sind.

Um also den Leitungsbedarf am Ramcontroller zu vermindern, wird ein Multiplexen auf allen Bussen des Ramcontroller-FPGAs eingeführt. Man faßt dazu die Lese- und Schreibleitungen eines Busses zusammen und benötigt dadurch nur noch die halbe Anzahl der Leitungen, muß das Lesen und Schreiben dafür nacheinander innerhalb eines Taktzyklus auf denselben Leitungen durchführen.

Die Datenrate auf den Bussen wird dadurch doppelt so hoch wie die Taktrate von PAr^2. Dies begründet auch den Nachteil des Multiplexens: Da sich die maximale Taktrate im wesentlichen nach der höchsten vorkommenden Datenrate in der

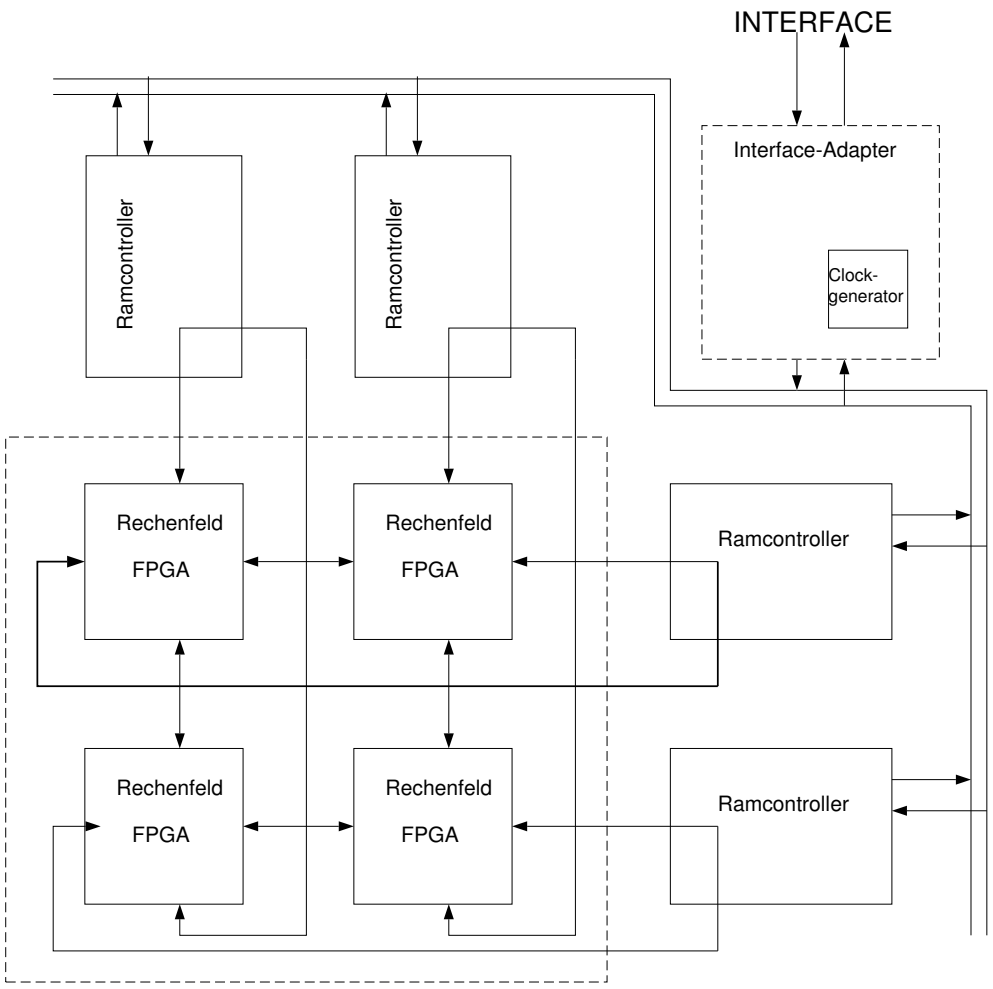

Bild 5 Prinzipschaltbild von PAr^2 — Kommunikation

Schaltung richtet, führt es zu einer Halbierung der Taktrate des gesamten Rechenfeldes. Da man auf diese Weise aber eine Halbierung der Anzahl der Leitungen am Ramcontroller erreicht, ist dieser Nachteil tolerierbar. Dies gilt insbesondere dann, wenn während der Laufzeit des Algorithmus große Datenmengen vom oder zum Host transportiert werden müssen. In einem solchen Fall nützt eine höhere Taktrate nämlich wenig, da die Datenübertragungsrate vom Host bzw. Interface bestimmt wird. Bei höherer Verarbeitungsgeschwindigkeit des Rechenfeldes müßte das Rechenfeld nur um so öfter angehalten werden, da der Host nicht mehr nachkommt, die Daten abzunehmen bzw. bereitzustellen. Außerdem können als RAM nun konventionelle SRAMs eingesetzt werden, die erheblich preisgünstiger und in größeren Speicherkapazitäten als Dual-Port RAM erhältlich sind.

Das in Bild 5 gezeigte Prinzipschaltbild stellt also nicht die physikalische Wirklichkeit dar, wie die Komponenten von PAr^2 verbunden sind. Es zeigt vielmehr das “Prinzip” in dem Sinne, wie die Hardware für den *implementierten*

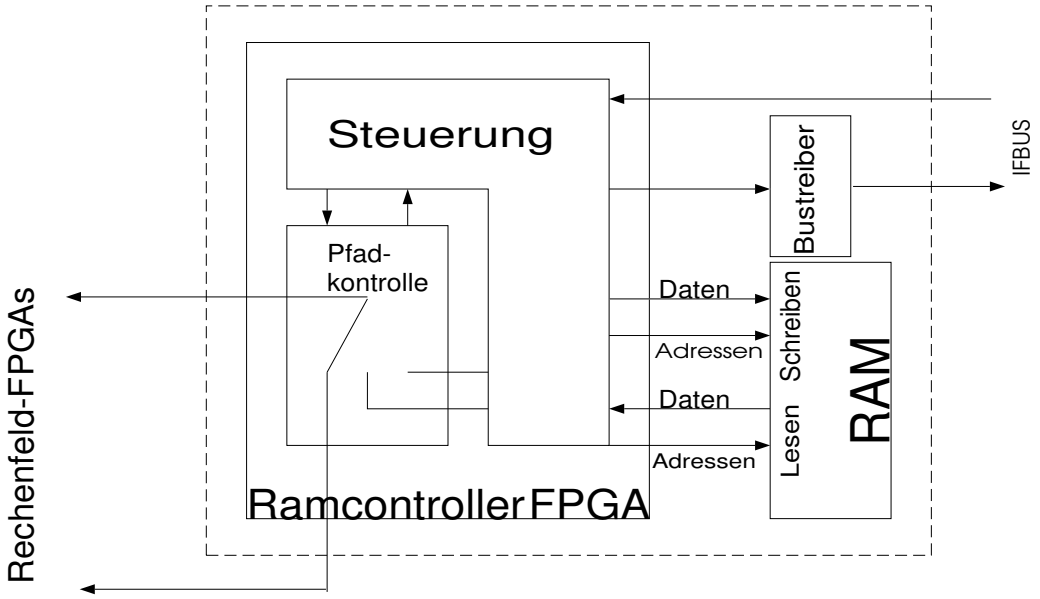

Bild 6 Prinzipschaltbild Ramcontroller zu Bild 5

Algorithmus aussieht. Es ist somit ein “virtuelles” Prinzipschaltbild im Gegensatz zu den physikalischen Prinzipschaltbildern zur Kommunikation im folgenden Kapitel (Bild 9, 11 und 13).

Zusätzlich zu diesen prinzipiellen Komponenten gibt es einen globalen Bus (General Purpose Bus, GPBUS), an dem alle FPGAs des Rechenfeldes und die der Ramcontroller angeschlossen sind (siehe Bild 7). Obwohl dieser globale Bus dem eigentlichen Konzept der Lokalität des Algorithmus widerspricht, wurde er eingeführt, um keine Leitungen an dem FPGA ungenutzt zu lassen. Denn dadurch, daß die Busbreite auf 24 Bit festgesetzt wurde, sind abzüglich reservierter Leitungen 10 Leitungen an jedem FPGA frei. Außerdem sind Anwendungen denkbar, in denen dieser Bus sinnvoll genutzt werden kann. Da dieser Bus auch zu den sieben nicht benötigen Global Primary bzw. Secondary Buffer (siehe Kapitel 3.1) führt, können hiermit beispielsweise zusätzliche Taktsignale verteilt werden.

Zur Programmierung von PAr^2 müssen die Programmdaten an die einzelnen FPGAs geleitet werden und sichergestellt werden, daß nur ein FPGA gleichzeitig programmiert wird. Dies erfolgt über die Zuordnung einer Adresse zu jedem FPGA, das nur dann an die Programmierleitungen angeschlossen wird, wenn die ensprechende Adresse anliegt. Die Programmierung erfolgt dann über einen Befehl an das Interface, das die Daten in das entsprechende FPGA leitet.

Um einen Algorithmus debuggen zu können, wurde die Möglichkeit des READBACKS der XILINX FPGAs auch in PAr^2 vorgesehen. Mit demselben Mechanismus wie bei der Programmierung kann ein FPGA angesprochen werden und der interne Zustand der Register im Chip ausgelesen werden. Die Leitungen

zum Programmierung und Debuggen der FPGAs sind zu dem PRBUS (Programm & Readbackbus) zusammengefaßt.

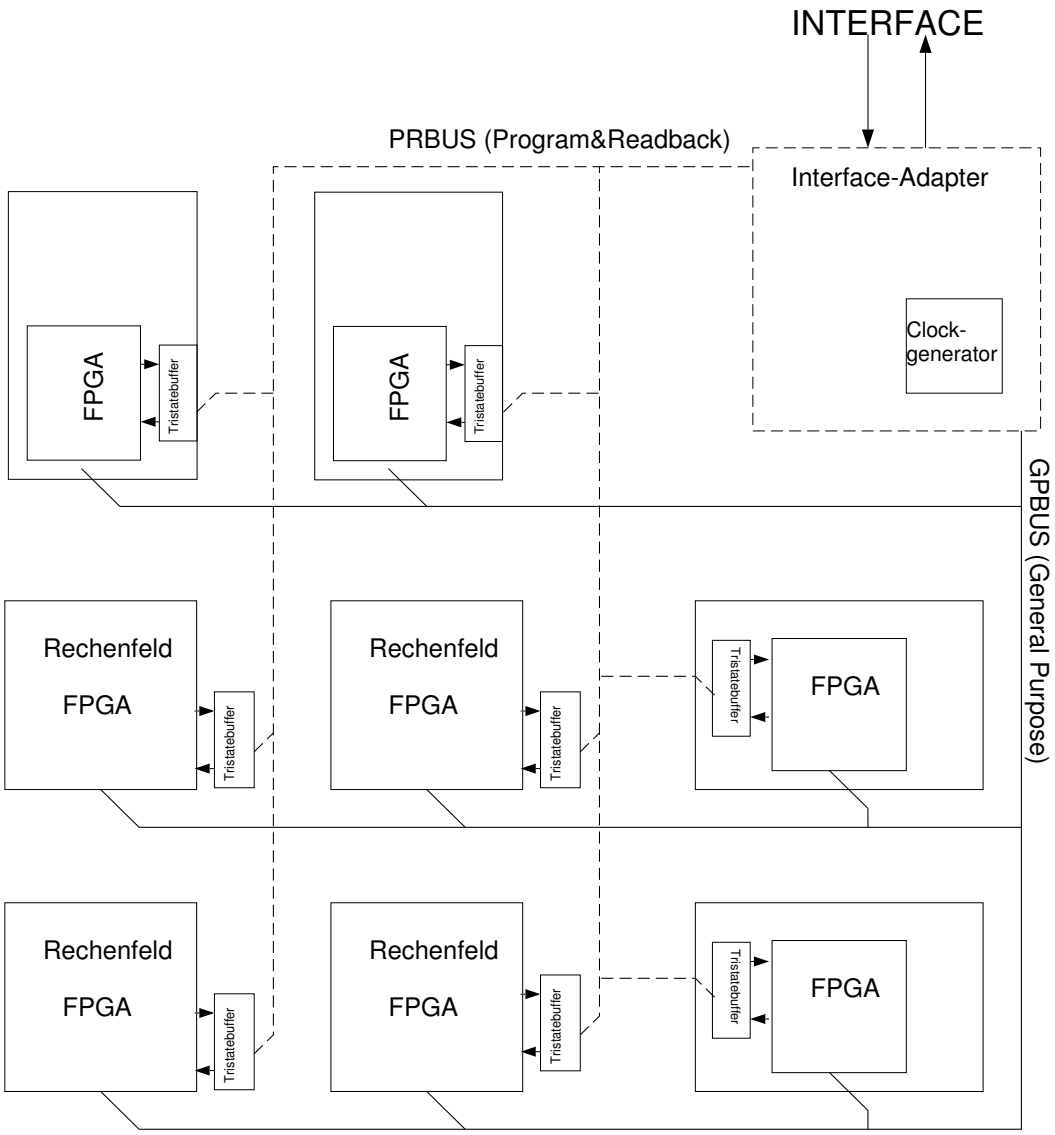

Bild 7 Prinzipschaltbild von PAr^2 — Programmierung, Readback und globaler Bus.

Aus der beschriebenen Notwendigkeit, die gesamte Hardware von PAr^2 anhalten zu können, ergibt sich in Verbindung mit dem READBACK-Mechanismus eine komfortables Fehlersuchverfahren. So ist es möglich, den Algorithmus im Einzel-Schritt-Verfahren abzuarbeiten und nach jedem Taktzyklus die internen Zustände der FPGAs auszulesen und zu überprüfen.

4 Die Komponenten von PAr^2

Die Hauptaufgabe bei der Konzeption und Realisierung von PAr^2 bestand darin, die Kommunikationsmittel für den *implementierten Algorithmus* bereitzustellen. Nach der Festlegung des Grundkonzepts im vorangegangenen Kapitel soll hier im einzelnen beschrieben werden, wie die Kommunikation der Elemente Rechenfeld, Ramcontroller mit RAM und Interface von PAr^2 geregelt ist. Im Anschluß daran werden die für diese Kommunikation benötigten Taktsignale abgeleitet und deren optimaler Verlauf bestimmt. Kapitel 4.4 befaßt sich mit dem Registermodell von PAr^2. Dieses Modell stellt eine Abstraktion von der physikalischen Realisierung der Kommunikation dar und ermöglicht eine Beschreibung durch fest definierte Schnittstellen an den FPGA-Grenzen. Der letzte Abschnitt dieses Kapitels befaßt sich mit der Programmierung der FPGAs.

Grundsätzlich geht das Konzept davon aus, daß zu jedem Taktzyklus der globalen Clock (ComCLK , **Com**munikation **CL**oc**K**) einmal Daten von jeder Datenquelle anfallen können, also von den FPGAs des Rechenfeldes, des Ramcontroller, vom RAM und vom Interface. Daß in der Regel beim *implementierten Algorithmus* nicht in jeden Taktzyklus ein Datum anfällt, stört dabei nicht, denn die transportierten Daten können von dem *implementierten Algorithmus* einfach nicht beachtet werden.

Den “Flaschenhals” in der Kommunikation zwischen Host und PAr^2 bildet hierbei sicherlich das Interface. Bei der gewählten Busbreite von 24 Bit und einer Kommunikation in beiden Richtungen vom und zum Interface mit einer Taktfrequenz von 8 MHz fallen am Interface 24*8*2/8 = 48 MB pro Sekunde an. Dies ist in seiner jetzigen Implementierung vom Interface bei einer Datenrate von 8MB/s nicht zu bewältigen, so daß eine Möglichkeit bestehen muß, das gesamte System von PAr^2 vom Interface aus anzuhalten, wenn das Interface die Daten nicht weiterleiten kann. Dieser Fall kann dabei nicht nur durch den hohen Datendurchsatz auftreten, sondern ebenso bei Kommunikationsengpässen mit dem Host.

4.1 Rechenfeld

4.1.1 Aufgabe

Das Rechenfeld besteht aus einem 3x3 Feld von XILINX XC4005 FPGAs und stellt somit eine Gatterkapazität von ca 45000 Gattern zur Verfügung. Die FPGAs sind durch lokale 24 Bit breite Busse miteinander verbunden. Hier soll der *implementierte Algorithmus* ablaufen.

4.1.2 Kommunikation der Rechenfeld FPGAs untereinander

Die Verbindungen der Rechenfeld-FPGAs untereinander sind unidirektional und statisch für den gesamten Ablauf eines Algorithmus. Das ist auch sinnvoll, denn eine Umkehr des Datentransfers auf einer Leitung während der Laufzeit des Algorithmus ist ausgeschlossen. Allerdings ist die Richtung des Datenflusses auf Grund der freien Programmierbarkeit der IOBs der XILINX FPGAs für jede Leitung unabhängig von den Richtungen der anderen Leitungen möglich.

Die Kommunikation zwischen zwei benachbarten Rechenfeld FPGAs gestaltet sich relativ einfach (Bild 8). An der Datenquelle der Leitung wird das IOB des entsprechenden FPGAs auf Ausgang mit vorgeschaltetem Register programmiert. An der Datensenke des anderen FPGAs befindet sich ein auf Eingang programmiertes Register im IOB. Beide Register, bei der Quelle wie bei der Senke, werden mit steigender Taktflanke von ComCLK beschrieben. Die Daten haben also eine Taktperiode von ComCLK Zeit, um von einem FPGA zum anderen zu gelangen.

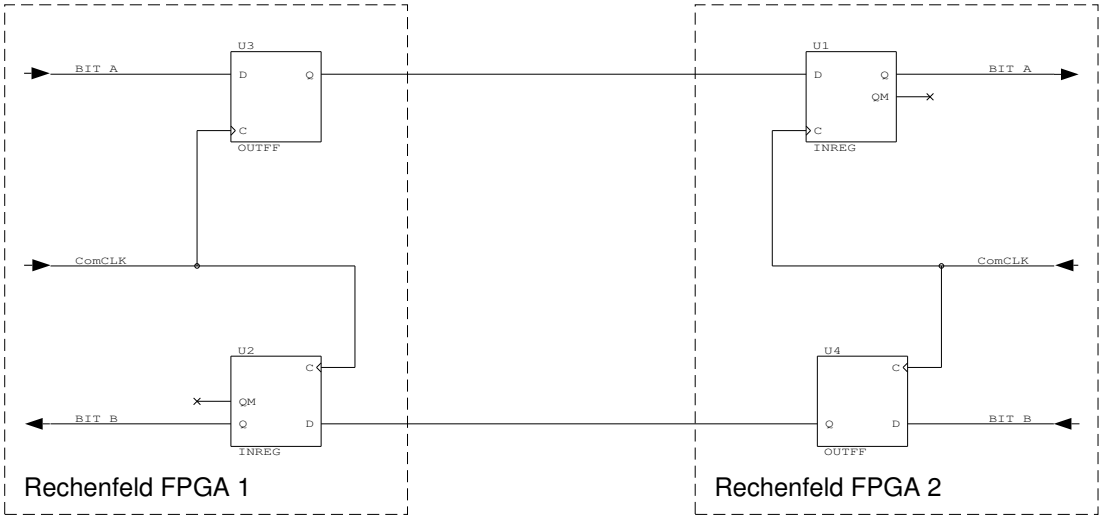

Bild 8 Kommunikation zwischen zwei Rechenfeld-FPGAs.

4.1.3 Kommunikation zwischen Rechenfeld und Ramcontroller

In Kapitel 3.4 wurde das Multiplexen aller Leitungen zum Ramcontroller eingeführt, um das Leitungsproblem am Ramcontroller zu lösen. Die dafür notwendige Multiplexschaltung zeigt Bild 9. Der gesamte Mehraufwand an Hardware gegenüber unidirektionalen Verbindungen kann in den IOBs der FPGAs untergebracht werden. Sie verfügen bereits über Tristate-Ausgangstreiber sowie Ein- und Ausgaberegister innerhalb eines IOBlocks (vgl Bild 2). Es gehen somit keine CLBs und damit Gatteräquivalente innerhalb der FPGAs verloren.

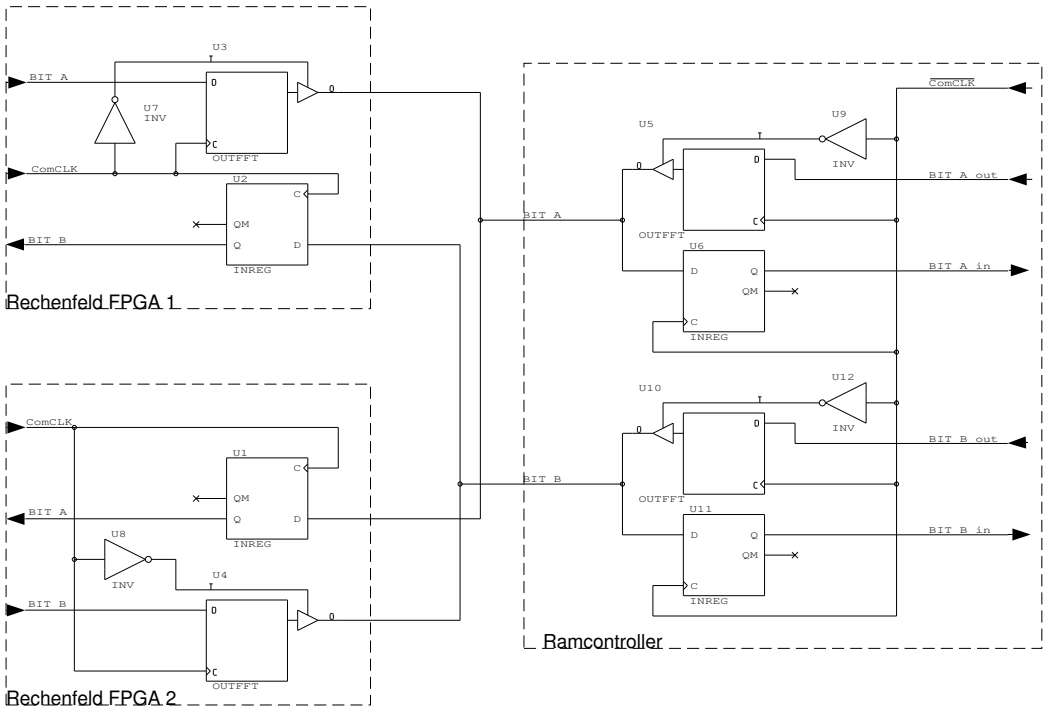

Bild 9 Kommunikation zwischen Rechenfeld und Ramcontroller

Für dieses Multiplexen schließt man alle einander entsprechenden Datenleitungen der beiden FPGAs und des Ramcontrollers kurz, also Bit A von Rechenfeld-FPGA 1 mit Bit A von Rechenfeld-FPGA 2 und Bit A des Ramcontrollers usw. In Bild 9 ist ein Beispiel für zwei Bit mit unterschiedlichen Richtungen gezeigt.

In der ersten Hälfte des Taktes (ComCLK=H) liest der Ramcontroller auf allen 24 Leitungen des Busses Daten ein. Dazu werden die Tristate-Ausgangsteiber in den Rechenfeld-FPGAs aktiviert (active-low) und so die Daten auf die Leitung gelegt. Dabei spielt es für den Ramcontroller keine Rolle, welches der beiden FPGAs nun auf welcher Leitung Daten ausgibt. Da die Leitungen vom *implementierten Algorithmus* aus unidirektional sind, kann es nicht zu einer Datenkollision kommen, denn es kann nur entweder FPGA1 oder FPGA2 an dieser Leitung auf Ausgang programmiert sein. Das andere FPGA muß auf diesem Bit einen Dateneingang programmiert haben. Sind die Daten stabil, werden sie mit fallender Taktflanke von ComCLK in den Ramcontroller eingelesen (vgl Bild 10).

In der zweiten Takthälfte (ComCLK=L) werden die Tristate- Ausgangstreiber der Rechenfeld-FPGAs deaktiviert und der Ramcontroller legt seine Daten auf den Bus. Daraufhin übernehmen die Rechenfeld-FPGAs mit steigender Taktflanke von ComCLK die Daten. Auch hier gilt, daß nur entweder FPGA1 oder FPGA2 des Rechenfeldes an einer Leitung einen Eingang programmiert haben kann.

Von "innen" betrachtet sieht dieses Kommunikationsschema für die FPGAs des Rechenfeldes aus, als hätten sie jeder einen eigenen 24 Bit breiten Bus

zum Ramcontroller. Auf diesem werden pro Taktperiode von ComCLK genau einmal (mit steigender Flanke von ComCLK) die Daten aus den Eingangsregister eingelesen bzw. in die Ausgangsregister geschrieben. Die Rechenfeld-FPGAs arbeiten also mit der Taktfrequenz von ComCLK.

Der Ramcontroller "sieht" einen 24 Bit breiten bidirektionalen Bus, bei dem die Leserichtung zu einem, die Schreibrichtung zu dem anderen FPGA des Rechenfeldes führt. Welche Richtung zu welchem Rechenfeld-FPGA führt, entscheidet sich erst bei der Programmierung von PAr^2 durch Festlegen der Datenrichtungen in den Rechenfeld-FPGAs. Der Ramcontroller arbeitet ebenfalls mit der Taktfrequenz von ComCLK. Da der Ramcontroller die Daten bei fallender Taktflanke übernimmt, das Rechenfeld jedoch bei steigender, herrscht zwischen den Ramcontrollern und Rechenfeld eine Phasenverschiebung von 180 Grad bzw. einem halben Takt. Der Ramcontroller wird deshalb mit dem invertierten Signal von ComCLK, also $\overline{\text{ComCLK}}$ betrieben. Dann arbeiten alle internen Register des Ramcontrollers ebenfalls mit steigender Flanke, allerdings mit steigender Flanke von $\overline{\text{ComCLK}}$.

Aus der obigen Darstellung ergibt sich, daß die FPGAs am Rand des Rechenfeldes nicht direkt miteinander kommunizieren können, sondern ihre Daten immer durch den Ramcontroller schicken müssen, wie das in den Grundüberlegungen in Kapitel 3.4 (Bild 5) dargelegt wurde. Selbst wenn dieser die Daten einfach nur weitergibt, ergibt sich eine größere Registeranzahl als zwischen zwei FPGAs innerhalb des Rechenfeldes. Darauf wird in Kapitel 4.4 eingegangen.

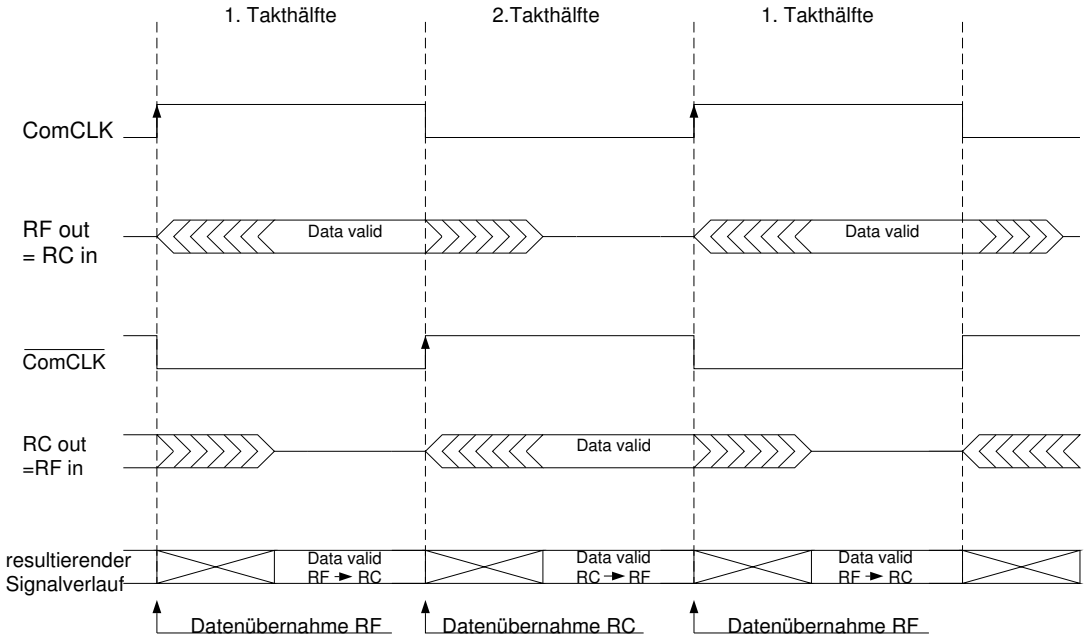

Bild 10 Prinzipielles Timingdiagramm für die Kommunikation zwischen Rechenfeld (RF) und Ramcontroller (RC).

Der IOB eines Pins eines auf Ausgang programmierten Rechenfeld-FPGAs sieht für die Busse innerhalb des Rechenfeldes (vgl. Bild 8) anders aus, als für die Ränder, die zu den Ramcontrollern führen (Bild 9). Diese Unregelmäßigkeit läßt sich aber durch das Einführen fester Signalbezeichnungen im FPGA (Kapitel 4.5) vor dem *implementierten Algorithmus* verbergen.

4.2 Ramcontroller

4.2.1 Aufgabe

Die Ramcontroller sind die Steuerzentralen der Rechenfeldes. Sie sind die Rückleitungen der Rechenfeld-FPGAs eingeschleift und können so die Daten im Rechenfeld verfolgen und beeinflussen. Je nach gewünschter Aktivität werden dann Daten vom Host gelesen oder an den Host gesendet, oder Daten ins RAM geschrieben oder vom RAM gelesen. All diese Aktionen werden durch die Steuerdaten, die auf den Bussen des Rechenfeldes fließen und vom Interface kommen, bestimmt.

Die Auswertung der Steuerinformation des Rechenfeldes ist dabei offen gelassen. In dieser Arbeit werden nur die entspechenden Schnittstellen zum Rechenfeld, Host und RAM zur Verfügung gestellt. Die Verbindung dieser Komponenten hängt vom *implementierten Algorithmus* ab und muß mit dessen Programmierung auch in den Ramcontroller-FPGAs definiert werden. Dies ist durchaus beabsichtigt; denn eine Spezialisierung von der Hardwareseite auf ein bestimmtes Steuerungsschema schränkt die Klasse der bearbeitbaren Algorithmen ein.

Die Hauptaufgabe des Speichers am Rand wird in den meisten Fällen das Zwischenspeichern der Daten für eine bestimmte Anzahl von Taktzyklen sein, der Speicher wird also als FIFO (First In – First out) genutzt werden. Allerdings ist es durchaus denkbar, daß mehrere FIFOs unterschiedlicher Tiefe auf verschiedenen Bits des Datenbusses realisiert werden müssen. Wollte man diese FIFOs durch eine allgemeine Schaltung unabhängig vom *implementierten Algorithmus* realisieren, hätte man sich auf eine bestimmte maximale Anzahl von FIFOs unterschiedlicher Tiefe festlegen und ihnen feste Bitnummern zuordnen müssen. Dies würde die Klasse der abarbeitbaren Algorithmen aber sehr einschränken.

Mit der jetzt zur Verfügung stehenden freien Gatterkapaziät von ca 4100 Gattern pro Ramcontroller-FPGA kann man eine individuelle Darstellung des RAMs für jeden *implementierten Algorithmus* gut realisieren. Da diese Realisierung relativ komplex ist, ist dabei an eine Bibliothek von Standardlösungen gedacht, aus der für den speziellen Fall eine Ansteuerung “maßgeschneidert” werden kann.

4.2.2 Kommunikation zwischen Ramcontroller und Interface

Die Ramcontroller sind über den Interface-Bus (IFBUS) mit dem Interface verbunden. Dieser besteht aus zwei 24 Bit breiten Teilbussen für Lesen und Schreiben.

Mit derselben Überlegung wie bei der Kommunikation zwischen Rechenfeld und Ramcontroller (4.1.3) kann man die Leitungsanzahl am Ramcontroller in Richtung Interface halbieren. Die zwei getrennten Busse für Datenein- und Datenausgabe werden an jedem Ramcontroller über Bustreiber zusammengefaßt (Bild 11).

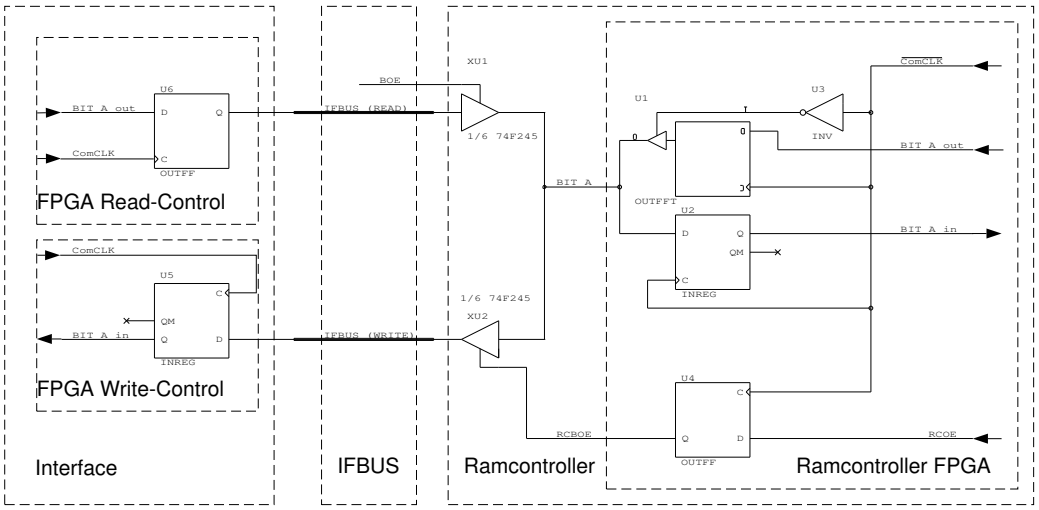

Bild 11 Kommunikation zwischen Ramcontroller und Interface

Betrachten wir zunächst den Datentransfer vom Ramcontroller zum Interface über den Write-Bus. Hat der Ramcontroller Daten zu senden, liegen sie an “Bit A out” an. Gleichzeitig muß die Steuereinheit des Ramcontrollers an der Leitung RCOE ein Low anlegen (für die Signalverläufe siehe auch Bild 12). Mit der nächsten steigenden Flanke von $\overline{\text{ComCLK}}$ werden die Daten in die IOB-Register übernommen. Dadurch wird das Signal $\overline{\text{RCBOE}}$ low und der Bustreiber XU2 aktiv. Gleichzeitig hiermit wird der Bustreiber eines anderen Ramcontrollers, der eventuell vorher den Bus belegt hat, inaktiv, da bei ihm RCOE high sein muß. Der WRITE-Bus ist nun ausschließlich von diesem FPGA belegt. Solange $\overline{\text{ComCLK}}$ high ist, ist der Ausgangstreiber im OUTFFT durchgeschaltet, und die Daten gelangen über die Tristatetreiber XU2 und den IFBUS(WRITE) zum Interface. Wechselt $\overline{\text{ComCLK}}$ von High auf Low, ist das gleichbedeutend mit einem Wechsel von ComCLK von Low auf High, was eine Übernahme der angelegten Daten in die IOBs des “FPGA Write-Control” des Interface zur Folge hat.

Nun erfolgt der Datentransfer in die andere Richtung (Ramcontroller-READ). Mit dem Wechsel von ComCLK auf High wurden die im “FPGA Read-Control”

an “Bit A out” anliegenden internen Daten in OUTFF übernommen und gelangen über IFBUS(READ) an XU1. Gleichzeitig wird der Tristatetreiber im Ramcontroller FPGA inaktiv. Analog zur Schreibrichtung würde man den Tristatetreiber XU1 mit $\overline{\text{ComCLK}}$ steuern. Da das Aktivieren dieses Treibers aber wesentlich schneller geschieht als das Abschalten des Treibers in den FPGAs, wurde aus Sicherheitsgründen ein zusätzliches Signal $\overline{\text{BOE}}$ eingeführt, dessen fallende Flanke gegenüber der von $\overline{\text{ComCLK}}$ verzögert ist. Solange dieses Signal Low ist, liegen die Daten an U2 (INREG) des Ramcontrollers an. Hier werden sie mit steigender Flanke von $\overline{\text{ComCLK}}$ übernommen. Damit wird auch BOE wieder High und der Treiber XU1 inaktiv und ein Schreib-/Lesezyklus ist abgeschlossen.

Während der Ramcontroller-READ-Phase bleibt der Treiber XU2 aktiv, wodurch die Lesedaten auch über den IFBUS(WRITE) an den INREGs des Interface anliegen. Dies stört aber nicht, da die Daten nur mit steigender Taktflanke von ComCLK ins Interface übernommen werden. Die Daten liegen aber nur solange an, bis ComCLK wieder Low wird.

4.2.3 Kommunikation zwischen Ramcontroller und RAM

Als RAM wurde statischer Speicher gewählt, da er ohne komplizierte Refresh-Schaltungen auskommt und die benötigten kurzen Zugriffszeiten von 12ns ermöglicht.

Der Speicher wird ebenfalls innerhalb eines Taktes von ComCLK sowohl gelesen als auch geschrieben. Dazu verfügen die Datenleitungen zum RAM über denselben Multiplex-Mechanismus wie bei den Rechenfeld- bzw. Interface-Anschluss. Die Adreßleitungen müssen aber ebenfalls gemultiplext werden, damit das Schreiben an einer anderen Adresse wie das Lesen innerhalb des selben Taktzyklus möglich ist. Diese Multiplexschaltung ist in Bild 13 gezeigt. Sie stellt dem Ramcontroller intern getrennte Adreßleitungen für Lesen (RADR0–14) und Schreiben (WADR0–14) zur Verfügung, die synchron mit steigender Taktflanke von $\overline{\text{ComCLK}}$ übernommen werden. Mit dieser Taktflanke werden auch die Schreibdaten der angelegten Schreibadresse in die Datenbus-IOBs übernommen. Ebenfalls gleichzeitig erscheinen die Lesedaten der Ramadresse des letzten Taktzyklus am internen Eingang der FPGAs.

Im nächsten Takt werden zuerst bei $\overline{\text{ComCLK}}$=H die Schreibadressen auf den Adreßleitungen ausgegeben. Gleichzeitig sind die Datenleitungen des FPGAs mit $\overline{\text{ComCLK}}$=H auf Ausgang geschaltet. Das $\overline{\text{WE}}$-Signal der RAMs wird nach Einhalten der Adreßhaltezeit global auf Low gesetzt, wodurch die Daten in das RAM geschrieben werden. Geht das $\overline{\text{WE}}$-Signal wieder auf High, ist der Schreibzugriff beendet. Dies erfolgt gleichzeitig mit dem Wechsel von $\overline{\text{ComCLK}}$ auf Low. Damit werden zum einen an den Adreßleitungen die Leseadresse ausgegeben, zum anderen die DatenbusIOBs des Ramcontrollers auf Eingang

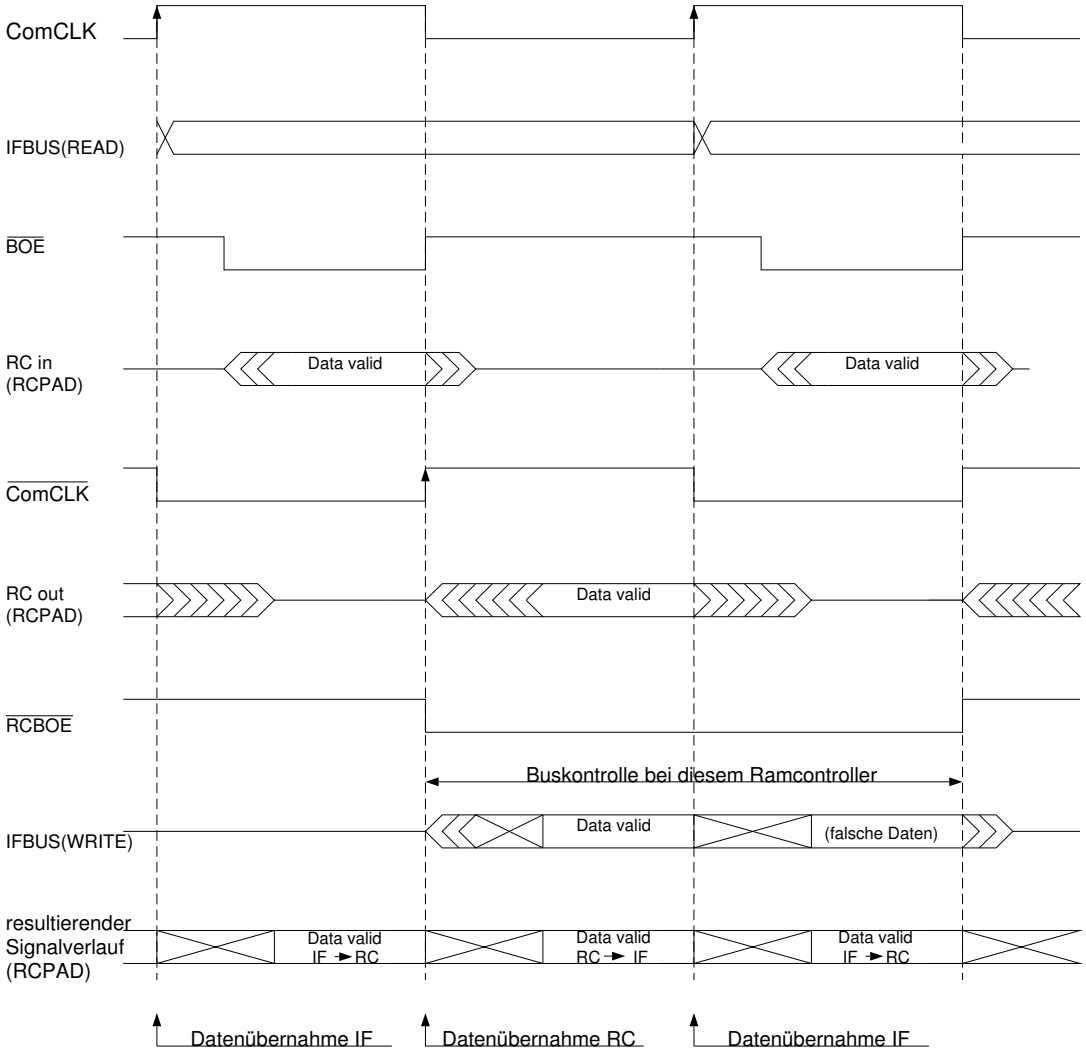

Bild 12 Prinzipielles Timingdiagramm für die Kommunikation zwischen Ramcontroller(RC) und Interface(IF) über den IFBUS.

geschaltet. Nach Abwarten der Adreßhaltezeit wird mittels des globalen $\overline{\text{OE}}$-Signals die Daten vom RAM auf den Datenbus gelegt. Mit steigender Taktflanke von $\overline{\text{ComCLK}}$ werden diese Daten ins FPGA eingelesen und der Ablauf beginnt von vorne.

Da die Schaltung völlig synchron mit globalen Taktsignalen arbeitet, würden Daten in jedem Taktzyklus sowohl vom RAM gelesen als auch ins RAM geschrieben. Es ist aber durchaus denkbar, daß nicht in jedem Taktzyklus Daten für das RAM anfallen. Die vom RAM gelesen Daten stören dabei wenig, sie können von dem *implementieren Algorithmus* einfach ignoriert werden. Das Schreiben von Daten ins RAM muß jedoch in einem solchen Fall verhindert werden, da nicht garantiert werden kann, daß über mehrere Taktzyklen am internen Datenbus

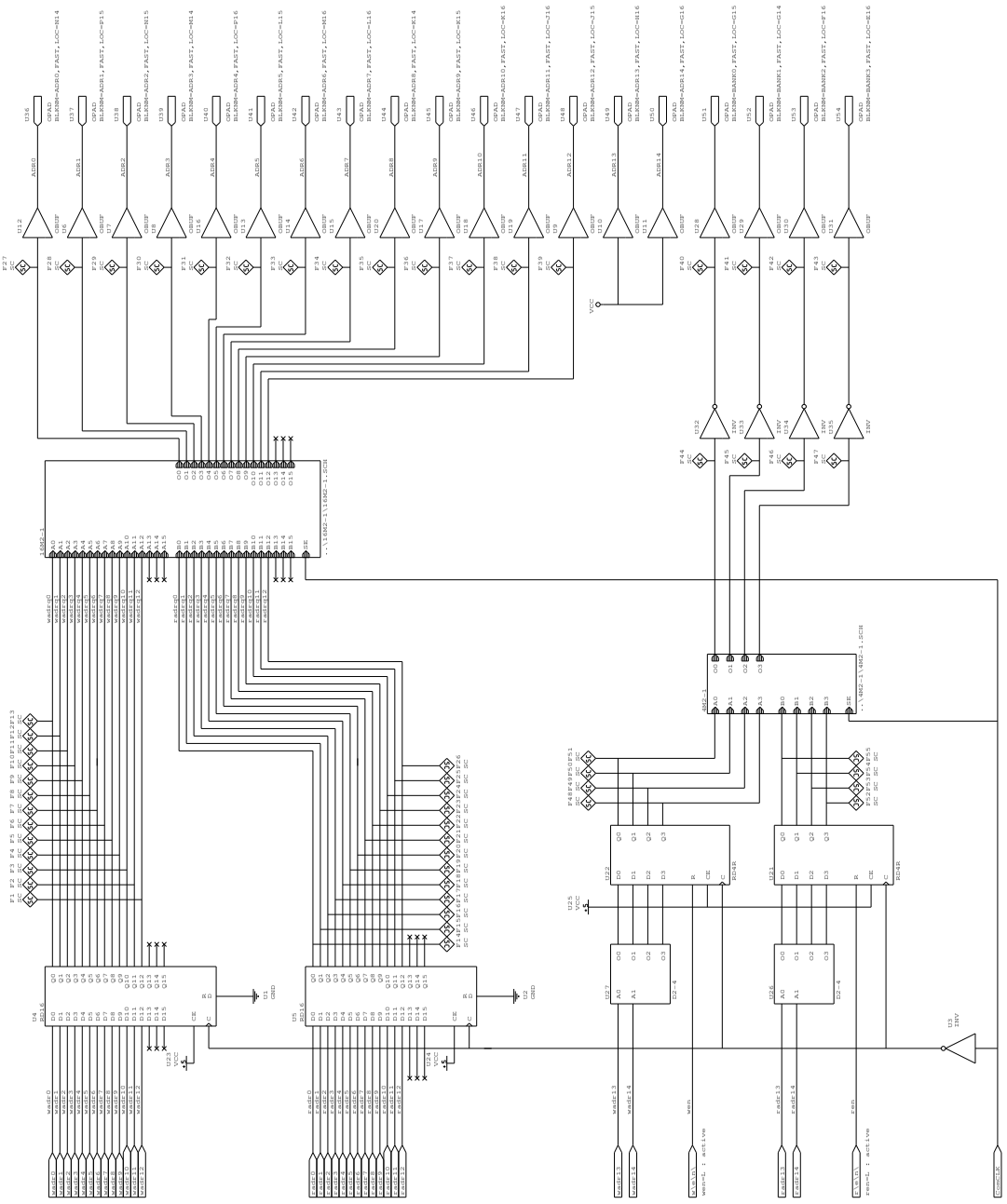

Bild 13 Adreßmultiplexer der Ramcontroller

des FPGAs gültige Schreibdaten anstehen. Dieser Schreibschutz erfolgt über das $\overline{\text{CS}}$ Signal des RAMs, das nur dann auf Low ist, wenn am $\overline{\text{WEN}}$ Eingang des Adreßmultiplexers ebenfalls ein Low anliegt. Liegt dort ein High an werden zwar die Schreibadressen ausgeben und Schreibdaten an den Datenbus gelegt, auch den $\overline{\text{WE}}$-Signal kommt aufgrund der globalen Generierung dieses Signals am RAM an, weil $\overline{\text{CS}}$ aber High ist, werden die Daten nicht übernommen. Aus Symmetriegründen wurde für die Leserichtung das analoge Signal $\overline{\text{REN}}$ eingeführt.

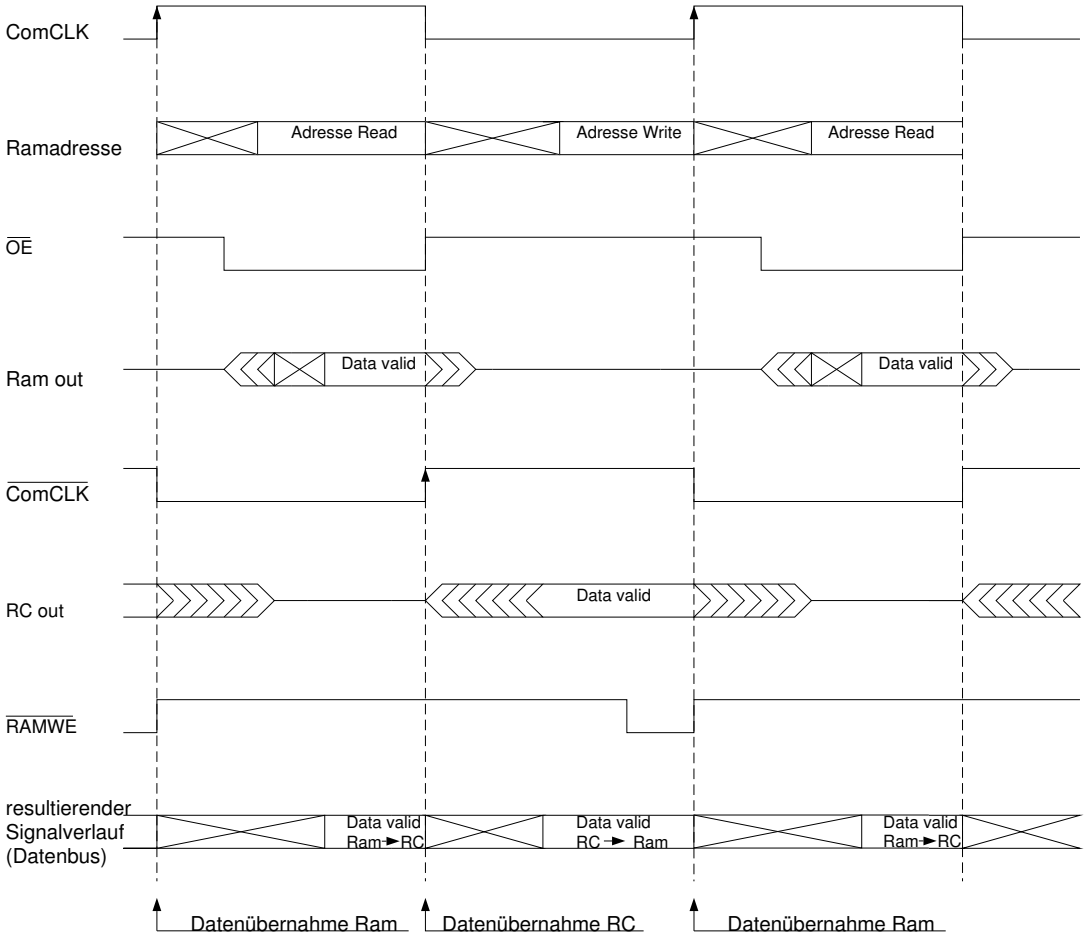

Bild 14 Prinzipielles Timingdiagramm für die Kommunikation zwischen Ramcontroller und RAM.

Mit Hilfe des $\overline{\text{CS}}$ Signals am RAM wird es auch möglich, mehrere RAM-Bausteine parallel zu schalten und nur über das $\overline{\text{CS}}$ Signal auszuwählen, welches RAM angesprochen wird. Auf diese Weise wurden 4 parallele BANKs vorgesehen, deren Auswahl über die obersten Adreßbits erfolgt. Die Schaltung wie sie jetzt vorliegt, ist für 8kB RAMs konzipiert, es ergibt also ein maximaler Adreßraum von 32k. Da aber das Pinout von 8kB und 32kB RAMs kompatibel ist, wurden die zusätzlichen Leitungen bereits vorgesehen, und so können maximal 4x32k=128k Adreßraum angesprochen werden. Dafür ist die Schaltung des Adreßmultiplexer aus Bild 13 an den vergrößerten Adreßraum anzupassen.

Die Schaltung des Adreßmultiplexer ist sehr zeitkritisch und erfordert deshalb eine optimale Plazierung und Verdrahtung auf dem FPGA, damit keine zu großen Laufzeiten auftreten. Deshalb ist vorgesehen, den Adreßmultiplexer aus einer Datei zu der restlichen Schaltung des FPGAs dazuzuladen. In dieser Datei sind dann bereits Plazierungsinformationen enthalten, welche die kurzen Laufzeiten

garantieren. Dies wird durch eine Beschreibung des Adreßmultiplexers durch ein sogenanntes Hardmacro erreicht (siehe [18]).

Der Adreßmultiplexer ist - wie der Umfang der Schaltung zu erkennen gibt - im Gegensatz zu den Multiplexern der Datenleitungen nicht in den IOBs des FPGAs unterzubringen. Er belegt 33 CLBs, also 17% des Ramcontroller-FPGAs.

Damit ist die Beschreibung der Kommunikation abgeschlossen. Im nächsten Abschnitt soll nun das dafür notwendige Timing hergeleitet werden.

4.3 Takterzeugung

4.3.1 Grundkonzept

Da die Schaltung synchron arbeitet, muß ein globaler Takt zur Versorgung aller FPGAs erzeugt werden. Neben diesem eigentlichen Taktsignal ComCLK sind noch zwei zusätzliche globale Signale zur Steuerung der statischen RAMs notwendig: das Output-Enable-Signal $\overline{\text{OE}}$ und das Write-Enable Signal $\overline{\text{WE}}$. Das Signal $\overline{\text{BOE}}$ zum Steuern der Tristatetreiber der Ramcontroller hat einen identischen Verlauf wie das $\overline{\text{OE}}$-Signal für die RAMs. Dieses Signal wird fortan mit $\overline{\text{BOE}}$ bezeichnet, das RAM $\overline{\text{WE}}$ Signal mit $\overline{\text{RAMWE}}$.

Alle Signale werden von einem schnellen Grundtakt BaseClock (BSCLK) mit 64 Mhz abgeleitet. Dieser Grundtakt speist einen 3 Bit Zähler, an den eine 3 Bit tiefes ROM angeschlossen ist. An den Datenausgängen des ROMs lassen sich so je nach Programmierung beliebige Signale aus maximal 8 Segmenten und einer Auflösung von 12.5 ns erzeugen. Bild 15 zeigt das Schaltung der Takterzeugung wie sie in einem FPGA des Interface untergebracht ist.

Da dieser Zähler und das ROM in einem der FPGAs des Interface untergebracht ist, ist es möglich, die durch das ROM erzeugten Taktsignale beliebig zu ändern. Deshalb wurden neben den erwähnten Taktsignalen noch vier weitere Leitungen des Interface reserviert, um bei Bedarf zusätzliche Taktsignale erzeugen zu können.

Wie erwähnt, ist die Möglichkeit vorgesehen, das Rechenfeld durch das Interface anzuhalten. Dazu dient das Signal HALT, das mit steigender Flanke von ComCLK in das Ausgangsflipflop von GTBUF geschrieben wird. An diesem Signal ist der Output-Enable Pin der Takttreiber angeschlossen. Geht dieses Signal auf high, werden die Ausgänge der Treiber inaktiv. Durch Pullup- bzw. Pulldown-Widerstände wird der Zustand der Signale hinter den Treibern “eingefroren”. Somit bleibt das Rechenfeld stehen. Wird das HALT-Signal wieder auf Low gelegt erfolgt das Aktivieren der Treiber synchron zu ComCLK, und das Rechenfeld läuft weiter.

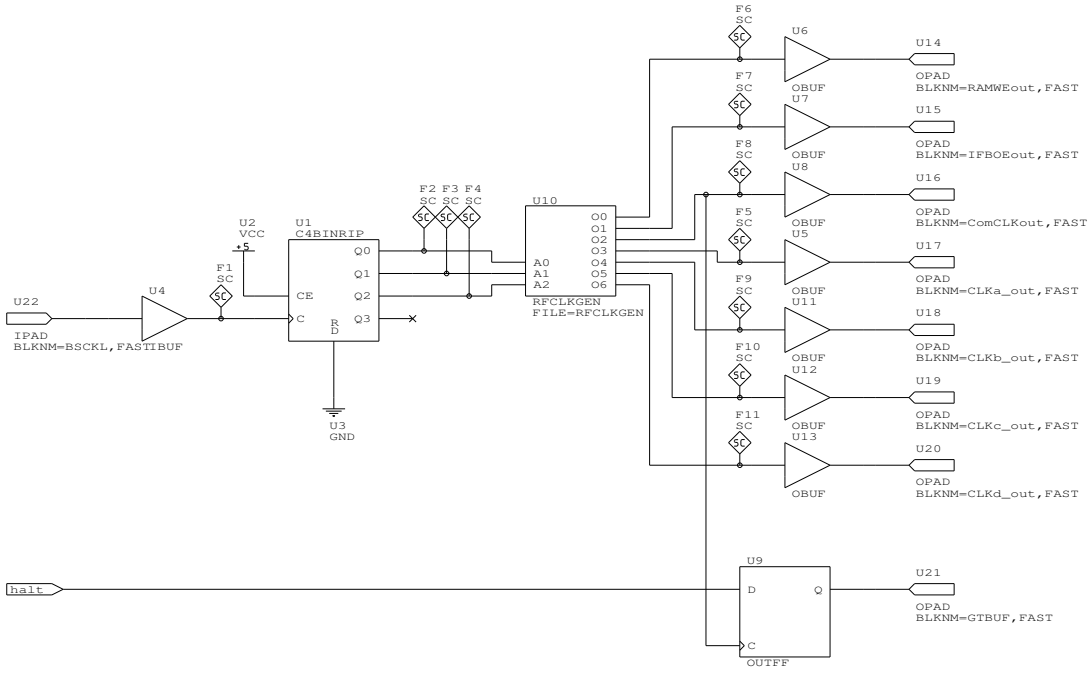

Bild 15 Prinzip der Taktsignalerzeugung aus einem Grundtakt

Das Ableiten der Taktsignale aus einem gemeinsamen Grundtakt wurde der Erzeugung von Taktsignalen mittels Verzögerungsglieder aus mehreren Gründen vorgezogen. So ist es zum einen möglich, die Signale durch einfaches Umprogrammieren des Interface FPGAs dem oben beschriebenen Rahmen zu ändern. Außerdem kann so bei Laufzeitproblemen das gesamte Timing von PAr^2 durch Auswechseln des Quarzoszillators für den Grundtakt geändert werden. Dies erwies sich insbesondere in der Testphase des Rechenfeldes als sehr nützlich. Indem man zuerst mit einem Quarz niedriger Frequenz die Schaltung testet, kann man laufzeitbedingte Fehler ausschließen. Wenn eventuelle logische Fehler beseitigt wurden, kann man durch Erhöhen der Taktfrequenz die maximale Datenrate festlegen. Sollte das Rechenfeld später vergrößert werden, wodurch sich längere Laufzeiten der Signale auf dem globalen Interface-Bus ergeben, kann die Taktfrequenz ebenfalls einfach angepaßt werden.

4.3.2 Bestimmung des optimalen Timing

Zur Bestimmung der optimalen Taktsignale wurden die einzuhaltenden Zeiten zum Umschalten der diversen Treiber und des RAMs als lineares Programm formuliert und dadurch das optimale Timing bestimmt. Dabei stellt sich das Problem, daß die einzelnen Gatterlaufzeiten zwar sehr genau bekannt waren([15], [11] und [5]), die Laufzeit der Signale auf den Leitungen jedoch nur schwer abzuschätzen war. Dies lag einerseits an der anfangs nicht bekannten Geometrie

des Rechenfeldes, zum anderen an den unbekannten dielektrischen Eigenschaften des Platinenmatrials. So wurden die Laufzeiten als Variablen in das lineare Programm eingesetzt, deren Werte erst durch empirische Messungen am fertigen Aufbau bestimmt werden konnten. Die Länge des gemultiplexten Busses zwischen Rechenfeld und Ramcontroller von etwa 40 cm ergab eine Laufzeit von ca 7ns. Zwischen Interface und Ramcontroller sind die Signalleitungen mit ca 60 cm noch länger und die Verzögerung somit noch größer (11 ns). Diese Verzögerungen sollen nun systematisch untersucht werden und daraus ein lineares Programm formuliert werden, um die maximale Taktfrequenz zu bestimmen.

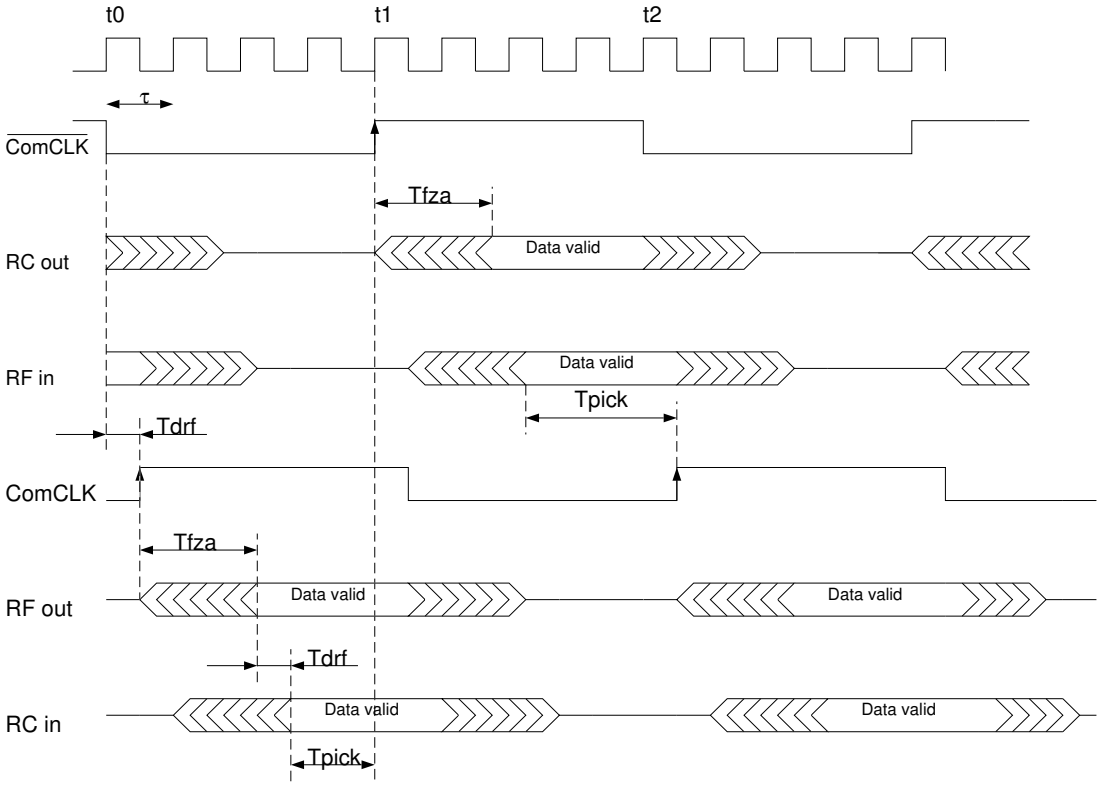

Bild 16 Vollständiges Timingdiagramm für die Kommunikation zwischen Rechenfeld und Ramcontroller

Betrachten wir zunächst die Kommunikation zwischen Rechenfeld und Ramcontroller (Bild 16). Hierbei wird von einer Taktverschiebung T_{drf} und einer Datenlaufzeit derselben Größe ausgegangen. Dies ist durchaus akzeptabel, da die Leitungslängen der entsprechenden Signale in derselben Größenordnung liegen. Für die Kommunikation vom Ramcontroller zum Rechenfeld erhalten die Daten dieselbe Verzögerung wie das Taktsignal. Es müssen also nur die Zeiten für das Einschalten der Ausgangstreiber des FPGAs (T_{fza}) und die Haltezeit am Eingangsregister (T_{pick}) eingehalten werden. Dies liefert uns als erste Bedingung im

linearen Programm:

$$T_{fza} + T_{pick} < t_2 - t_1$$

Bei der Kommunikation vom Rechenfeld zum Ramcontroller (2. Takthälfte von ComCLK in Bild 16) ist der Takt $\overline{\text{ComCLK}}$ des Rechenfeldes bereits um T_{drf} gegen den Takt des Ramcontrollers verschoben. Nachdem das Rechenfeld seine Daten durchgeschaltet hat (T_{fza}), dauert es wiederum T_{drf} Sekunden, bis die Daten am Ramcontroller-FPGA anliegen. Dort müssen sie mindesten T_{pick} Sekunden anstehen, bevor ComCLK auf high wechselt. Dies liefert uns eine weitere Bedingung für das lineare Programm:

$$T_{drf} + T_{pick} + T_{drf} + T_{fza} < t_1 - t_0$$

Betrachten wir nun die Signale zwischen Interface und Ramcontroller (Bild 17). Da alle Signale vom Interface erzeugt werden, erhalten die Daten bei der Kommunikation vom Interface zu den Ramcontroller dieselbe Verzögerung, wie die Taktsignale selbst. Somit muß die Verzögerung der Signale in dieser Richtung – analog zu dem oben betrachteten Fall – nicht berücksichtigt werden (erste Takthälfte in Bild 17). Die relevanten Zeiten sind die Aktivierungszeit des externen Treibers (T_{bza}) und die Datenhaltezeit (T_{pick}). Die Bedingung an das lineare Programm ergibt sich so zu:

$$T_{bza} + T_{pick} < t_1 - t_3$$

wenn t_3 der Zeitpunkt am Interface ist, zu dem das Signal $\overline{\text{BOE}}$ von High auf Low wechselt. Es ist aber möglich, daß das $\overline{\text{BOE}}$–Signal schneller schaltet, als die Durchlaufzeit der Daten durch das Flipflop (T_{okop}) und den externen Treiber XU1 (T_{bd}) ist. Dies wird mit folgender Ungleichung berücksichtigt:

$$T_{okop} + T_{bd} + T_{pick} < t_1 - t_0$$

Dazu kommt die Bedingung, daß der externe Treiber nicht zu früh schalten darf, damit nicht zwei Ausgänge gegeneinander arbeiten:

$$T_{faz} - T_{short} < t_3 - t_0 + Tbza$$

wobei T_{short} die Zeit ist, die zwei Ausgänge maximal gegeneinander geschaltet sein dürfen.

Bei der Kommunikation in der anderen Richtung vom Ramcontroller zum Interface laufen die Daten den Taktsignalen sozusagen entgegen, was eine doppelte Verzögerung der Daten mit sich bringt. Die Signale vom Interface besitzten eine Verzögerung von T_{dif}. Das bedeutet, daß bei der Kommunikation vom Ramcontroller zum Interface der Ramcontroller erst mit einer Verzögerung von T_{dif}

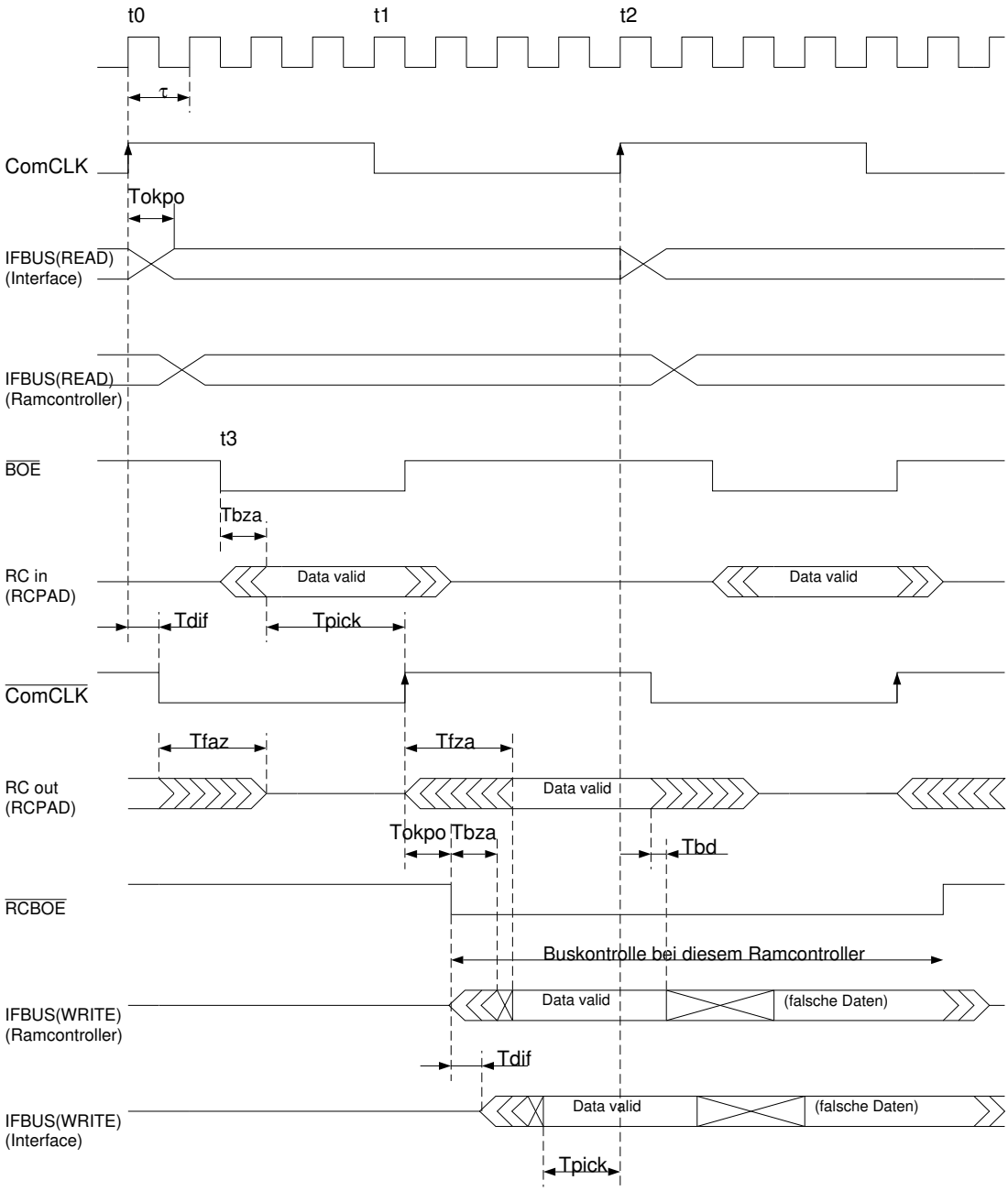

Bild 17 Vollständiges Timingdiagramm für die Kommunikation zwischen Ramcontroller und Interface

gegenüber dem Interface die Daten absendet. Ebenfalls mit dieser Verzögerung wird das Register des Steuersignals $\overline{\text{RCBOE}}$ getriggert. Bis dieses Signal tatsächlich am Pad erscheint, vergehen T_{okpo} Sekunden. Weitere T_{bza} Sekunden dauert es, bis der Treiber XU2 von den hochohmigen Zustand in den aktiven Zustand gewechselt ist. Nun ist das Signal bis zum Interface wieder T_{dif} lang unterwegs und muß dort wieder T_{pick} Sekunden anstehen, bevor ComCLK auf high

wechselt. Somit ergibt sich als Bedingung:

$$T_{dif} + T_{pick} + T_{dif} + T_{okop} + T_{bza} < t_2 - t_1$$

Auch hier muß eine Bedingung bezüglich der Durchlaufzeit berücksichtigt werden:

$$T_{dif} + T_{fza} + T_{dif} + T_{pick} < t_2 - t_1$$

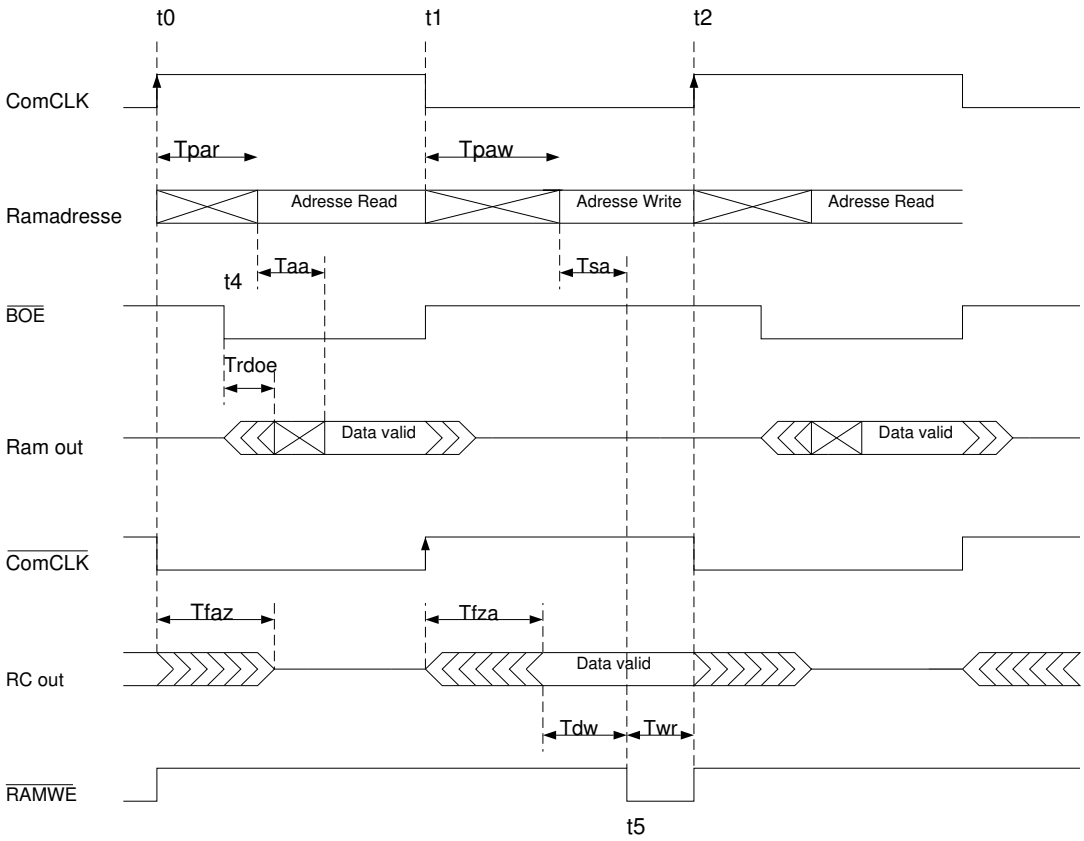

Bild 18 Vollständiges Timingdiagramm für die Kommunikation zwischen Ramcontroller und Ram.

Als dritter Teil soll nun die Kommunikation zwischen RAM und Ramcontroller untersucht werden. Da hier die Leitungslängen relativ kurz sind, wurden keine Signallaufzeiten berücksichtigt. Das zeitkritischste Element ist der Adreßmultiplexer, da hier auch interne Laufzeiten auf dem FPGA berücksichtigt werden müssen. Die Verzögerungszeiten zwischen den Flanken von ComCLK und dem Erscheinen der Daten am PAD wurden mittels des XILINX-Tool XDELAY bestimmt. Dabei ist T_{par} die Zeit, die es dauert, bis nach positiver Flanke von ComCLK die Leseadressen gültig sind. T_{paw} ist die entsprechende Zeit für die Schreibdaten nach fallender Flanke von ComCLK.

Daraus lassen sich im wesentlichen schon die Bedingungen formulieren. Für den Lesezugriff auf das RAM muß die Adresse mindestens T_{aa} Sekunden stabil

sein, bevor die Daten gültig sind. Diese Daten müssen wiederum T_{pick} Sekunden am Ramcontroller anstehen, bevor $\overline{\text{ComCLK}}$ auf High wechselt:

$$T_{par} + T_{aa} + T_{pick} < t_1 - t_0$$

Das Umschalten des Datenbusses nimmt T_{rdoe} Sekunden nach der negativen Flanke von $\overline{\text{BOE}}$ (t_4) in Anspruch:

$$T_{rdoe} + T_{pick} < t_1 - t_4$$

Auch soll eine maximale Kurzschlußzeit zweier Ausgänge von T_{short} garantiert werden:

$$T_{faz} - T_{short} < t_4 - t_0 + T_{rdoe}$$

Für den Schreibzugriff erhält man die analogen Bedingungen:

$$\begin{aligned} T_{paw} + T_{sa} &< t_5 - t_1 \\ T_{fza} + T_{dw} &< t_5 - t_1 \\ T_{wr} &< t_2 - t_5 \end{aligned}$$

wobei t_5 der Zeitpukt ist, wenn das Signale $\overline{\text{WE}}$ von High auf Low wechselt.

Für konkrete Zahlenwerte sind natürlich die meisten der angegebenen Ungleichungen redundant, aber das lineare Programm sollte so allgemein wie möglich formuliert werden.

Da alle Signale von dem gemeinsamen Takt BSCLK (τ) abgeleitet werden, müssen die Schaltzeitpunkte an ganzzahligen Vielfachen von τ liegen:

$$\begin{aligned} t_1 - t_0 &= a * \tau \\ t_2 - t_0 &= b * \tau \\ t_3 - t_0 &= c * \tau \\ t_4 - t_0 &= d * \tau \\ t_5 - t_0 &= e * \tau \\ a, b, c, d, e &\in \mathbb{N} \end{aligned}$$

Insgesamt ergibt sich durch Zusammenfügen aller Bedingungen folgendes Optimierungsproblem:

$$
\begin{aligned}
& min \quad \tau, \\
(b-a) * \tau &\geq T_{fza} - T_{pick} \\
a * \tau &\geq 2 * T_{drf} \\
(a-c) * \tau &\geq T_{bza} + T_{pick} \\
a * \tau &\geq T_{okop} + T_{bd} + T_{pick} \\
c * \tau &\geq T_{faz} - T_{short} - T_{bza} \\
(b-a) * \tau &\geq 2 * T_{dif} + T_{pick} + T_{okop} + T_{bza} \\
(b-a) * \tau &\geq 2 * T_{dif} + T_{fza} + T_{pick} \\
a * \tau &\geq T_{par} + T_{aa} + T_{pick} \\
(a-d) * \tau &\geq T_{rdoe} + T_{pick} \\
d * \tau &\geq T_{faz} - T_{short} - T_{rdoe} \\
(e-a) * \tau &\geq T_{paw} + T_{sa} \\
(e-a) * \tau &\geq T_{fza} + T_{dw} \\
(b-e) * \tau &\geq T_{wr} \\
& a, b, c, d, e \in \mathbb{N}
\end{aligned}
$$

Ein Problem hierbei ist, daß nicht nur τ, sondern auch die ganzzahligen Variablen a, b, c, d, e zu bestimmen sind. Da sie als Faktoren vor der unabhängigen Variablen τ stehen, handelt es sich eigentlich nicht mehr um ein linerares Programm, sondern um ein quadratisches Optimierungsproblem mit teilweise ganzzahligen Variablen. Ein solches Optimierungsproblem ist im allgemeinen schwer zu lösen. Deshalb werden alle sinnvollen Werte für a bis e durch Enumerieren eingesetzt. “Sinnvoll” heißt hierbei Werte zwischen eins und einer Obergrenze (im Programm die Variable `resolution`). Für b wird dabei immer die Obergrenze `resolution` eingesetzt, da b die Periode aller Taktsignale bestimmt ($b * \tau = t_2 - t_0$). Für jede Kombination der Variablen a, c, d, e muß dann ein einfaches lineares Programm gelöst werden. Das minimale τ aller gefundenen Lösungen liefert die maximale Taktfrequenz der BSCLK und die Werte für die Faktoren a bis e.

Im Programm wurde `resolution=8` gesetzt, da sich eine Zweierpotenz leicht als Binärzähler realisieren läßt. Höhere Werte können zwar eventuell ein besseres Ergebnis liefern, sind aber praktisch nicht mehr realisierbar. Denn ein Faktor von beispielsweise zehn würde bedeuten, daß die BSCLK zehn mal schneller läuft als die resultierende ComCLK. Solch hohe Frequenzen für das BSCLK-Signal sind aber praktisch nicht mehr handhabbar.

Das Enumerieren der möglichen Werte mag unelegant erscheinen. Da aber das lineare Programm nur eine Variable zu bestimmen hat, handelt es sich faktisch um eine Maximumbildung. Diese braucht sehr wenig Rechenzeit und so fällt die Enumeration selbst bei 4096 Durchläufen (`resolution=8`) kaum ins Gewicht.

Das entsprechende Programm zur Bestimmung der Zeiten ist im Anhang A abgedruckt. Für die dort eingesetzten Werte ergab sich eine maximale Frequenz von ComCLK von 8.62 MHz. In der Realisierung wurde 8 MHz gewählt, entsprechend einem BSCLK-Quarz von 64 MHz. Der resultierende Verlauf der Signale zeigt Bild 19.

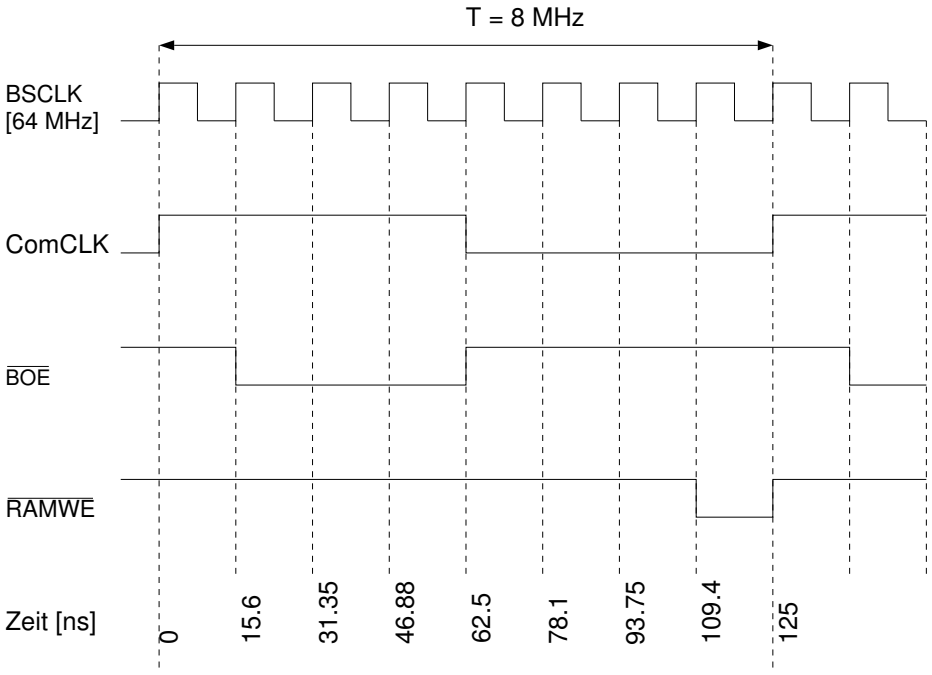

Bild 19 Resultiernder Verlauf der globalen Taktsignale von PAr^2

4.4 Modellierung der Kommunikation

Aus den in den vorangegangenen Abschnitten vorgestellten Kommunikationsschaltungen läßt sich ein Modell für die Kommunikationsvorgänge auf Registerebene ableiten.

Die Taktverschiebung zwischen Interface und Rechenfeld auf der einen Seite und Ramcontroller auf der anderen soll dabei näher betrachtet werden. Interface und Rechenfeld arbeiten wie erwähnt auf steigende Taktflanke von ComCLK, bilden also eine mit positiver Flanke getriggerte Logik, während der Ramcontroller auf steigende Taktflanke von $\overline{\text{ComCLK}}$ seine Daten taktet, also negativ getriggerte Logik beinhaltet. An der Grenze folgt somit einem Register mit positiver Flanke (Rechenfeld) ein Register mit negativer Flanke (Ramcontroller), wenn man den

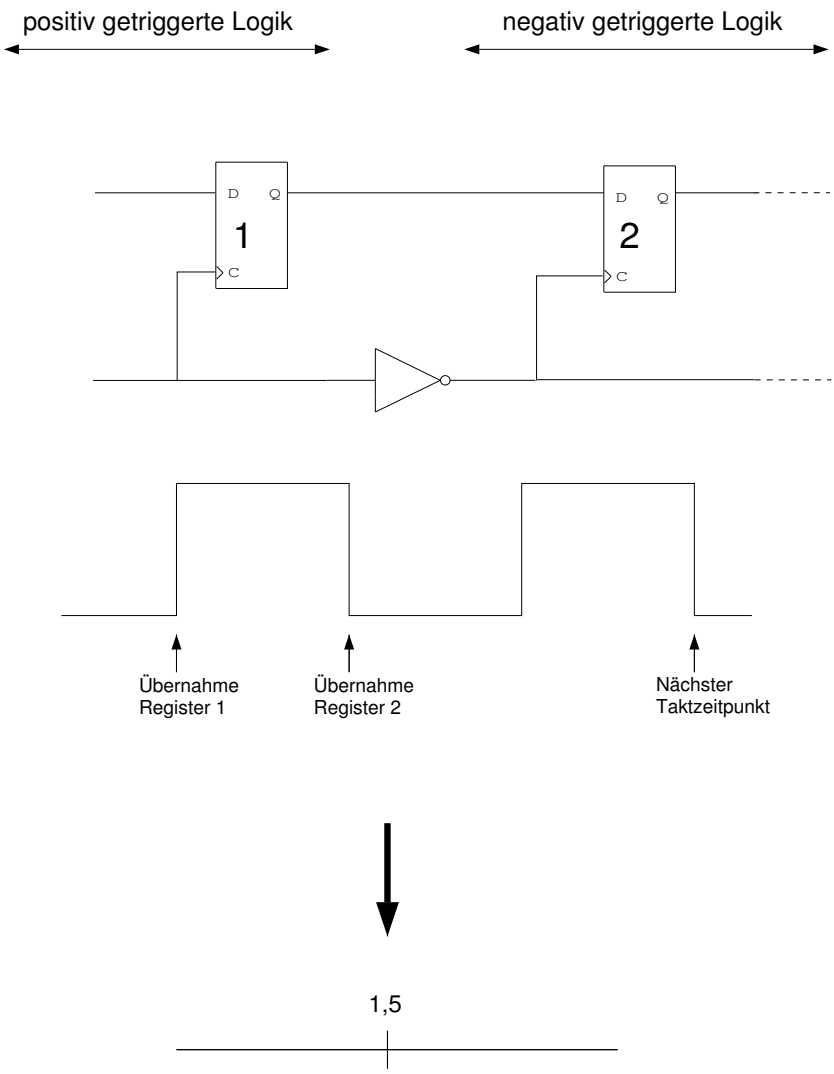

Bild 20 An der Grenze zwischen positiver getriggerter und negativ getriggerter Logik wirken zwei Register wie eineinhalb.

Datenfluß vom Rechenfeld zum Ramcontroller betrachtet. In der umgekehrten Datenflußrichtung folgt entsprechend ein Register mit negativer Flanke auf ein Register mit postiver Flanke. Zusammengefaßt ergeben beide Reihenfolgen jeweils 1,5 Register (siehe Bild 20). Da “halbe Register” schwer vorstellbar sind und sie auch in COMPAR nicht modelliert werden können, muß man mit Hilfe einer Schnittmengentransformation die Registerzahl ganzzahlig machen.

Die Schnittmengentransformation ist ein systematisches Verfahren, um Signalflußgraphen zu transformieren. Das Registermodell von PAr^2 kann als Graph interpretiert werden, wenn man jedem FPGA einen Knoten zuordnet und die Verbindungsleitungen als Kanten auffaßt. Die Schnittmengentransformation gibt nun eine Regel an, mit der man die Registerzahlen in den Kanten des Graphen ver-

ändern kann, ohne die Funktionsweise der Schaltung zu ändern [14]. Dazu wählt man sich eine Menge G von Knoten. Wenn man nun in alle Leitungen (Kanten), die in diese Menge G hineinführen, c Register hinzufügt und bei den Leitungen (Kanten), die hinausführen, c Register wegnimmt, hat sich an der Schaltungsfunktion nichts geändert.

Diese Registerzahlen c müssen nicht notwendigerweise ganzzahlig sein. Wählt man als Menge G alle Ramcontroller mit Adreßmultiplexer und RAM und setzt $c = \frac{1}{2}$, ergeben sich im Hinpfad zwei Register und im Rückpfad eines (siehe Bild 21). Natürlich könnte man auch $c = -\frac{1}{2}$ setzten und erhielte so im Hinpfad eines und im Rückpfad zwei Register. Dies macht für die Funktionsweise keinen Unterschied.

Die in Bild 21 eingezeichneten Hin- und Rückleitungen müssen natürlich nicht bei jedem *implementierten Algorithmus* vorkommen. Es ist auch ein Datenfluß in nur eine Richtung bei einigen FPGAs denkbar. Das Bild soll jedoch alle Spezialfälle abdecken.

Das Rechenfeld ist also nicht völlig homogen, die Ränder müssen immer über den Ramcontroller kommunizieren, auch wenn sie eigentlich die Daten nur an ein anderes FPGA des Rechenfeldes weitergeben wollen. Denn selbst bei einer direkten Durchverbindung der Leitungen im Ramcontroller sind in diesen Leitungen drei Register im Gegensatz zu zwei Registern innerhalb des Rechenfeldes.

Das Beschreiben des RAMs erfolgt mit einer Verzögerung von einem Takt, entsprechend befinden sich in Adreß- und Datenbus in Schreibrichtung jeweils ein Register. Die Daten, die man beim RAM-Zugriff lesen kann, sind immer die Daten der Adresse, die zum Zeitpunkt vorher angelegen hat. Entsprechend liegt in der Adreßleitung ein Register, in der rückführenden Datenleitung jedoch keines.

Die Partitionierung in COMAR muß den Algorithmus so aufteilen, daß er nicht mehr Ressourcen benötigt, als das Rechenfeld insgesamt zur Verfügung stellen kann. Außerdem darf keinem einzelnen FPGA mehr Hardware zugeordnet werden, als ein XILINX-Baustein fassen kann. Und drittens muß die Aufteilung des Algorithmus so erfolgt sein, daß die Register, die durch die Kommunikation entsprechend Bild 21 vorhanden sind, im Algorithmus im entsprechenden Datenpfad auch vorgegeben sind.

Mit Hilfe des vorgestellten Registermodells wird es möglich, die Kommunikation innerhalb von PAr^2 zu modellieren und dem *implementierten Algorithmus* fest definierte Schnittstellen zur Verfügung zu stellen. Dies kann abstrahiert von der tatsächlichen physikalischen Realisierung der Kommunikation geschehen, der implementierte Algorithmus muß davon nichts wissen.

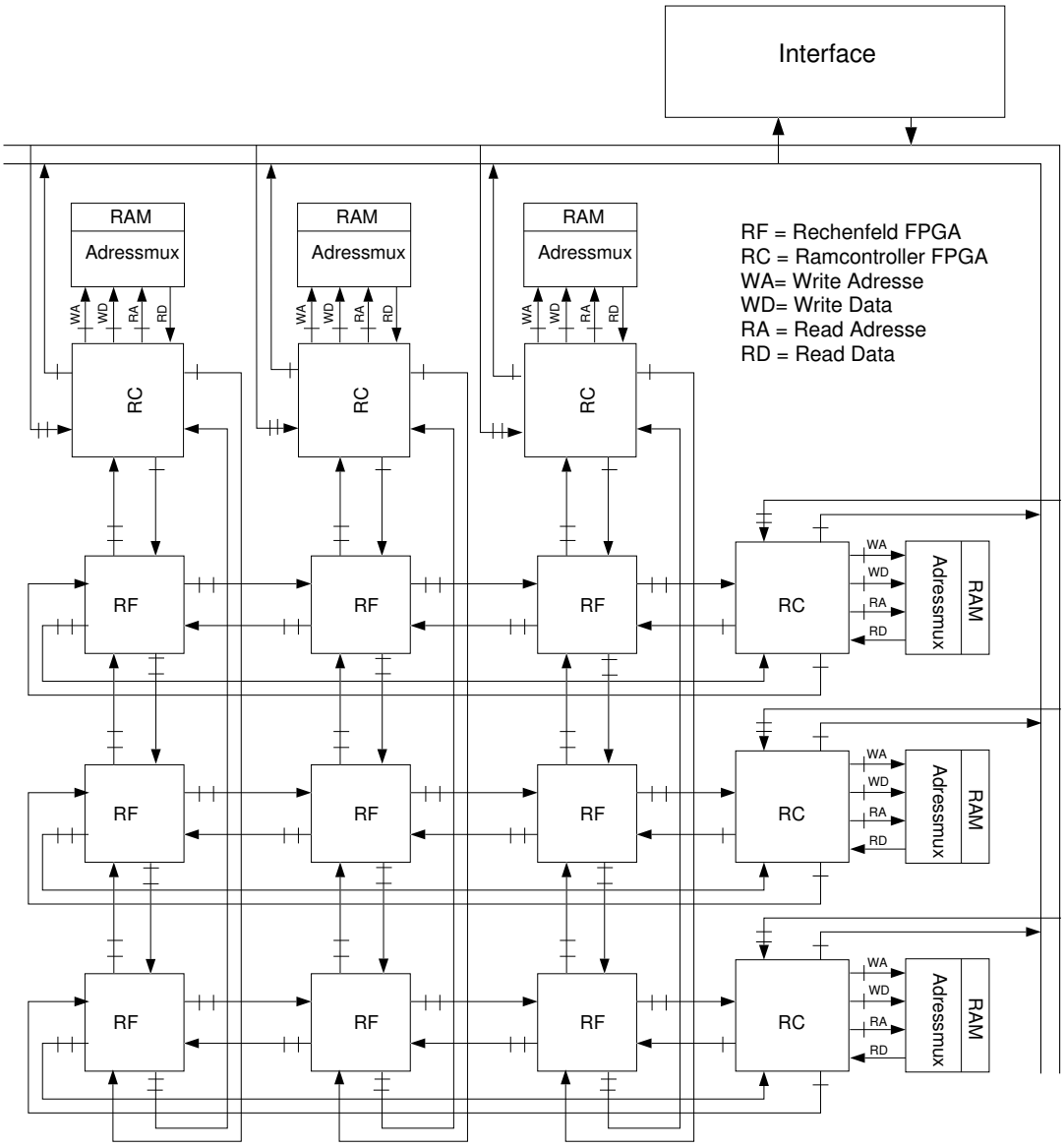

Bild 21 Registermodell von PAr^2

4.5 Programmierung und Debugging

4.5.1 Vorbereiten der Programmierung

Der zu implementierende Algorithmus muß, wie im vorangegangenen Kapitel dargelegt, erst von COMPAR so partitioniert werden, daß einzelne Segmente entstehen, die nicht mehr Schaltunglogik beinhalten, als ein FPGA aufnehmen kann. Ebenso muß die Schaltung für die Ramcontroller spezifiziert werden. So erhält man 15 Beschreibungen für die verschiedenen FPGAs. Die Beschreibung der Chipinhalte erfolgt mit einem lesbaren Netzlistenformat, das von XILINX

entwickelt wurde, dem sogenannten XNF-Format. Aus diesem Format wird mit Hilfe des PPR-Programms (Partition, Place & Route) ein Zwischenformat erzeugt, das mittels des Programms MAKEBITS in den Bitstrom zur Programmierung eines FPGAs umgewandelt wird. Das Programm PPR hat dabei relativ komplexe Aufgaben zu lösen, denn es muß die Netzliste, die es als Eingabe erhält, so partitionieren, daß die CLBs optimal genutzt werden und die Verdrahtung zwischen den CLBs durchführen. Dieser Vorgang kann für ein gut ausgelastetes XC4005 FPGA durchaus mehr als eine Stunde Rechenzeit in Anspruch nehmen.

Die Beschreibung der Kommunikationsvorgänge hat gezeigt, daß unterschiedliche Inhalte in die Input/Output-Blöcke der FPGAs des Rechenfeldes programmiert werden müssen. Die Ränder des Rechenfeldes benötigen in ihren Ausgangspins zusätzlich Tristatetreiber, die innerhalb des Rechenfeldes nicht benötigt werden. Um diese Unregelmäßigkeit vor dem *implementierten Algorithmus* zu verbergen, werden einheitliche Signalbezeichungen innerhalb des Chips im XNF-Netzlistenformat eingeführt. Für die neun unterschiedlichen Zuordnungen der Ränder zu den FPGAs des Rechenfeldes müssen an die Netzliste nur noch die IOB-Belegungen angefügt werden, die für die neun Chips in neun Dateien spezifiziert sind. Die Festlegung der Namen ist in Tabelle 1 für das Rechenfeld aufgeführt. Es existieren pro Pin zwei Datenrichtungen (**READ** und **WRITE**), es kann an den implementieren Algorithmus jedoch immer nur eines der beiden Signale – abhängig von der Datenrichtung – angeschlossen werden. Die zweite, nicht benötigte Richtung wird von dem Programm PPR unter Ausgabe einer Warnung entfernt, weil entweder der Ausgang eines Flipflops (bei **READ**) oder der Eingang eines Flipflops (bei **WRITE**) nicht angeschlossen ist.

Signalbezeichnungen des Rechenfeld-FPGA				
	Datenbus links	Datenbus rechts	Datenbus oben (up)	Datenbus unten (down)
Pinbelegung	D3 - N2	C15 - N14	A1 - A14	R3 - R13
Signal in FPGA hinein (READ)	DLR0 - DLR23	DRR0 - DRR23	DUR0 - DUR23	DDR0 - DDR23
Signal aus FPGA hinaus (WRITE)	DLW0 - DLW23	DRW0 - DRW23	DUW0 - DUW23	DDW0 - DDW23

Tabelle 1 Signalbezeichnungen im Rechenfeld-FPGA

Ebenso kann man bei den Ramcontroller-FPGAs vorgehen. Diese sind zwar alle identisch in der Pinbelegung, aber die Programmierung der IOBs sowie der Adreßmultiplexer brauchen nicht für jedes FPGA noch zusätzlich von COMPAR erzeugt zu werden. Diese immer gleichen, der Hardware entsprechenden Beschrei-

bungen sind auch als Datei abgelegt, die an die eigentliche Spezifizierung des Ramcontrollers angehängt wird, bevor PPR aufgerufen wird. Für den Adreßmultiplexer enthält diese Datei, wie in Kapitel 4.2.3 erwähnt, Plazierungsinformationen, um kurze Signallaufzeiten zu garantieren. Die Festlegung der Signalbezeichnung für den Ramcontroller zeigen die Tabellen 2 und 3.

Signalbezeichnungen des Ramcontroller-FPGA			
	Datenbus Rechenfeld	Datenbus RAM	Datenbus Interface
Pinbelegung	D3 - N2	A1 - A14	R3 - R13
Signal in FPGA hinein (READ)	DRFR0 - DRFR23	DRAMR0 - DRAMR23	DIFR0 - DIFR23
Signal aus FPGA hinaus (WRITE)	DRFW0 - DRFW23	DRAMW0 - DRAMW23	DIFW0 - DIFW23

Tabelle 2 Signalbezeichnungen im Ramcontroller-FPGA

Signalbezeichnung Adressmultiplexer Ramcontroller-FPGA		
	Adressbezeichnung	Enable-Signal
Lesezugriff (READ)	RADR0 - RADR14	$\overline{\text{REN}}$
Schreibzugriff (WRITE)	WADR0 - WADR14	$\overline{\text{WEN}}$

Tabelle 3 Signalbezeichnungen im Ramcontroller-FPGA

4.5.2 Programmiervorgang

Die Programmierung der FPGAs erfolgt einzeln nacheinander im Serial-Slave-Mode [15].

Dazu darf immer nur ein FPGA mit den Programmierleitungen über den PRBUS an das Interface angeschlossen sein. Die Auswahl erfolgt über Tristatebustreiber, die über eine Adresse angesprochen werden. Die Adressenauswahl nimmt ein BCD-zu-dezimal Decoder global für alle FPGAs vor. Zu jeder Adresse gehört genau ein Treiber, dessen FPGA dann an den PRBUS geschaltet wird.

Wurde so ein FPGA ausgewählt, wird mit der fallenden Flanke von $\overline{\text{PROG}}$ die Programmierung eingeleitet. Das $\overline{\text{INIT}}$—Signal quittiert diesen Vorgang durch einen Wechsel auf High. Nach einer Pause von mindestens 40μs werden die Daten seriell über DIN mit der Frequenz CCLK eingelesen. Sollte während der

Programmierung ein Fehler auftreten, wird das $\overline{\text{INIT}}$—Signal vom FPGA auf Low gezogen.

Da sich bei einer Neuprogrammierung des Rechenfeldes die Richtungen der Leitungen ändern können, muß ein Mechanismus vorgesehen werden, der verhindert, daß während einer Programmierphase zwei Ausgänge aufeinander treffen. Dazu dient das Signal GTS (Global TriState), das an jedes FPGA geführt wird. Bevor die Programmierung gestartet wird, wird dieses Signal aktiviert, worauf alle FPGA-Ausgänge in den hochohmigen Zustand gehen. Jetzt können die FPGAs der Reihe nach programmiert werden, ohne daß die Gefahr besteht, daß zwei Ausgänge gegeneinander geschaltet werden. Außerdem muß garantiert werden, daß das gesamte Rechenfeld gleichzeitig losläuft. Während der Programmierung sorgt das $\overline{\text{DONE}}$–Signal dafür, das die Ausgänge von bereits programmierten FPGAs hochohmig bleiben, bis alle FPGAs programmiert wurden. Dazu ist das Signal nicht wie die anderen Leitungen über die Tristatetreiber geschaltet, sondern alle DONE-Leitungen aller FPGAs sind direkt parallel verbunden. Dies ist möglich, da es sich um Open-Kollektor-Ausgänge handelt. Der Ausgangspegel dient dem FPGA nach der Programmierung gleichzeitig als Information, ob die Ausgänge aktiviert werden dürfen. Dazu "hält" man das DONE-Signal von außen, also durch das Interface auf Low.

Ein Programmiervorgang soll deshalb folgendermaßen ablaufen:

- Anhalten des globalen Taktes ComCLK.
- Alle Pins aller FPGAs durch Umschalten von GTS auf High hochohmig machen.
- DONE-Signal auf Low ziehen.
- Selektieren und Programmieren der einzelnen FPGAs.
- DONE-Signal "loslassen"; geht es auf High, ist kein Fehler während der Programmierung aufgetreten.
- GTS=Low.
- ComCLK starten.

4.5.3 Readback

Der READBACK-Mechanismus der XILINX FPGAs ermöglicht ein Auslesen der internen Registerzustände. Die so erhaltenen Informationen können Debuggingzwecken dienen.

Um die Informationen aus einem bestimmten FPGA auszulesen, wird derselbe Adressierungsmechanismus verwendet wie bei der Programmierung. Die Tristatetreiber schalten also nicht nur die Leitungen für die Programmierung an das Interface, sondern auch die Leitungen RTRIG, RIP, RCLK und RDATA für das READBACK.

Für weitere Einzelheiten zum Programmier- bzw. Readbackvorgang sei auf [15] und [7] verwiesen.

5 Realisierung der Hardware

Am Anfang der Überlegungen stand unter anderem, das Konzept des Rechenfeldes möglichst modular zu halten, um es bei Bedarf vergrößern zu können. Deshalb wurde für die Realisierung zuerst versucht, eine Platine für jedes FPGA des Rechenfeldes vorzusehen, die über Stecker einen beliebigen Ausbau des Rechenfeldes ermöglichen.

Der Aufwand an Steckverbindungen wäre aber auf dem Rechenfeld enorm gewesen. Denn bei einem zweidimensionalen Rechenfeld sind die vier möglichen Richtungen in der Ebene bereits durch die Anschlüsse der Nachbarprozessoren belegt. Da aber auch globale Leitungen für den Takt und die Programmierung der FPGAs benötigt werden, hätte man diese von unten oder oben für jedes FPGA extra zuführen müssen und so Probleme bei der Verteilung dieser Signale auf dem Rechenfeld bekommen.

Deshalb wurde in der Realisierung ein Kompromiß eingegangen: Die Ramcontroller, die bereits eine relativ hohe Komplexität aufweisen, wurden jeweils auf einer Platine aufgebaut, auf die das RAM mit einer zusätzlichen Platine (Ramcard) eingesteckt wird. Das gesamte Rechenfeld mit den neun FPGAs wurde aber auf einer großen Platine untergebracht. Diese Platine ist nicht erweiterbar, d.h. sollte man ein größeres Rechenfeld wünschen, müßte dafür eine komplett neue Rechenfeld-Platine gebaut werden. Allerdings kann man durch die vorhandenen Kurzschlußverbindungen das Rechenfeld beliebig verkleinern. Die Ramcontroller können in einer vergrößerten Version des Rechenfeldes aber weiterhin genutzt werden.

Alle Platinen wurden zweiseitig durchkontaktiert gefertigt. Für die Anpassung an die Schnittstelle des Interfaces ist eine zusätzliche Platine (Interface-Adapter) nötig. Diese wurde in Fädel-Technik realisiert.

5.1 Überblick

Der Aufbau gliedert sich in drei große Teile: das eigentliche Rechenfeld mit neun FPGAs, die sechs Ramcontroller am Rand zur Steuerung von Speicherzugriffen und der Kommunikation mit dem Interface, sowie dem Interface-Adapter, der die physikalische Schnittstelle zum Interface bildet.

Die Datenbusbreite beträgt 24 Bit, dementsprechend sind die RAMs auch zu 3x8 Bit breiten Daten angeordnet. In der momentanen Ausstattung sind pro Ramcontroller 24 x 8k Ram, also insgesamt 144 kB Speicher auf dem gesamten Rechenfeld aufgebaut. Es ist aber ein Ausbau mit bis zu 24 x 128k RAM je Ramcontroller möglich, was einem Gesamtspeicher von 2304 kB entspricht.

Der Rechentakt beträgt 8 MHz. Wegen des getrennten Busses für READ und WRITE zwischen Interface und Ramcontroller können somit maximal 3 * 2 * 8 = 48MByte Daten pro Sekunden zum Transport für das Interface anfallen.

Bild 22 zeigt das Blockschaltbild von PAr^2. Das Rechenfeld mit den neun FPGAs befindet sich auf einer großen Rechenfeldplatine. Jeder Ramcontroller befindet sich mit RAM auf einer eigenen Platine, die über eine VG64-Leiste an das Rechenfeld angeschlossen wird. Der horizontale, 60 Leitungen umfassenden IFBUS verbindet alle Ramcontroller am oberen Rand des Rechenfeldes mit der Interface-Adapter Platine. Durch den entsprechenden vertikalen Bus sind die Ramcontroller am rechten Rand der Rechenfeldplatine mit dem Interface-Adapter verbunden. Eine 40polige Verbindung zwischen Interface-Adapter und Rechenfeld dient hauptsächlich der Programmierung der FPGAs, sowie der Verteilung der globalen Signale PRBUS, GPBUS und ComCLK. Auf dem Interface-Adapter ist des weiteren die Grundtakterzeugung, sowie die Takttreiber und Taktverteilung untergebracht. Zusätzlich sind in Bild 22 die Adressen der FPGAs für die Programmierung und das Readback eingezeichnet.

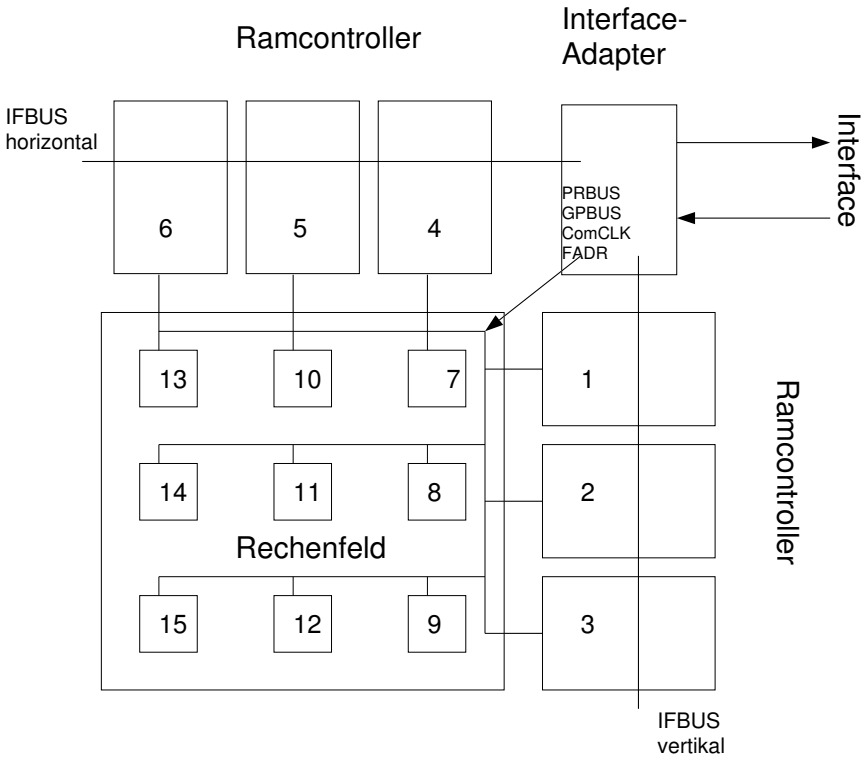

Bild 22 Blockschaltbild mit Verteilung der globalen Signale sowie der Adressen der FPGAs

5.2 Rechenfeld

Das Rechenfeld aus den 3 x 3 FPGAs befindet sich auf einer großen Platine von ca 40 x 42 cm. Sie ist sehr regelmäßig aufgebaut. Zu jedem FPGA existiert ein Tristate-Treiber, der die Programmierleitungen und Readbackleitung des entsprechenden FPGAs auf den Program&Readback-Bus schaltet. Die Ansteuerung dieser Treiber erfolgt über einen BCD-zu-dezimal-Dekoder 74154, der 16 Adressen auswählen kann. So werden durch diesen Decoder nicht nur die Rechenfeld-FPGAs adressiert, sondern sechs Leitungen werden an die Ramcontroller weitergeleitet. Die freibleibende Adresse 0 soll immer dann adressiert sein, wenn keines der FPGAs programmiert werden soll.

Die Verteilung der globalen Signale (ComCLK, PRBUS, GPBUS) erfolgt nach dem in Bild 22 angegebenen Schema. Die Ramcontroller werden also von dem Rechenfeld her mit den globalen Signalen versorgt (siehe auch Steckerbelegung im Anhang).

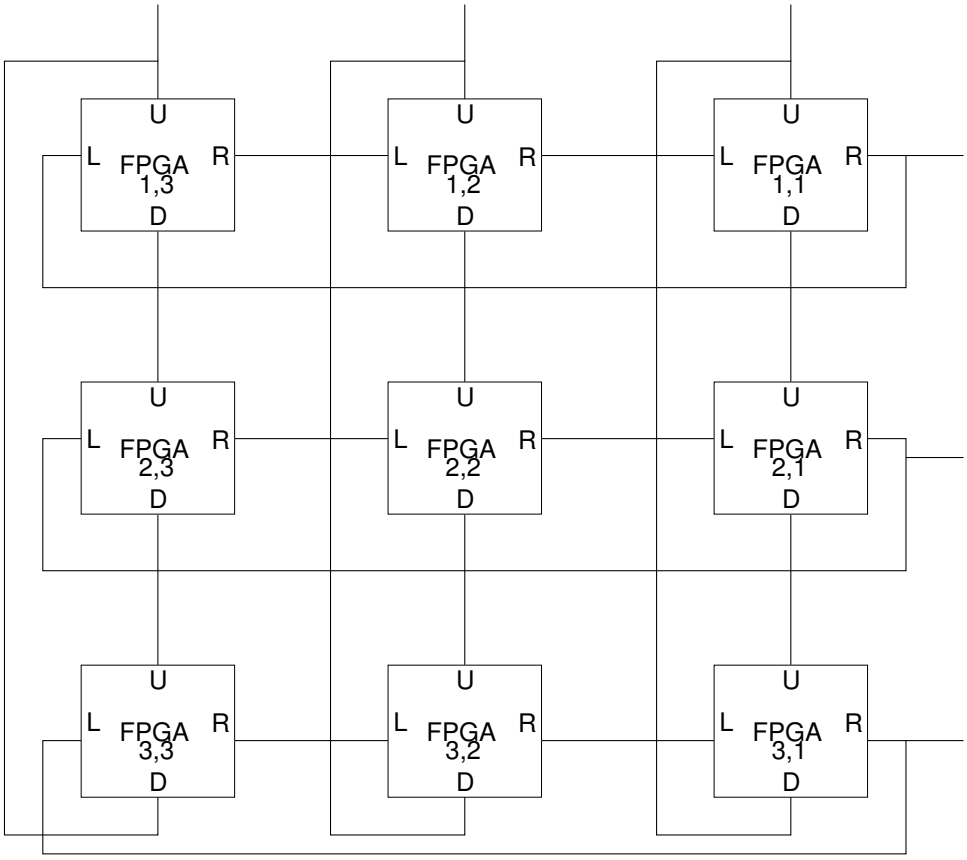

Bild 23 Phyikalische Verdrahtung der lokalen Verbindungen der FPGAs auf dem Rechendfeld.

Jedes FPGA ist mit seinen vier Nachbarn über einen 24 Bit breiten Bus verbunden. Die FPGAs am Rand sind über lange Rückleitungen an den entsprechen-

den Partner auf der anderen Seite der Platine angeschlossen. Ausserdem werden diese Leitungen auf die Ramcontrolleranschlüsse geführt (Bild 23). Die langen Rückleitung bewirken Signallaufzeiten. Erschwerend kommt hinzu, daß gerade diese Rückleitungen auch zum Ramcontroller führen und somit gemultiplext sind, d.h. eine höhere Datenrate fassen müssen. Es gibt ein Verbindungsschema, das den Nachteil der langen Rückleitungen nicht aufweist: überspringt man bei der Verdrahtung immer ein FPGA, so ergibt sich zwischen allen FPGAs ungefähr dieselbe Leitungslänge (Bild 24). Somit könnten beliebig große Rechenfelder realisiert werden, ohne daß es zu Laufzeitproblemen auf dem Rechenfeld käme.

Aus mehreren Gründen wurde diese Schema aber trotzdem nicht verwendet. Zum einen ist die Verdrahtung auf der Platine wesentlich schwieriger, weil die Leitungen oft schräg geführt werden müssen, was bei einem Bus von 24 Leitungen einen erheblichen Platzbedarf bedeutet. Des weiteren ist der Gewinn dieses Verdrahtungsschemas bei einem 3x3–Rechenfeld relativ bescheiden, denn die Rückleitungen nach dem einfachen Verdrahtungsschema sind nur unwesentlich länger, als bei der Verdrahtung nach Bild 24. Und drittens führt die Umplazierungen der einzelnen FPGA zu erheblichen Komplikationen bei der Zuordung der Busse am FPGA.

Wenn man davon ausgeht, daß der implementierte Algorithmus regelmäßig aufgebaut ist, wird jedes FPGA auf der Rechenfeldplatine annähernd denselben Inhalt bekommen. Verdrahtet man die FPGAs nach dem in Bild 24 gezeigten Schema, ändern sich nicht nur ihre Positionen auf dem Rechenfeld, sondern es drehen sich teilweise die Zuordungen von ”links” und “rechts” bzw. “oben” und “unten” um. Betrachten wir dazu FPGA (1,2). Verglichen mit Bild 23 zeigt sich, daß die Anschlüsse links und rechts vertauscht sind. Das bedeutet, daß man entweder den Inhalt des Chips entsprechend “umdrehen”, oder den Chip auf der Platine anders anschließen muß.

Dreht man den Chipinhalt um, handelt man sich interne Unregelmäßigkeiten ein, weil die FPGAs nicht völlig homogen aufgebaut sind. Die Anschlüsse auf der Platine zu ändern ist aber auch relativ unrealistisch, denn um die Anordnung entsprechend Bild 24 zu erreichen, müßten die Chip teilweise spiegelverkehrt, d.h. von der Unterseite der Platine eingelötet werden. Außerdem verliert man so die Regelmäßigkeit im Layout. Deshalb wurde das einfachere Layout nach Bild 23 gewählt.

An jeden der vier Busse eines FPGA ist ein 26pol Pfostenstecker angeschlossen (JM10 – JM42). Er kann einerseits zu Meßzwecken dienen, andererseits ermöglicht er es, das Rechenfeld zu verkleinern. Dazu überbrückt man die nicht benötigten Rechenfeld-FPGA—Sockel (die FPGAs sollten in diesem Fall aus den Sockeln ausgebaut werden !) an ihren Pfostensteckern mit Flachbandkabeln. So ist es einfach möglich, auch ein beliebig kleineres Rechenfeld zu erhalten.

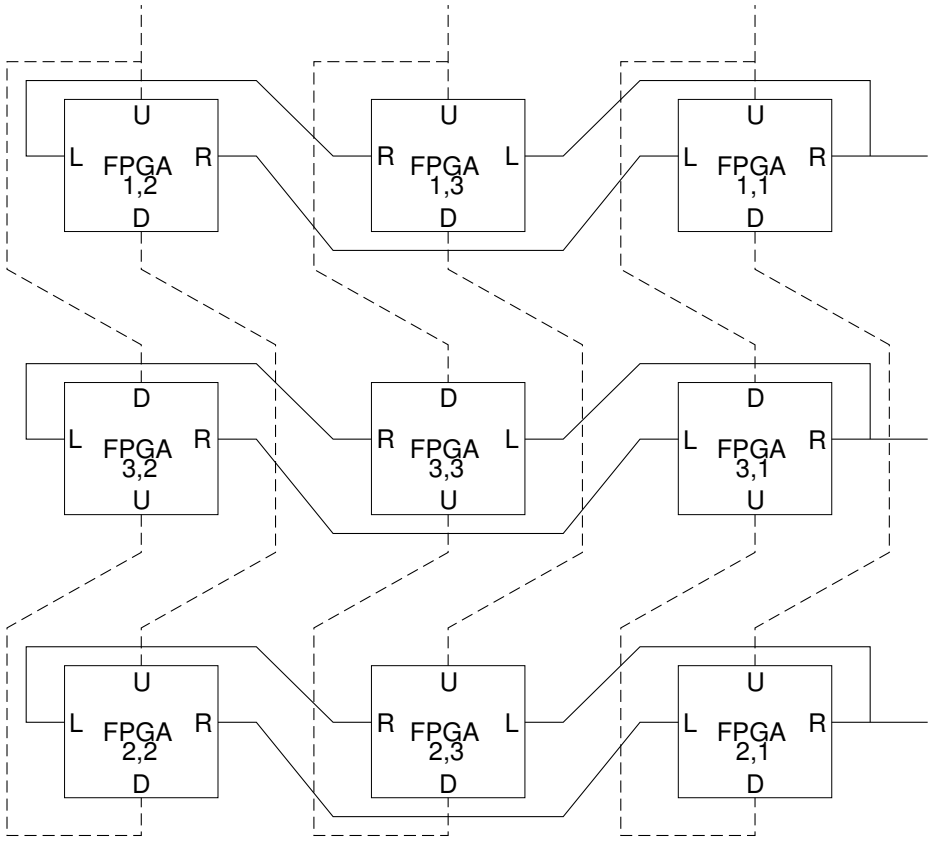

Bild 24 Mögliches Verdrahtungsschema der FPGAs des Rechenfeldes mit konstanten Leitungslängen.

5.3 Ramcontroller

Der Ramcontroller ist die komplexeste Platine des Projekts. Sie enthält neben dem eigentlichen Ramcontroller-FPGA mit Program&Readback Buffer den Demultiplexer zum Anschluß an den IFBUS, zwei Steckplätze zur Aufnahme der Ramkarten, sowie den Anschluß an das Rechenfeld. Der Rechenfeld-Anschluß ist als VG64–Leiste ausgeführt und liefert den lokalen Datenbus vom Rechenfeld sowie den PRBUS, den GPBUS und das ComCLK-Signal. Ein 60poliges Flachbandkabel führt den IFBUS und die Signale $\overline{\text{BOE}}$ und $\overline{\text{RAMWE}}$ direkt vom Interface zum Ramcontroller.

Die Steckerbelegungen sind im Anhang aufgeführt. Ebenfalls im Anhang ist der Schaltplan abgedruckt.

Der Ramausbau sollte möglichst flexibel möglich sein. Dies wurde im Kapitel 4.2.3 bereits mit der Einführung des Bankings ermöglicht. Um in der Realisierung die durch das Banking gewonnene Flexibilität nicht zu verlieren, wurden die Rambausteine nicht direkt auf der Ramcontroller-Platine untergebracht, sondern werden über zwei 50poligen Steckplätze angeschlossen. Diese können jeweils eine Platine aufnehmen, die die RAMs trägt (Ramcard). Da diese Ramcard nur

die gewünschten Speicherbausteine ohne Zusatzbauteile tragen muß, ist sie einfach zu entwerfen und einfach an die gewünschten Verhältnisse anpaßbar.

Die realisierte "Ramcard I" stellt die einfachste Möglichkeit dar, die RAMs anzuschließen. Sie ist in der Lage, entweder drei 8Kx8 RAMs oder drei 32Kx8 RAMs aufzunehmen, also genau eine Bank. Jumper ermöglichen die Auswahl einer der vier Banks, an die die RAMs gelegt werden sollen. Mit einer zweiten Karte diesen Typs läßt sich der Speicher verdoppeln, indem man dort einfach die nächste Bank durch Jumpern auswählt. Erst wenn mehr als zwei Banks genutzt werden sollen, muß eine neue Platine entworfen werden, die mindesten zwei Banks auf einmal ansprechen kann.

Alle Daten- und Adreßleitungen sind mit Abschlußwiderständen versehen, um die Reflektionen am Leitungsende so gering wie möglich zu halten. Diese Abschlußwiderstände sind mit den Widerstandsnetzwerden RN1 - RN10, sowie den Widerständen R4 - R8 realisiert (siehe auch Kapitel 5.5).

Die für das Demultimplexen des IFBUS benötigten Tristatetreiber sind ebenfalls auf der Platine untergebracht (IC 2– IC 7). IC 2 bis IC 4 puffern den Zugriff auf den IFBUS(WRITE), also die Datenausgabe vom Ramcontroller zum Interface. Ihre Freigabe erfolgt durch das $\overline{\text{RCBOE}}$-Signal wie in Kapitel 4.2.2 erläutert. Die Treiber IC 5 bis IC7 sind für die Gegenrichtung zuständig und werden von dem globalen Signal $\overline{\text{BOE}}$ gesteuert. Der Widerstand R11 sorgt im nichtprogrammierten Zustand des FPGAs dafür, das die Treiber inaktiv sind.

Für Meß– und Kontrollzwecke sind die Stecker JM1 bis JM3 vorgesehen. An JM1 liegt der Datenbus Richtung Rechenfeld, während JM2 die Daten vor den Demultiplexern zum Interface führt. JM3 schließlich stellt die Anschlüsse des PRBUS hinter dem Tristatetreiber direkt am FPGA zur Verfügung. Die Belegung dieser Meßpunktstecker ist am Anhang aufgeführt (JM1 und JM2 sind vom Typ B, JM3 vom Typ A).

5.4 Interface Adapter

Beim Interface-Adapter laufen alle Fäden der Kommunikation zusammen.

Der Anschluß an das Interface erfolgt über zwei VG64 Leisten, wovon eine zum FPGA für die Leserichtung, die andere zum FPGA für die Schreibrichtung führt. Die Belegungen dieser Stecker ist zusammen mit den Schaltplänen wie üblich im Anhang aufgeführt.

5.4.1 Signalverteilung

Die Ramcontroller werden über den horizontalen (mittels CN4) und den vertikalen (mittels CN5) IFBUS angeschlossen. Diese Busse sind als Flachbandverbin-

dungen ausgeführt und am Ende mit Terminierungswiderständen abgeschlossen, um die Reflektionen am Leitungsende so gering wie möglich zu halten.

Die Verbindung CN3 zum Rechenfeld umfaßt den GPBUS und PRBUS mit den Adreßleitungen zum Selektieren der FPGAs (FADR0–7), sowie den globalen Takt ComCLK.

5.4.2 Taktverteilung

Neben der Verteilung der Signale auf die Rechenfeld– und Ramcontrollerplatinen, wird auf der Interface–Adapter–Platine der globale Takt BSCLK erzeugt, und über Treiber werden die einzelnen Taktsignale verstärkt.

Die Grundtakterzeugung geschieht mittels eines TTL-Quarzoszillators, dessen Ausgang direkt über den Interface-Anschluß #2 (CN2) zum FPGA #2 des Interface führt. Dort ist, wie in Kapitel 4.3 beschrieben, die programmierbare Takterzeugung untergebracht. Als Signale kommen von dort die Leitungen RAMWE_out, ComCLK_out, BOE_out, CLKa_out bis CLKd_out, sowie GTBUF zurück. Alle Signale werden durch Tristate-Treiber 74F245 verstärkt. Dies bewirkt eine einheitliche Verzögerung aller Taktsignale ohne Phasenverschiebung. Deshalb werden die für den Betrieb des Interface benötigten Taktsignale von hier auch wieder an das Interface zurückgegeben und nicht direkt innerhalb der Interface-FPGAs geroutet.

Das Signal GTBUF dient dazu, den Treiber U2 zu deaktivieren, wodurch wegen der nachgeschalteten Pullup-Widerstände alle Ausgänge auf high gehen. Damit ist es möglich, das Rechenfeld anzuhalten (Schaltbild Anhang F, Bild 41). An U2 werden die Signale BOE_out und RAMWE_out geteilt und jeweils über einen eigenen Treiber verstärkt. Die Signale führen dann zum horizontalen bzw. vertikalen IFBUS. Somit muß nicht ein Treiber alle RAMs ansteuern, sondern zwei können sich die Arbeit teilen, was einen weniger stark verzerrten Signalverlauf zur Folge hat.

Die Signale, die über U3 geführt werden, können nicht abgeschaltet werden. Dies muß so sein, da hiervon auch der Takt des Interface selbst herkommt, der natürlich nicht mit angehalten werden darf, wenn das Rechenfeld stehen bleibt.

5.4.3 General Purpose Bus

Für den GPBUS wurde ein Jumper-Feld auf der Interface-Adapter-Platine vorgesehen, um diese Leitungen möglichst flexibel nutzen zu können.

So sollen zum einen zusätzliche Clocksignale verteilt werden können. Eine denkbare Anwendung eines weiteren Clocksignals könnte beispielsweise in der Verwendung als Takt für die neun FPGAs des Rechenfeldes liegen. Diese FPGAs könnten dann mit einer höheren Frequenz arbeiten. Auch einen Datenaustausch

zwischen diesen FPGAs wäre noch möglich, da sich die Kommunikation der Rechenfeld-FPGAs untereinander relativ einfach gestaltet (vgl. 4.1.2). Nur die Kommunikation mit den Ramcontrollern müßte mit der langsameren Frequenz ComCLK erfolgen.

Zum andern sollen globale Signale vom Interface an die FPGAs weitergeleitet werden können oder umgekehrt globale Signale von den FPGAs zum Interface gelangen. Da das Interface getrennte FPGAs für Lese- und Schreibrichtung hat, sollte der IFBUS teilweise an das eine oder andere Interface-FPAG angeschlossen werden können.

Diese Aufgaben werden mit einem Jumper-Feld (vgl. Bild 42) realisiert. Die Bits 0–3 des GPBUS können mittels der Jumper JP22–JP25 an die zusätzlichen Clocksignale CLKa-CLKd angeschlossen werden. Sollte dies nicht erwünscht sein, können diese Bits über die Jumper JP18–JP21 zur Aufteilung an den GPin– bzw. GPout-Bus weitergeleitet werden. Das Interface stellt zwei Richtungen für die möglichen globalen Leitungen des GPBUS zur Verfügung : acht Leitungen führen zum Rechenfeld hin (GPin0–7), acht weitere vom Rechenfeld weg (GPout0–7). Die Zuordnung des GPBUS zu dem GPin- bzw. GPout-Bus erfolgt über die Jumper JP2–JP17.

5.5 Layout

5.5.1 Stromversorgung

Da das gesamte System von PAr^2 synchron arbeitet, wurde bei der Verteilung der Stromversorgung mit Sorgfalt gearbeitet. Das gleichzeitige Schalten aller Bauteile kann nämlich erhebliche Stromspitzen mit sich bringen.

So wurde die Stromversorgung für jeden Ramcontroller extra geführt, für das Rechenfeld wurden drei Stromversorungsanschlüsse vorgesehen. Die Versorgungsleitungen treffen sich erst direkt am Netzteil. Es gilt nämlich, die gemeinsamen Leitungswege zweier Stromkreise so kurz wie möglich zu halten, um die galvanische Kopplung zu minimieren. Optimal wäre hierfür eine streng sternförmige Spannungsverteilung, bei der alle Verbraucher direkt an einem gemeinsamen Punkt angeschlossen sind. Dies läßt sich bei einem so umfangreichen Projekt wie dem Rechenfeld aber nicht realisieren. So wurden die VCC-Versorungsleitungen sternförmig zu den einzelnen Platinen geführt und dort möglichst günstig verteilt. Insbesondere wurde darauf geachtet, daß sich keine Schleifen in der VCC-Versorung ausbilden.

Zusätzlich wurde vor jedem Anschluß eines Zweiges an das Netzteil ein Tiefpaß in Form eines Π-Gliedes zwischengeschaltet (Bild 25). Diese Tiefpässe sollen hochfrequente Störsignale, die durch das Schalten der Treiber entstehen,

auf den lokalen Versorgungszweig beschränken, in dem sie entstehen. Die Spule L besteht dabei nur aus fünf Windungen Kupferdraht. Da neben den sechs Ramcontrollern und den drei Anschlüssen des Rechenfeldes auch der Interface-Adapter mit Spannung versorgt werden muß, wurde dieser Tiefpaß insgesamt zehnmal benötigt.

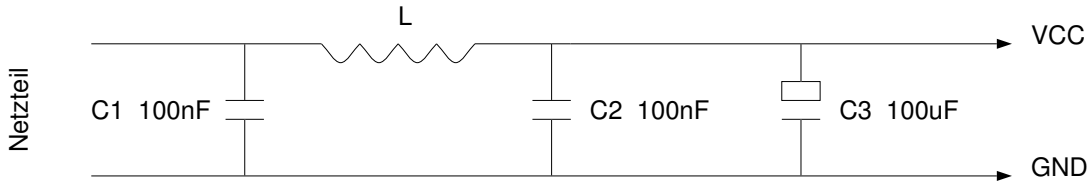

Bild 25 Π-Tiefpaßfilter in den Versorgungsleitungen

Die Stromversorgung des Oszillators, der das BSCLK-Signal erzeugt, wurde noch einmal zusätzlich gefiltert. Dies erwies sich als notwendig, da sich die sehr hohen Frequenzen des Grundtaktes sonst zu stark auf die Versorgungsspannung übertragen hätten. Dieser Filter ist in Bild 26 zu sehen.

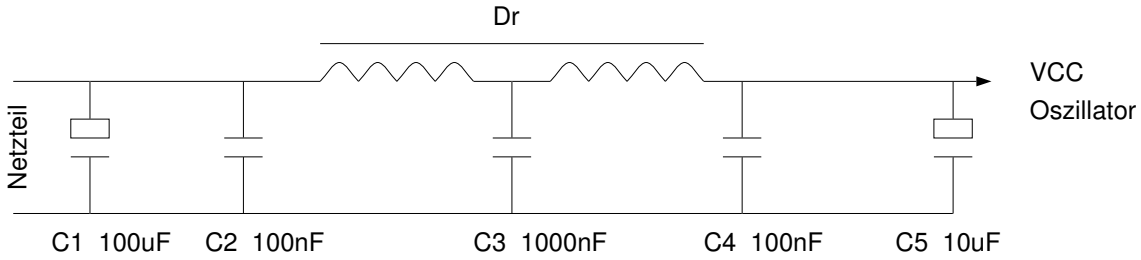

Bild 26 Zusätzlicher Filter in der Stromversorungsleitung des Grundtaktoszillators

Eine weitere Schutzmaßnahme vor Spannungsspitzen bieten die reichlich vorhandene Abblockkondensatoren. Es wurden nicht nur IC-Sockel mit eingebautem Abblockkondensator verwendet, sondern zusätzlich möglichst nah an “Störquellen” wie RAMs und FPGA Kondensatoren angeschlossen.

5.5.2 Signalleitungen

Da auf allen Taktleitungen sowie dem Interface-Bus relativ hohe Frequenzen auftauchen, wurden diese Leitungen mit Abschlußwiderständen versehen, um Reflektion am Leitungsende zu minimieren. Für die Berechnung dieser Abschlußwiderstände, muß man den Wellenwiderstand der Leitungen berücksichtigen und versuchen, einen Abschluß möglichst nahe an diesem Wellenwiderstand zu finden. Denn nur bei einem Abschluß mit dem Wellenwiderstand entstehen keine

Reflektionen am Leitungsende. Dabei muß als zweite Beschränkung der maximale Ausgangsstrom der Treiber berücksichtigt werden, der nicht überschritten werden darf.

Bild 27 zeigt das Prinzip des Abschlusses, der als Thevenin-Terminierung bezeichnet wird [3]. Hochfrequenzmäßig sind VCC und GND kurzgeschlossen, wodurch sich als Abschlußwiderstand die Parallelschaltung der beiden Widerstände ergibt:

$$R_a = \frac{R_1 * R_2}{R_1 + R_2}$$

Als Belastung für den Ausgang ergibt sich im Low-Zustand (U_{oL} am Ausgang des Treibers) der Strom I_L, zu :

$$I_L = \frac{U_{VCC} - U_{oL}}{R_1} - \frac{U_{oL}}{R_2} \approx \frac{U_{VCC}}{R_1}$$

für $U_{oL} \approx 0$. Entsprechend erhält man für den High-Zustand (U_{oH}) die Belastung zu

$$I_H = \frac{U_{VCC} - U_{oH}}{R_1} - \frac{U_{oH}}{R_2} \approx -\frac{U_{VCC}}{R_2}$$

für $U_{oH} \approx U_{VCC}$

Bild 27 Thevenin-Terminierung eines Gatterausgangs

Eine genaue Berechnung des Wellenwiderstandes der Leitung ist sehr aufwendig, da die Geometrie der Leiterbahn berücksichtigt werden muß. Da das Signal im allgemeinen über mehrere Stecker und somit auch unterschiedliche Leitungsformen geführt wird, wurde der Einfachheit halber ein Schätzwert von 100Ω zugrunde gelegt [3].

Die Treiber der 74F-Serie können leicht 24 mA treiben, so daß sich für $R_1 = 180\Omega$ und $R_2 = 220\Omega$ ergeben. Der Wellenwiderstand ergibt sich dann zu $R_a = 99\Omega$. Die Ausgänge der FPGAs können aber nur 12mA treiben, womit sich rechnerisch $R_1 = 360\Omega$ und $R_2 = 440\Omega$ ergibt. Um nicht bis an die

Belastungsgrenze der FPGAs zu gehen, wurden beide Widerstände zu $R_1 = R_2 = 470\Omega$ gewählt. Daraus folgt eine hochfrequenzmäßiger Abschlußwiderstand von 235Ω. Einige Leitungen wie der Bus zwischen Ramcontroller und Rechenfeld sind bidirektional, deshalb wurden in einem solchen Fall Abschlußwiderstände an beiden Enden der Leitung vorgesehen, die dann jeweils den Wert $1k\Omega$ haben. Der selbe Wert wurde für die beiden Abschlüsse des Interface-Bus gewählt, der sich in einen horizontalen und einen vertikalen Bus aufteilt und deshalb auch zweimal abgeschlossen wird.

Die resultierende Abschlußwiderstände weichen zwar relativ stark von dem zugrundegelegeten Wellenwiderstand von 100Ω ab, für die Praxis erwiesen sich diese Abschlüsse jedoch als ausreichend.

Ein besonderes Problem stellt die Leitung für BSCLK dar, die 64 MHz über ein Flachbandkabel vom Interface-Adapter zum Interface transportieren muß. Dies führte in der Testphase zu erheblichen Problemen. Deshalb wurde für dieses Signal das Flachbandkabel maßgeschneidert: Die Leitung, die das Signal BSCLK transportiert, wurde an beiden Enden aufgeschnitten und durch ein abgeschirmtes Kabel ersetzt.

Bei der Konzeption der Platinen wurde die Wichtigkeit der Abschlußwiderstände unterschätzt, weshalb keine Abschlußwiderstände auf dem Bus zwischen Rechenfeld und Ramcontroller vorgesehen wurden. Sie werden jetzt nachträglich über die Meßstecker auf der Ramcontroller- bzw. Rechenfledplatine angeschlossen. Ebenfalls nachträglich wurden die Abschlußwiderstände für das Taktsignal ComCLK auf der Unterseite der Rechenfeldplatine eingebaut. Das Signal ist jetzt dreimal mit jeweils $1k\Omega$ gegen Masse und VCC abgeschlossen. Problematisch ist der Abschluß der zusätzlichen Taktsignale, die über den global Bus transportiert werden sollen. Da hier nicht klar ist, welches Signal von einem 74F-Treiber angesteuert wird (im Fall des Clocksignals) und welches von einem FPGA-Ausgang (im Fall eines globalen Steuersignals), können nicht alle Abschlüsse festgelegt werden. So wurde für die Verteilung des CLKa-Signals die entsprechende Leitung terminiert, während alle anderen GPBUS-Signale offen sind.

5.6 Test

Da zum Zeitpunkt der Fertigstellung dieser Diplomarbeit das Interface noch nicht verfügbar war, konnten nur einige grundlegende Tests in der Kommunikation durchgeführt werden, wobei das XC4000 Demoboard von XILINX als Interface-Ersatz diente [16]. Die Programmierung erfolgte mit Hilfe der XChecker Soft- und Hardware von XILINX [17]. Sie ermöglicht es, ein FPGA über die serielle Schnittstelle eines Rechner zu programmieren.

Im einzelnen wurden folgende Tests durchgeführt:

- **Testen der Adreßdecodierung und Programmierung** mehrerer FPGAs nacheinander.
- **Test der Kommunikation zwischen Interface und Ramcontroller.** Es wurden auf denselben Bits des IFBUS Daten gelesen und geschrieben, um den Multiplexmechanismus zu testen. Die Datenübertragung lief auch mit dem “letzten” Ramcontroller am Interfacebus erfolgreich bei 8 MHz.
- **Test der Kommunikation zwischen Ramcontroller und Rechenfeld.** Auch hier wurde auf denselben Bits Daten in beide Richtungen geschickt, um den Multiplexmechanismus zu überprüfen, sowie die am weitesten auseinanderliegenden FPGAs angeschlossen, um den Fall der längsten Signallaufzeit abzudecken. Dabei erwies es sich als notwendig, am Ramcontroller Abschlußwiderstände an den Bus anzuschließen (vgl. vorangegangenen Abschnitt). Dieser Test lief ebenfalls bei 8MHz erfolgreich.
- **Test der Kommunikation zwischen Ramcontroller und RAM.** Auch dieser Test verlief bei 8MHz erfolgreich, sowohl bezüglich des Adreß- als auch des Datenmultiplexens.
- **Test der Kommunikation innerhalb des Rechenfeldes.** Diese Kommunikation ist die einfachste, da sie ohne Multiplexen auskommt. Deshalb war der erfolgreiche Test bei 8MHz zu erwarten. Als weitergehender Test wurde diese Kommunikation mit Hilfe eines zusätzlichen Taktsignals, das mit CLKa erzeugt und über GPBUS0 verteilt wurde, bei 16MHz getestet und verlief ebenfalls erfolgreich. Damit ist es prinzipiell möglich, den inneren Teil des Rechenfeldes mit der doppelten Taktfrequenz laufen zu lassen wie die Ränder bzw. die Ramcontroller. Dies stellt aber hohe Anforderungen bei der Entwicklung eines Algorithmus, die in jedem Fall in COMPAR zu erbringen wären.

Da die Korrektheit der Signale nur mit einem Logic Analyser festgestellt werden konnte, wurden jeweils nur vier Bit der 24 Bit breiten Busse getestet.

5.7 Verbesserungsvorschläge zur Realisierung

Hardwaretechnisch müßte mehr Gewicht auf den korrekten Abschluß aller Leitungen mit Abschlußwiderständen gelegt werden. Der Hochfrequenzaspekt der Signale bei 8 MHz wurde unterschätzt. Schließt man an den Treiber eines 8MHz-Signales nur offenes Stück Kabel von 40–50 cm wird die Signalform durch Reflektionen am Leitungsende so stark überlagert und mit Überschwingern versehen, daß von der ursprünglichen Signalform faßt nichts mehr übrig bleibt. Als Verbesserung sollten bei künftigen Platinenlayouts alle Leitungen, die von FPGAs gesteuert werden mit mindestens jeweils ein 1kΩ gegen VCC und GND

abgeschlossen werden. Diese Fehleinschätzung ist aber nicht so gravierend, da die Terminierungen ohne Probleme an den vorhandenen Meßsteckern angeschlossen werden können.

6 Zusammenfassung

In der vorliegenden Arbeit wurde ein zweidimensionales, synchrones Rechenfeld entworfen und realisiert, auf dem Algorithmen abgearbeitet werden können, die mit dem Entwurfssystem COMPAR erzeugt werden.

Das Rechenfeld basiert auf XILINX FPGAs vom Typ XC4005. Es stellt eine Gatterkapazität von 45000 Gattern bei einer Taktrate von 8 MHz zur Verfügung. Die Datenbusbreite beträgt 24 Bit. Pro Ramcontroller können maximal 384kB RAM angeschlossen werden, was einem Gesamtspeicher von 2304kB entspricht. Die maximale Datenrate zum Interface beträgt 24 MB/s sowohl für die Lese- als auch die Schreibrichtung, insgesamt also 48 MB/s.

In dieser Arbeit wurden die Kommunikationsschemata der Teilprozessoren (FPGAs) aus dem Gesamtkonzept von PAr^2 entwickelt. Durch eine Beschreibung der Hardware auf Registerebene und der Einführung einheitlicher Signalbezeichnungen als Schnittstellen innerhalb der FPGAs wurde eine Modellierung des Rechenfeldes unabhängig von der tatsächlichen Hardwarerealisierung gefunden. Dieses Registermodell (Bild 21,Kapitel 4.4) dient als Ausgangspunkt für die Repräsentation der Hardware in abstrakteren Beschreibungen wie sie in COMPAR notwendig sind.

Neben der realisierten Hardware ist zum Betrieb des Rechenfeldes eine Schnittstelle in COMPAR nötig, sowie ein Interface, um das Rechenfeld an einem Host betreiben zu können. Momentan wird an diesem Interface gearbeitet, die Fertigstellung in Kürze wird dann auch ein umfangreiches Testen der Hardware von PAr^2 ermöglichen.

Literaturverzeichnis

[1] U. Arzt, J. Teich, and L. Thiele. The concepts of COMPAR: A compiler for massive parallel architectures. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, pages 681–684, San Diego, May 1992.

[2] K.M. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley Publ. Comp., Reading, Mass., 1988.

[3] R. Dreyer. Strahenschutz, EMV-Aspekte beim Leiterplattendesign. *ELRAD*, pages 56–61, Sept. 1992.

[4] M. Gokhale et.al. Building and using a highly parallel programmable logic array. *IEEE Computer*, pages 81–89, Jan. 1992.

[5] Fairchild. *Fast Fairchild Advanced Schottky TTL. Fast Data Book*. Fairchild, 1985.

[6] H.-J. Herpel and M. Glesner. A rapid prtoryping approach to requirements validation of application specific embedded controllers. In *ITG-Fachberich 122 , ITG-Fachtagung*, pages 23–34, Nov. 1992.

[7] P. Knapp. *Konzept und Implementierung einer Steuerung des PAr^2-Rechenfeldes durch das PSI-Interface*. Diplomarbeit am Lehrstuhl für Mikroelektronik, Universität, des Saarlandes, Saarbrücken, 1993.

[8] J. König. *Programmable SBus Interface*. Lehrstuhl für Mikroelektronik, Universität des Saarlandes, 6600 Saarbrücken, 1993.

[9] J. N. Morris. *Hardware design of a rapid prototyping reconfigurable logic system*. North Carolina State University, ECE Dept., Box 7911, Raleigh, NC 27695-7911, 1992.

[10] V. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the localization of algorithms for VLSI processor arrays. *in: VLSI Signal Processing III, IEEE Press, New York*, pages 459–470, 1989.

[11] Cypress Semiconductor. *CMOS/BiCMOS Data Book*. Cypress Semiconductor,3901 North First St., San Jose, CA 95134, 1992.

[12] J. Teich and L. Thiele. Control generation in the design of processor arrays. *Int. Journal on VLSI and Signal Processing*, 3(2):77–92, 1991.

[13] J. Teich and L. Thiele. A transformative approach to the partitioning of processor arrays. In *Proc. Int. Conf. on Application Specific Array Processors, IEEE Computer Society Press*, pages 4–20, Berkeley, August 1992.

[14] L. Thiele. *Materialien zur Vorlesung Mikroelektronik 3*. Lehrstuhl für Mikroelektronik, Universität des Saarlandes, 6600 Saarbrücken, 1991.

[15]XILINX. *The XC4000 Data Book*. XILINX, 2100 Logic Drive, San Jose, CA 95124, 1991.

[16]XILINX. *XACT Reference Guide, The XC4000 Design Demonstration Board*. XILINX, 2100 Logic Drive, San Jose, CA 95124, 1992.

[17]XILINX. *XACT Reference Guide, XChecker Universal Download/Readback Cable and Logic Probe*. XILINX, 2100 Logic Drive, San Jose, CA 95124, 1992.

[18]XILINX. *The XC4000 Design Implementation User Guide*. XILINX, 2100 Logic Drive, San Jose, CA 95124, 1992.

Anhang A Programm zur Bestimmung des Timing

Zur Bestimmung des optimalen Timings wurde ein C-Programm geschrieben, das im folgenden abgedruckt ist. In der Datei `calctime.h` (Listing 1) sind alle Zeiten vordefiniert, die Datei `calctime.c` (Listing 2) enthält das eigentliche Programm. Es entspricht den in Kapitel 4.3 hergeleiteten Zeitabhängigkeiten. Das Ergebnis des Programmlaufs zeigt Listing 3.

```
/* CALCTIME.H */

/* Delaytime-definitions follow */
#define Tshort  3 /*max time two outputs connected
   while switching to tristate */

#define Tch 0 /*min data hold after clock */
#define Tfza 20  /*max FPGA PAD clock to PAD tristate
   to active (empirisch) */
#define Tfaz    20
#define Tpick   16 /*min FPGA data vaild before clock */
#define Tokop 12
#define Tpar 20 /*max delay RAM adresse read from
   RCMux to PAD (empirisch)*/
#define Tpaw 37  /*max delay clk low to RAM adresse
   write valid on PAD (empirisch)*/

/* External Buffers of the IFBus 74F245*/
#define Tbza 8 /*max Buffer tristate to active */
#define Tbd 6 /*max Buffer propagation delay */
#define Tboz Tfza-Tbza /*min Buffer outputs disable */

/* RAM-Specifications: 12ns RAM CY7B185/186 */
#define Trdoe 6 /*max: RAM OE low to data valid */
#define Taa 12 /*min: RAM adress to data valid */
#define Tsa 0 /*min: Adress set-up to write start */
#define Twr 8 /*min: RAM write pulse width */
#define Tdw 0 /*min: Data valid to WE */

#define Tdh     Tpick+Tbd /*min: data hold valid before clock */

#define Tdrf 8 /* Propagation delay on RF-LOOP wire */
#define Tdif 11 /* Propagation delay on IFBUS wire */

#define resolution 8 /* Number of Segments between one Clocktime */
```

Listing 1 calctime.h

```
/*
   Optimal timing computation for FPGA-Array
   Tobias Blickle 11.9.92
*/

#include "calctime.h"
#include <stdio.h>
#define rows 13
#define LARGE 1E20
set_up_table(LPA,LPb,a,b,c,d,e)
int *LPA;
int *LPb;
int a,b,c,d,e;
{
 /*input data follows , see documentation*/

 LPA[0]=b-a;  LPb[0]=Tfza-Tpick;
 LPA[1]=a; LPb[1]=2*Tdrf+Tpick+Tfza;
 LPA[2]=a-c; LPb[2]=Tbza+Tpick;
 LPA[3]=a; LPb[3]=Tokop+Tbd+Tpick;
 LPA[4]=c; LPb[4]=Tfaz-Tshort-Tbza;
 LPA[5]=b-a; LPb[5]=2*Tdif+Tpick+Tokop+Tbza;
 LPA[6]=b-a; LPb[6]=2*Tdif+Tfza+Tpick;
 LPA[7]=a; LPb[7]=Tpar+Taa+Tpick;
 LPA[8]=a-d; LPb[8]=Trdoe+Tpick;
 LPA[9]=d; LPb[9]=Tfaz-Tshort-Trdoe;
 LPA[10]=e-a; LPb[10]=Tpaw+Tsa;
 LPA[11]=e-a; LPb[11]=Tfza+Tdw;
 LPA[12]=b-e; LPb[12]=Twr;
}
main()
{
 int *A=(int *)calloc(rows,sizeof(int));
 int *b=(int *)calloc(rows,sizeof(int));
 int rueck;
 int am,cm,dm,em;
 int aopt,copt,dopt,eopt;
 double val=LARGE;
 double minimum;
 char *command="cat calctime.h\n";
 printf("Computing optimal timing for FPGA array.\n\n");
 printf("The maximal overlay time of two outputs beeing active\n");
 printf("at the same time ist set to %d ns.\n",Tshort);
 printf("The following restrictions are set:\n\n");
 fflush(stdout);
 system(command);

 /* Enumerate all possible values */
 for (am=1;am<resolution;++am)
  for (cm=1; cm<resolution;++cm)
    for (dm=1;dm <resolution;++dm)
```

```
      for (em=1;em <resolution;++em)
 {
  set_up_table(A,b,am,resolution,cm,dm,em);
  rueck=get_minimum(A,b,rows,&minimum);
  if (rueck==0 && minimum<val)
  {
  val=minimum;aopt=am;copt=cm;dopt=dm;eopt=em;
  }
 }
if (val==LARGE)
 {
 printf("No solution was found !\n");
 printf("Please check restrictions !\n");
 }
else
 {
 printf("Soloution found:\n");
 printf("\ta\tb\tc\td\te\t   tau[ns]\t  BSCLK[MHz]\t   ComCLK[Mhz]\n");
 printf("\t%d\t%d\t%d\t%d\t%d\t%10.5f\t%10.5f\t%10.5f\n",
  aopt,resolution,copt,dopt,eopt,val,1000/val,125/val);
 }
printf("\nTo edit other restrictions change the file 'calctime.h'\n");
printf("and re-run 'make calctime' to compute output.\n");
return(0);
}
/*
  Determines the minimum x,
  subj. to   a*x >= b

  returns 0, if solution is valid
*/
int get_minimum(a,b,r,min)
int *a,*b,r;
double *min;
{
 register int i;
 *min=-LARGE;

 for (i=0;i<r;++i)
 {
 if (a[i]==0) return (-2);
 if (a[i]<0 && b[i]>0) return (-3);
 if ((double)b[i]/(double)a[i]>=*min)
  *min=(double)b[i]/(double)a[i];
 }

 if (*min==-LARGE) return -1;
 return 0;
}
```

Listing 2 calctime.c

```
Computing optimal timing for FPGA array.

The maximal overlay time of two outputs beeing active
at the same time ist set to 3 ns.
The following restrictions are set:

/* Delaytime-definitions follow */
#define Tshort  3 /*max time two outputs connected
   while switching to tristate */

#define Tch 0 /*min data hold after clock */
#define Tfza 20  /*max FPGA PAD clock to PAD tristate
   to active (empirisch) */
#define Tfaz    20
#define Tpick   16 /*min FPGA data vaild before clock */
#define Tokop 12
#define Tpar 20 /*max delay RAM adresse read from
   RCMux to PAD (empirisch)*/
#define Tpaw 37  /*max delay clk low to RAM adresse
   write valid on PAD (empirisch)*/

/* External Buffers of the IFBus 74F245*/
#define Tbza 8 /*max Buffer tristate to active */
#define Tbd 6 /*max Buffer propagation delay */
#define Tboz Tfza-Tbza /*min Buffer outputs disable */

/* RAM-Specifications: 12ns RAM CY7B185/186 */
#define Trdoe 6 /*max: RAM OE low to data valid */
#define Taa 12 /*min: RAM adress to data valid */
#define Tsa 0 /*min: Adress set-up to write start */
#define Twr 8 /*min: RAM write pulse width */
#define Tdw 0 /*min: Data valid to WE */

#define Tdh     Tpick+Tbd /*min: data hold valid before clock */

#define Tdrf 8 /* Propagation delay on RF-LOOP wire */
#define Tdif 11 /* Propagation delay on IFBUS wire */

#define resolution 8 /* Number of Segments during one Clocktime */
Soloution found:
 a b c d e    tau[ns]   BSCLK[MHz]  ComCLK[Mhz]
 4 8 1 1 7   14.50000   68.96552    8.62069

To edit other restrictions change the file 'calctime.h'
and re-run 'make calctime' to compute output.
```

Listing 3 Ergebnis des Programmlaufes von calctime

Anhang B Technische Angaben Rechenfeld

Der Schaltplan des Rechenfeldes ist auf der folgenden Seite abgedruckt. Die Reproduktionsqualität ist leider nicht sehr gut, da das Orginal im Format DIN A0 erstellt ist. Dieser Plan enthält nicht die auf der Platine vorhandenen Meßstecker. An jedem FPGA ist für jede Richtung ein Meßstecker vorhanden, der den lokalen Bus in einem 26poligen Wannenstecker zugänglich macht. Mit diesen Stecker ist es möglich, auch kleinere Rechenfelder zu realiseren, indem die nicht benötigten FPGAs überbrückt.

Der Schaltplan des Blocks PRGRBK, der die Tristatetreiber enthält, ist in Bild 36 im Anhang E gezeigt. Er gilt gleichermaßen für die Rechfeld- wie für die Ramcontrollerplatine.

Auf den folgenden Seiten sind die Pinbelegungen der Stecker CN1–CN7 der Rechenfeld-Platine aufgelistet. Die Belegungen der Meßstecker JM1 – JM45 sind in Anhang G gegeben. Die Stecker JM1–JM9 sind vom Typ A, während JM10–JM45 vom Typ B sind.

Zu den Verbindungen CN2–CN7, die zu den Ramcontroller führen ist folgendes zu bemerken: Da es sich bei diesen Stecker und seinem Gegenstück auf der Ramcontroller-Platine um VG64–Verbinder mit 90° Anschlüssen handelt, ist die Nummerierung — obwohl die Stecker miteinander verbunden werden — spiegelverkehrt. Dies ist bereits in der hier aufgeführten Belegungstabelle berücksichtigt.

Bild 28 Schaltplan des Rechenfeldes

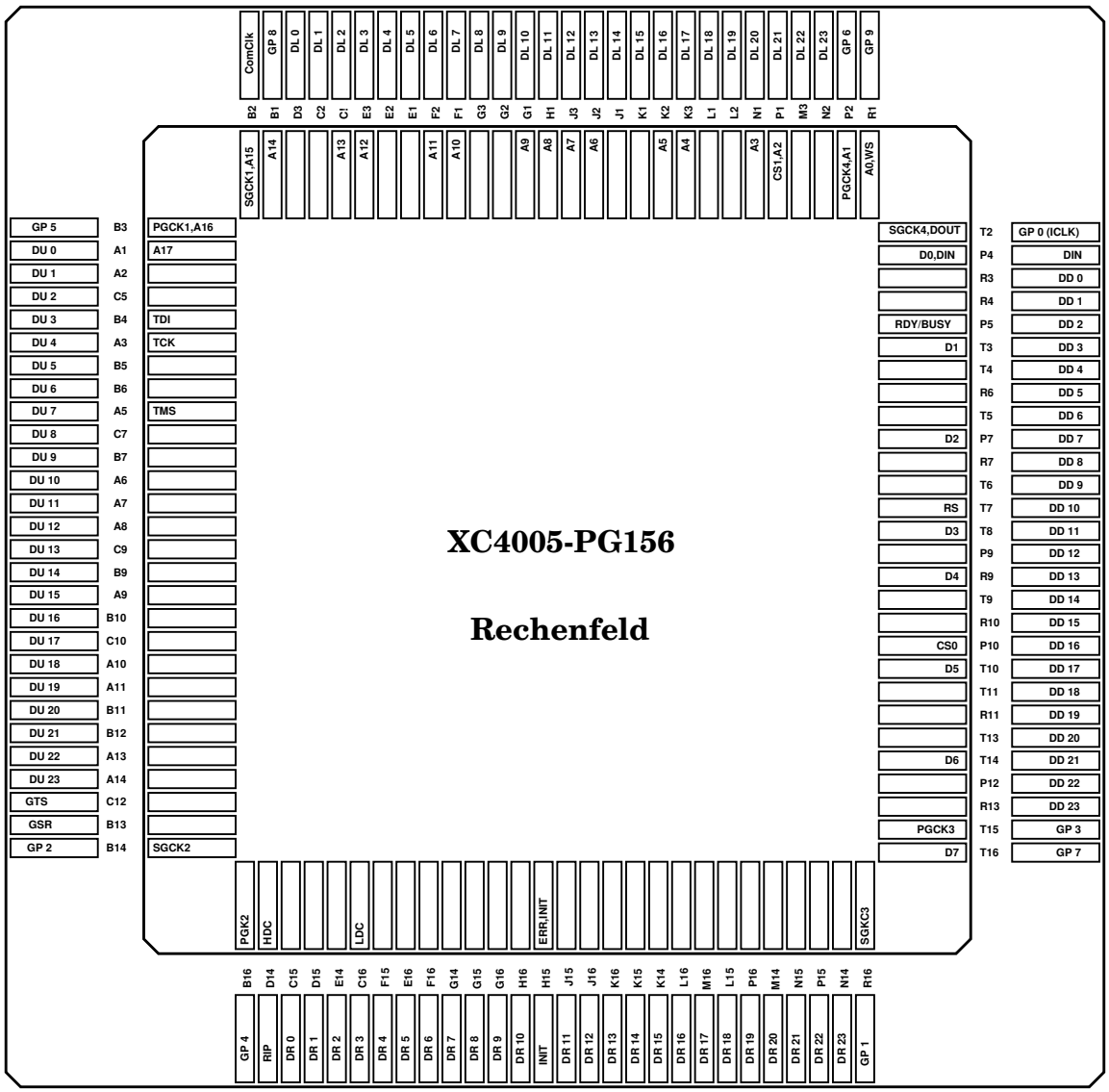

Bild 29 Pinbelegung eines Rechenfeld-FPGAs (Sicht entsprechend dem XILINX-Tool XACT)

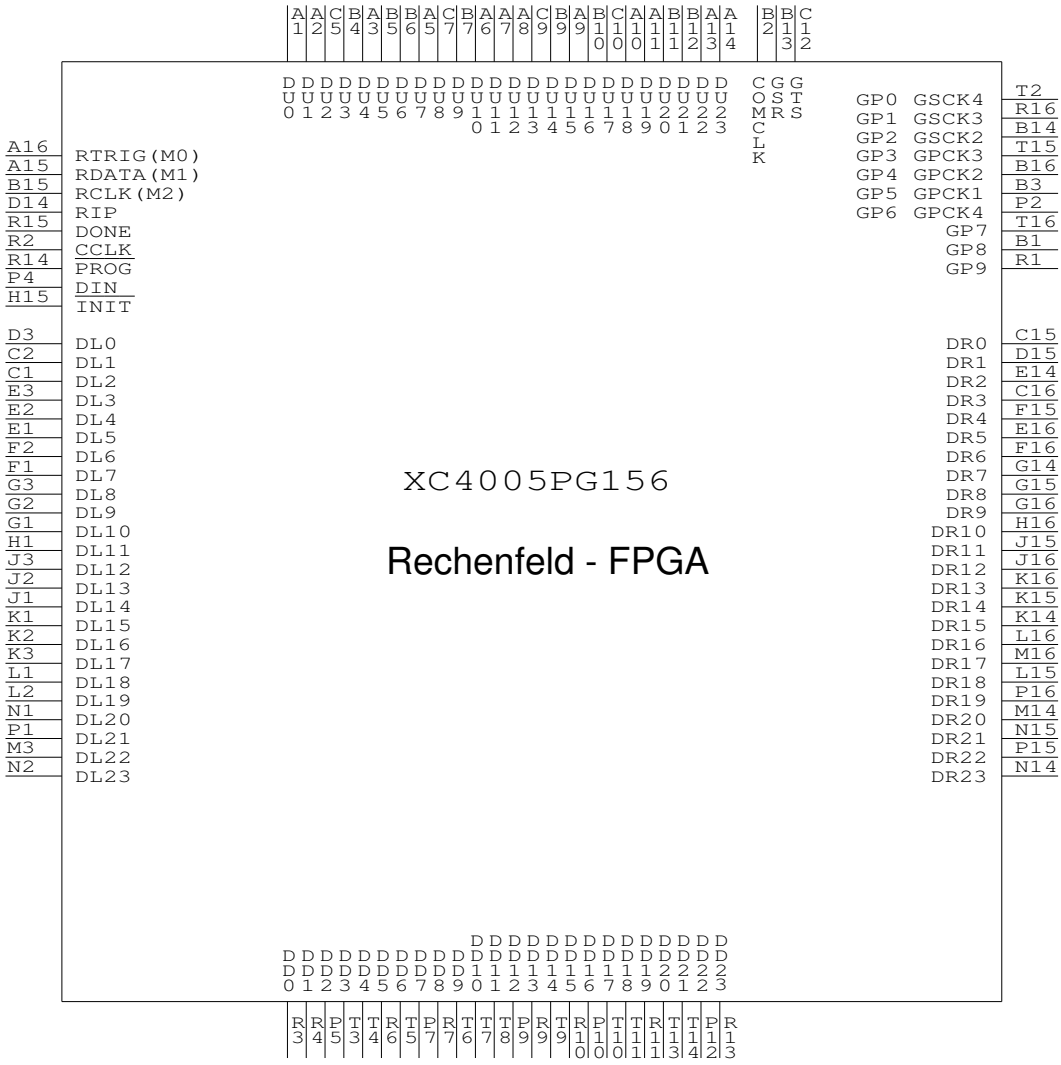

Bild 30 Pinbelegung eines Rechenfeld-FPGAs

Rechenfeld-Platine			
40poliger Flachbandkabelanschluß			
CN1 Interface-Adapter-Anschluß			
1	GND	$\overline{\text{BOE}}$ (NC)	2
3	GND	ComCLK (1)	4
5	GND	ComCLK (2)	6
7	GND	$\overline{\text{RAMWE}}$ (NC)	8
9	GTS	GSR	10
11	GND	GP0	12
13	GP1	GP2	14
15	GP3	GP4	16
17	GP5	GP6	18
19	GP7	GP8	20
21	GP9	GND	22
23	PR0 ($\overline{\text{INIT}}$)	PR1 (DIN)	24
25	PR2 ($\overline{\text{PROG}}$)	PR3 (CCLK)	26
27	PR4 (RIP)	PR5 (RCLK)	28
29	PR6 (RDATA)	PR7 (RTRIG)	30
31	PR8 ($\overline{\text{DONE}}$)	GND	32
33	FADR0	FADR1	34
35	FADR2	FADR3	36
37	FADR4 (NC)	FADR5 (NC)	38
39	FADR6 (NC)	PADR7 (NC)	40

Tabelle 4 Pinbelegung Rechenfeld CN1

Rechenfeld-Platine			
VG 64 Leiste männlich (s. Bemerkung im Text)			
CN 2 - CN 7 Ramcontrolleranschluß			
A1	GND	GND	C1
A2	DRF1	DRF0	C2
A3	DRF3	DRF2	C3
A4	DRF5	DRF4	C4
A5	DRF7	DRF6	C5
A6	GND	GND	C6
A7	DRF9	DRF8	C7
A8	DRF11	DRF10	C8
A9	DRF13	DRF12	C9
A10	DRF15	DRF14	C10
A11	GND	GND	C11
A12	DRF17	DRF16	C12
A13	DRF19	DRF18	C13
A14	DRF21	DRF20	C14
A15	DRF23	DRF22	C15
A16	GND	GND	C16
A17	GP8	GP9	C17
A18	GP6	GP7	C18
A19	GP4	GP5	C19
A20	GP2	GP3	C20
A21	GP0	GP1	C21
A22	GND	GND	C22
A23	SELECT	GND	C23
A24	RTRIG (PR7)	$\overline{\text{DONE}}$ (PR8)	C24
A25	RCLK (PR5)	RDATA (PR6)	C25
A26	CCLK (PR3)	RIP (PR4)	C26
A27	DIN (PR1)	$\overline{\text{PROG}}$ (PR2)	C27
A28	GND	$\overline{\text{INIT}}$ (PR0)	C28
A29	GSR	GND	C29
A30	GND	GND	C30
A31	ComCLK	GTS	C31
A32	GND	GND	C32

Tabelle 5 Pinbelegung Rechenfeld CN2 – CN7

Anhang C Technische Angaben Ramcontroller

Auf der folgende Seite ist der Schaltplan des Ramcontrollers abgebildet, anschließend folgt die Belegung des FPGAs sowie die Steckerbelegungen auf der Platine.

Zu dem Stecker CN1 (Rechenfeldanschluß) gilt das gleiche wie beim Rechenfeld gesagte: Da es sich bei diesem Stecker und seinem Gegenstück auf der Rechenfeldseite um VG64 Verbinder mit 90° Anschlüssen handelt, ist die Nummerierung — obwohl die Stecker miteinander verbunden werden — spiegelverkehrt. Dies ist bereits in der hier aufgeführten Belegungstabelle berücksichtigt.

Bild 31 Schaltplan des Ramcontrollers

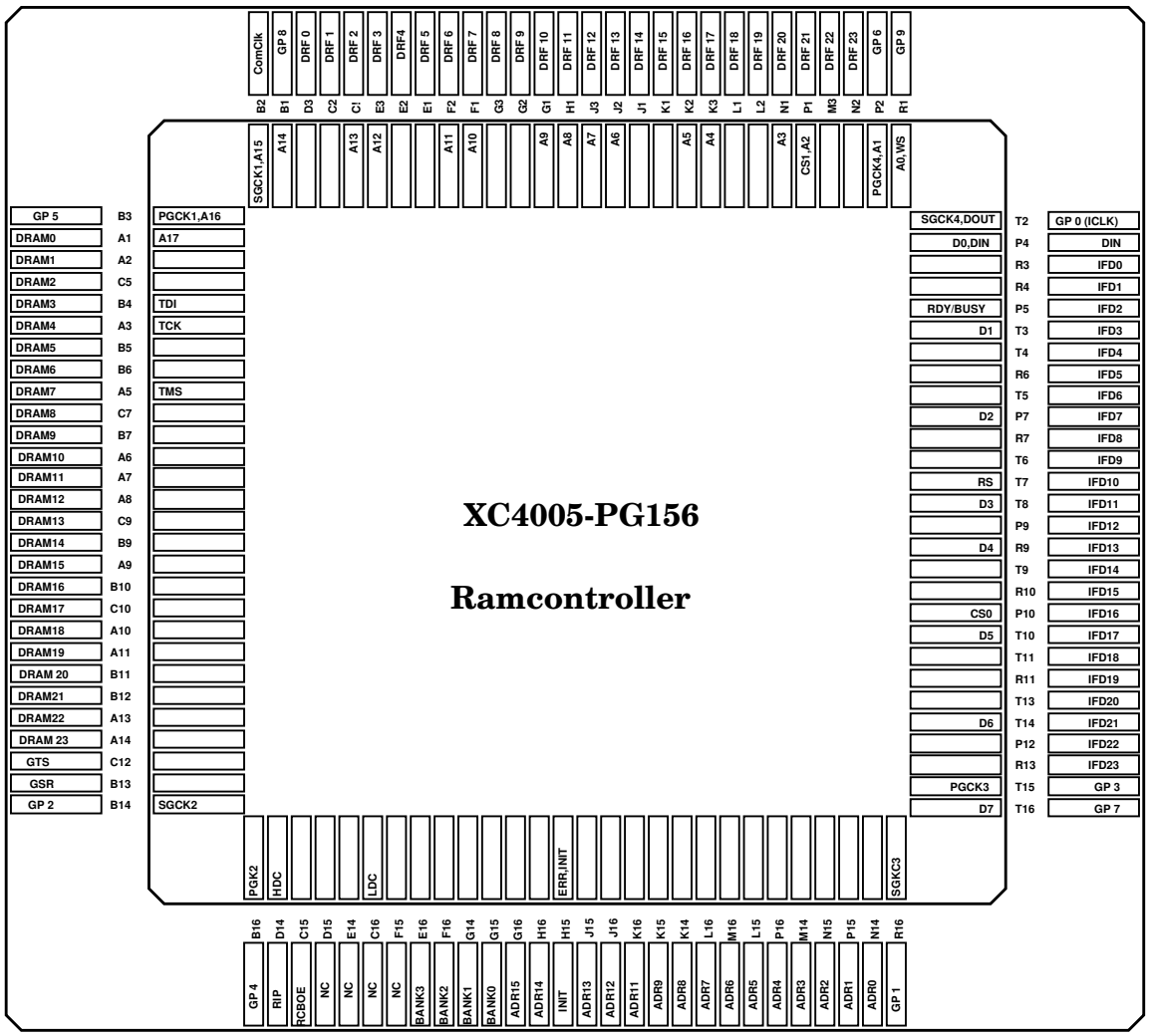

Bild 32 Pinbelegung eines Ramcontroller-FPGA (Sicht entsprechend dem XILINX-Tool XACT)

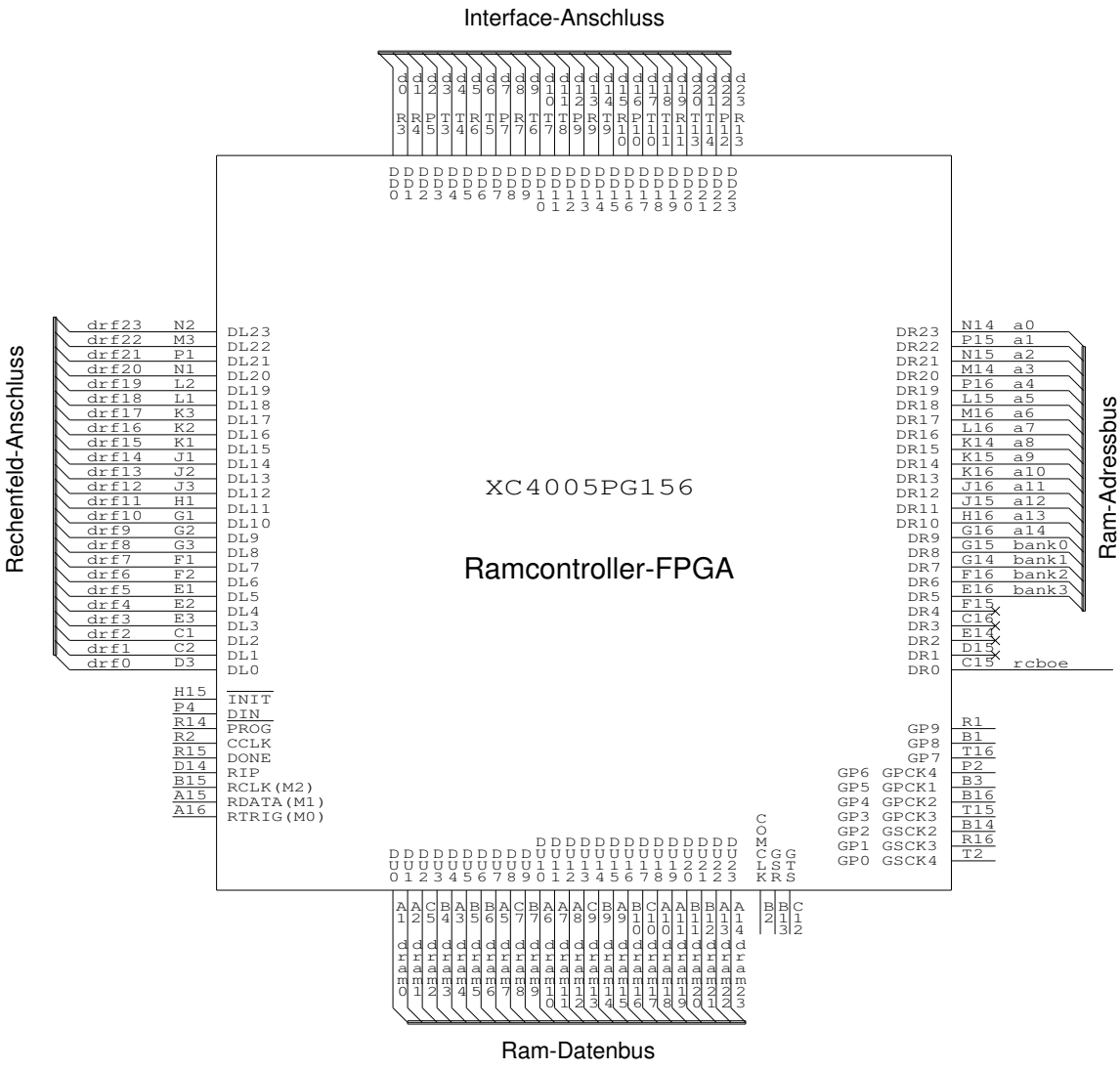

Bild 33 Pinbelegung eines Rechenfeld-FPGAs

Ramcontroller-Platine			
VG 64 Leiste weiblich (s. Bemerkung im Text)			
CN 1 Rechenfeldanschluß			
A1	GND	GND	C1
A2	ComCLK	GTS	C2
A3	GND	GND	C3
A4	GSR	GND	C4
A5	GND	$\overline{\text{INIT}}$ (PR0)	C5
A6	DIN (PR1)	$\overline{\text{PROG}}$ (PR2)	C6
A7	CCLK (PR3)	RIP (PR4)	C7
A8	RCLK (PR5)	RDATA (PR6)	C8
A9	RTRIG (PR7)	$\overline{\text{DONE}}$ (PR8)	C9
A10	SELECT	GND	C10
A11	GND	GND	C11
A12	GP0	GP1	C12
A13	GP2	GP3	C13
A14	GP4	GP5	C14
A15	GP6	GP7	C15
A16	GP8	GP9	C16
A17	GND	GND	C17
A18	DRF23	DRF22	C18
A19	DRF21	DRF20	C19
A20	DRF19	DRF18	C20
A21	DRF17	DRF16	C21
A22	GND	GND	C22
A23	DRF15	DRF14	C23
A24	DRF13	DRF12	C24
A25	DRF11	DRF10	C25
A26	DRF9	DRF8	C26
A27	GND	GND	C27
A28	DRF7	DRF6	C28
A29	DRF5	DRF4	C29
A30	DRF3	DRF2	C30
A31	DRF1	DRF0	C31
A32	GND	GND	C32

Tabelle 6 Pinbelegung Ramcontroller CN1

Ramcontroller-Platine			
60 poliger Flachbandkabelanschluß			
CN2 Interface-Adapter-Anschluß			
1	NC (VCC f. Terminierung)	GND	2
3	$\overline{\text{RAMWE}}$	GND	4
5	$\overline{\text{BOE}}$	GND	6
7	READ23	READ22	8
9	READ21	READ20	10
11	WRITE23	WRITE22	12
13	WRITE21	WRITE20	14
15	GND	READ19	16
17	READ18	READ17	18
19	READ16	WRITE19	20
21	WRITE18	WRITE17	22
23	WRITE16	GND	24
25	READ15	READ14	26
27	READ13	READ12	28
29	WRITE15	WRITE14	30
31	WRITE13	WRITE12	32
33	GND	READ11	34
35	READ10	READ9	36
37	READ8	WRITE11	38
39	WRITE10	WRITE9	40
41	WRITE8	GND	42
43	READ7	READ6	44
45	READ5	READ4	46
47	WRITE7	WRITE6	48
49	WRITE5	WRITE4	50
51	GND	READ3	52
53	READ2	READ1	54
55	READ0	WRITE3	56
57	WRITE2	WRITE1	58
59	WRITE0	GND	60

Tabelle 7 Pinbelegung Ramcontroller CN2

Ramcontroller-Platine			
50 poliger Platinenstecker			
CN3, CN4 Ramkarte			
1	VCC	GND	2
3	A0	$\overline{\text{RAMWE}}$	4
5	A2	A1	6
7	A3	A4	8
9	A5	A6	10
11	A7	A8	12
13	A9	A10	14
15	A11	A12	16
17	A13 (CE2)	A14 (NC)	18
19	BOE	BANK0	20
21	BANK1	BANK2	22
23	BANK3	GND	24
25	DRAM23	DRAM22	26
27	DRAM21	DRAM20	28
29	DRAM19	DRAM18	30
31	DRAM17	DRAM16	32
33	DRAM15	DRAM14	34
35	DRAM13	DRAM12	36
37	DRAM11	DRAM10	38
39	DRAM9	DRAM8	40
41	DRAM7	DRAM6	42
43	DRAM5	DRAM4	44
45	DRAM3	DRAM2	46
47	DRAM1	DRAM0	48
49	GND	VCC	50

Tabelle 8 Pinbelegung Ramcontroller CN3, CN4

Anhang D Technische Angaben Ramcard I

Die Ramcard kann entweder drei RAMs vom Typ 7B185 (8k x 8) oder drei RAMs des Typs 7C199 (32k x 8) oder pincompatible Typen aufnehmen. Die Zugriffszeit sollte 12ns sein. Die Pinbelegung der Sockel auf der Ramcard sind in Bild 34 aufgeführt. Bei einem Einbau von 32k x 8 RAMs muß der Adreßmultiplexer in den Ramcontroller-FPGAs natürlich an den erweiterten Adreßraum anpaßt werden.

7C199	7B185	Pinbelegung RAMCARD				7B185	7C199
A5	NC	A14	1	28	VCC		
A6		A4	2	27	$\overline{WE}$		
A7		A5	3	26	A13	CE2	A4
A8		A6	4	25	A3		
A9		A7	5	24	A2		
A10		A8	6	23	A1		
A11		A9	7	22	$\overline{OE}$		
A12		A10	8	21	A0		
A13		A11	9	20	$\overline{CE}$		
A14		A12	10	19	D7		
		D0	11	18	D6		
		D1	12	17	D5		
		D2	13	16	D4		
		GND	14	15	D3		

Bild 34 Pinbelegungen der DIL-Sockel der Ramcard

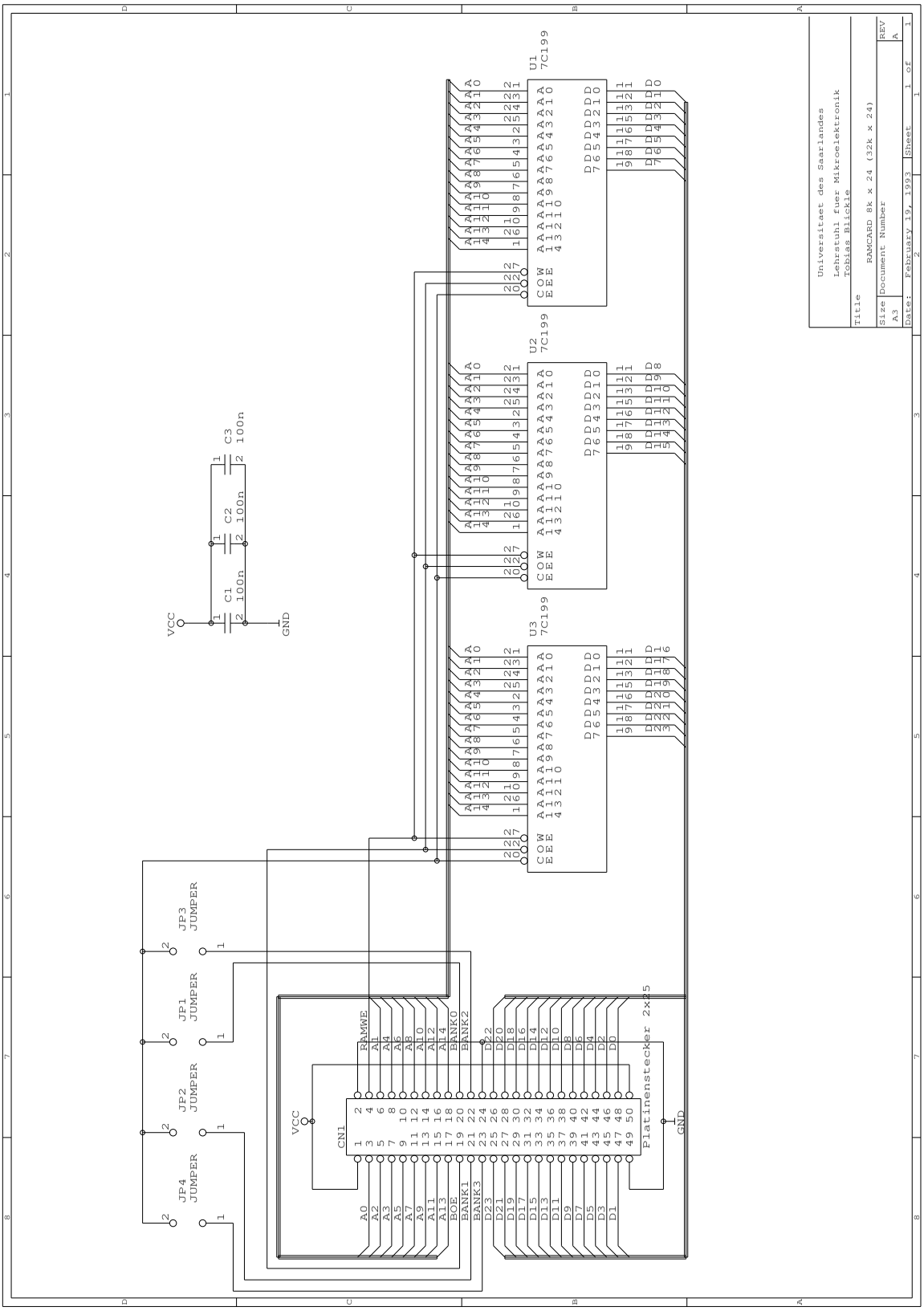

Bild 35 Schaltbild der Ramcard I

Anhang E Technische Angaben Program&Readback Buffer

Der Program&Readback-Bus (PRBUS) darf immer nur an höchstens ein FPGA gleichzeig angeschlossen sein. Deshalb besitzt jedes FPGA einen Program&Readback Treiber, der die entsprechenden Anschlüsse des FPGAs erst dann zum PRBUS durchschaltet, wenn ein Adressierungssignal ($\overline{\text{SELECT}}$) dies erlaubt.

Im nicht selektierten Zustand sorgt ein Widerstandsnetzwerk für den richtigen logischen Pegel an den Eingängen des FPGAs.

Den Schaltplan zeigt Bild 36 . Das DONE-Signal wird nicht über Treiber geschaltet, sondern führt direkt zu allen FPGAs, da dadurch ein synchrones Umschalten aller FPGAs vom Programmiermodus in den Ablaufmodus gewährleistet wird.

Zu Meß- und Testzwecken sind einige Signale auf den Meßstecker JM geführt (genauers siehe Meßsteckerbelegung in Anhang G).

Der Plan enthält keine feste Nummerierung der Bauelement, da sie unterschliedlich für Ramcontroller und Rechenfeld ausfällt, bzw. das Rechenfeld diese Schaltung neun mal enthält.

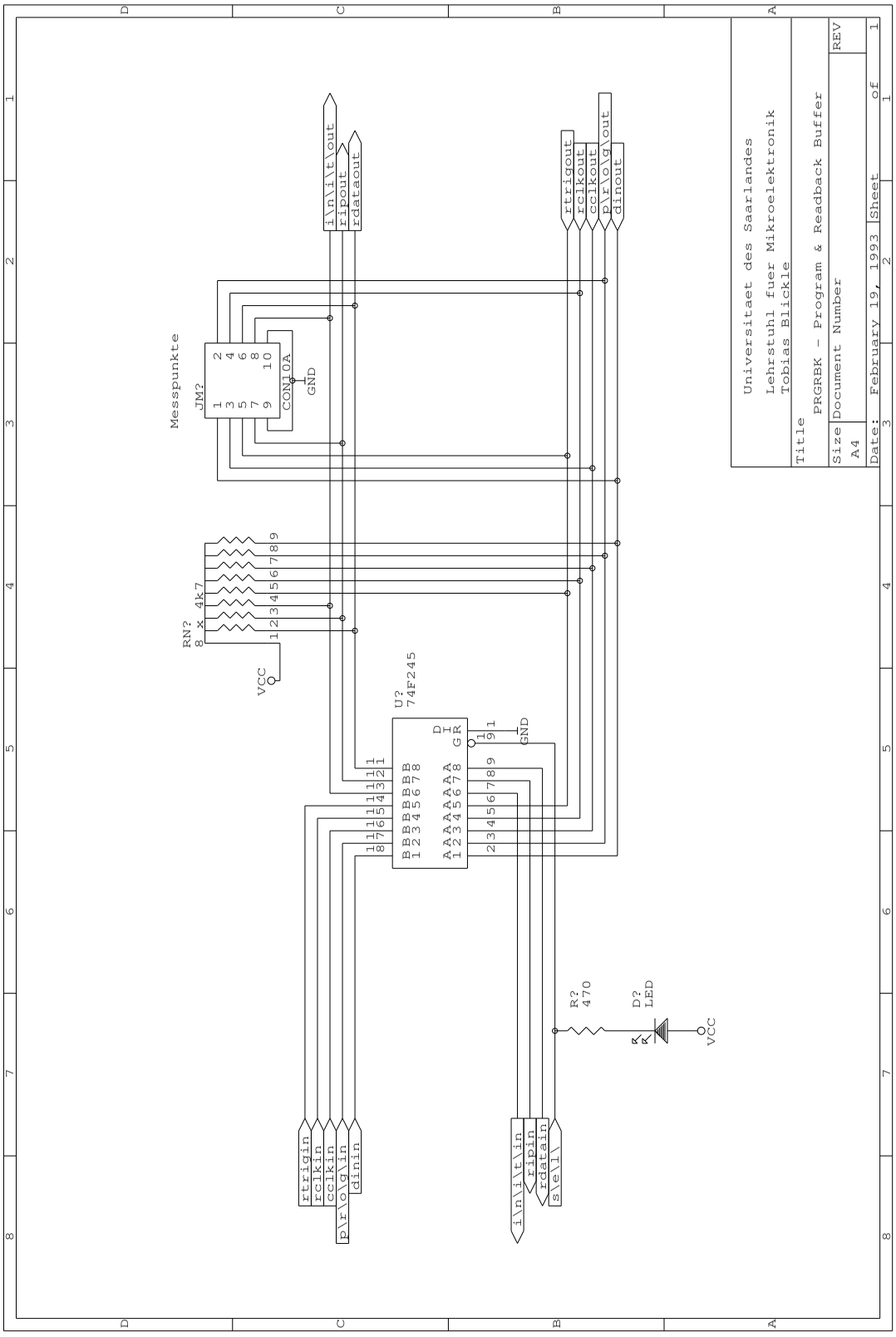

Bild 36 Schaltbild des Program&Readback Buffer

Anhang F Technische Angaben Interface-Adapter

Der Interface-Adapter in Fädel-Technik realisiert. Das Layout der Platine zeigt Bild 38. Die Belegung des Jumper-Blocks für die Signalzuordung im GPBUS ist in Bild 37 zu sehen. Die Schaltpläne des Interface-Adapters sind in Bild 39, 40, 41 und 42 gezeigt.

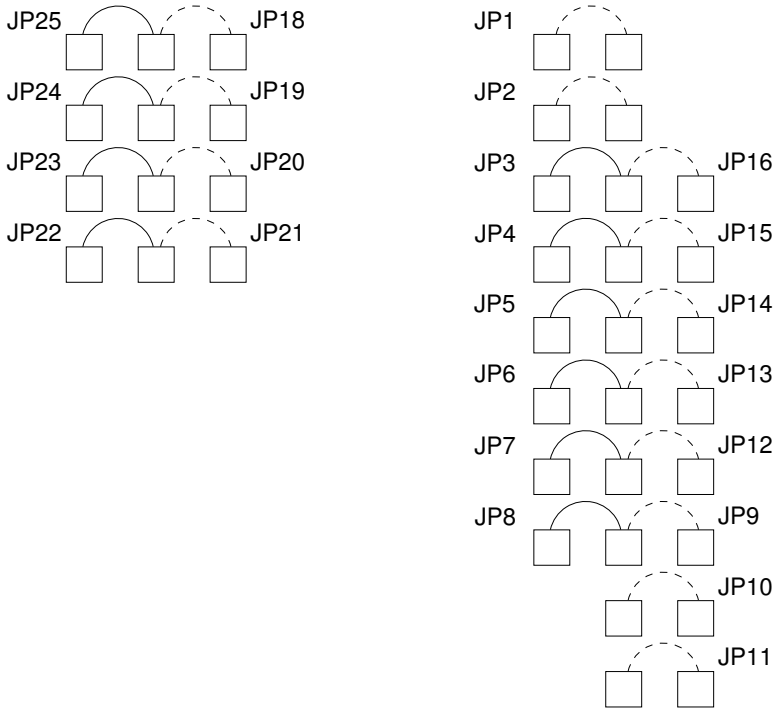

Bild 37 Lage der GPBUS Jumper im Jumperfeld

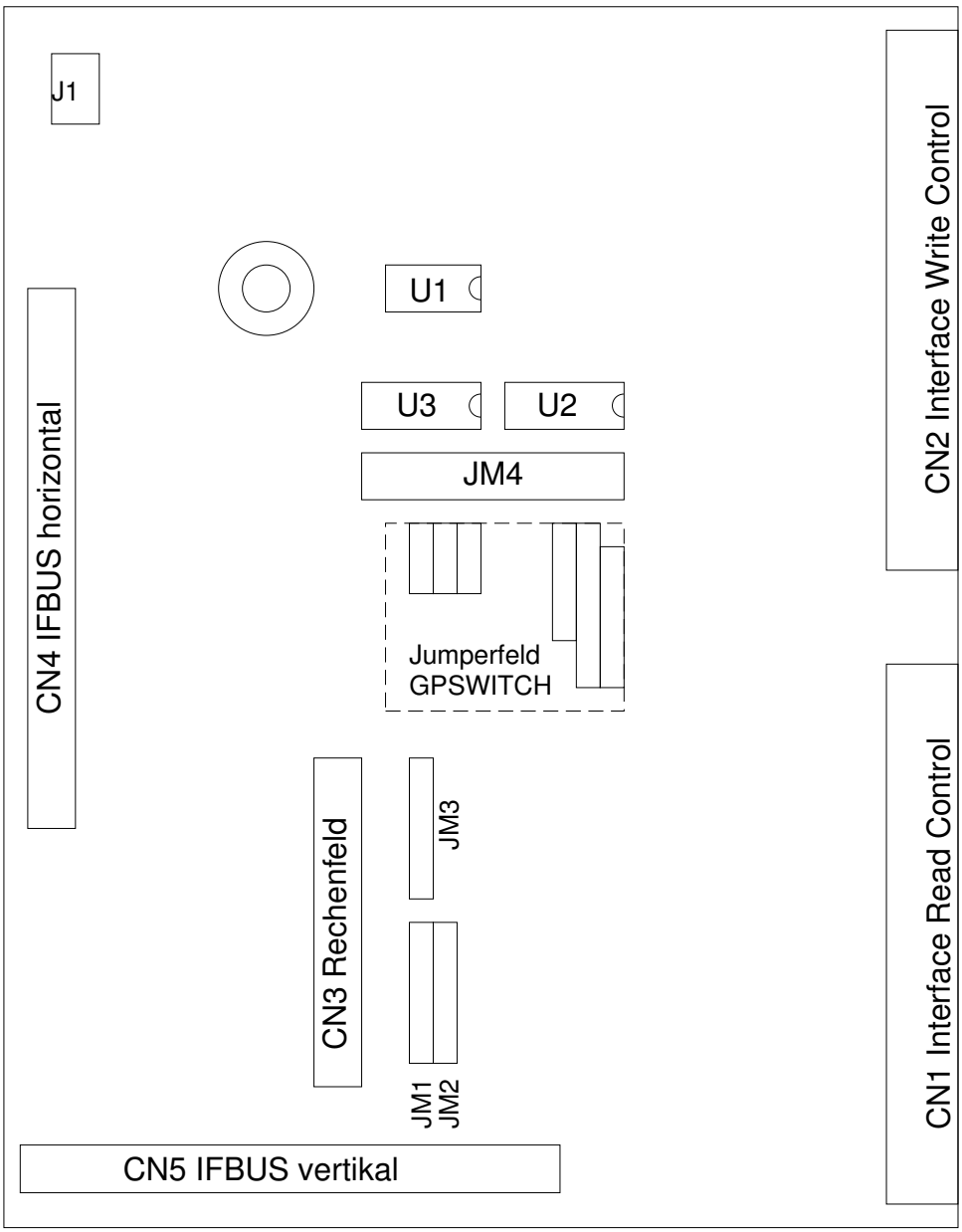

Bild 38 Layout der Interface-Adapter-Platine

Bild 39 Schaltbild des Interface-Adapters

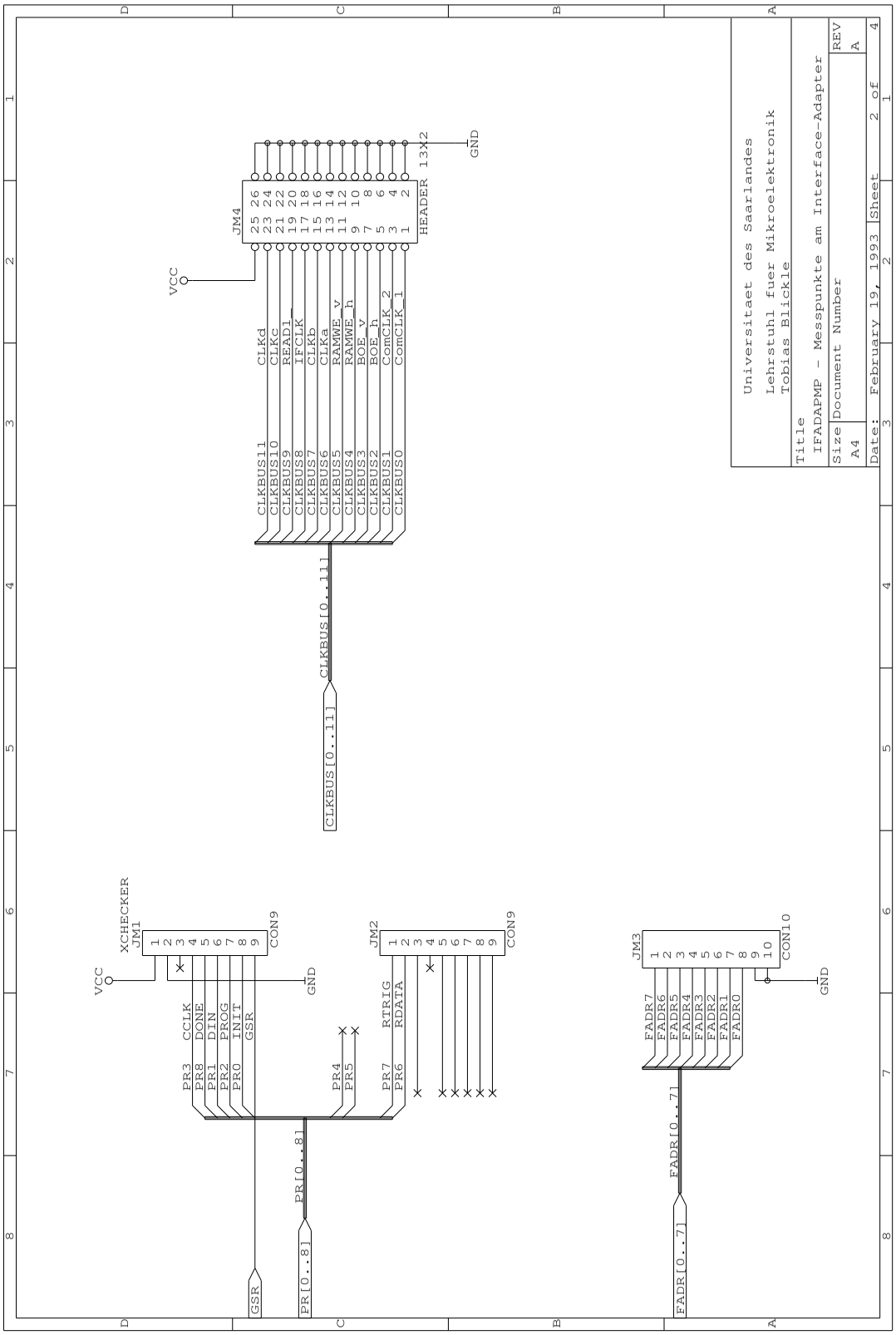

Bild 40 Meßpunkte des Interface-Adapters

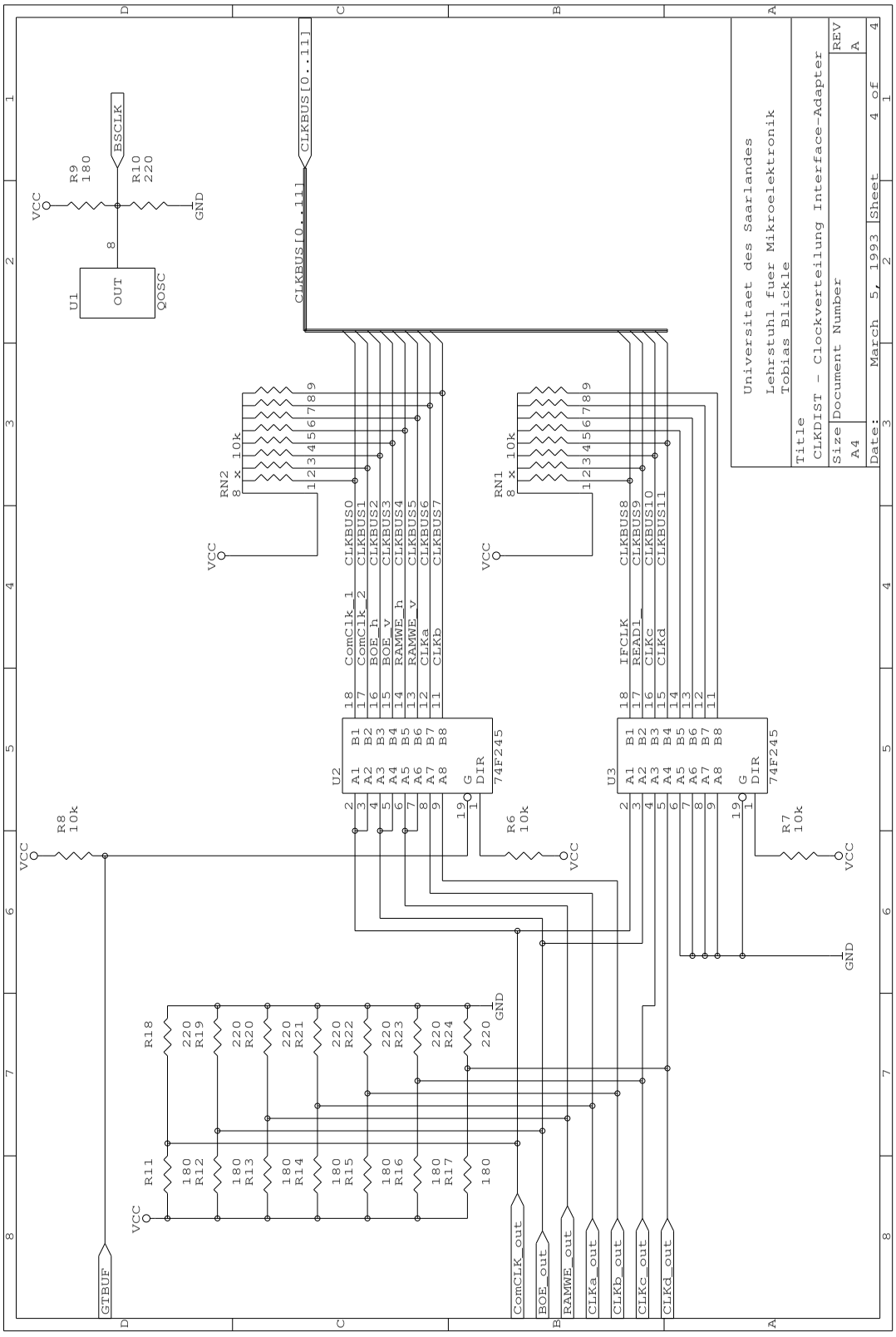

Bild 41 Teilschaltbild Taktsignalverteilung

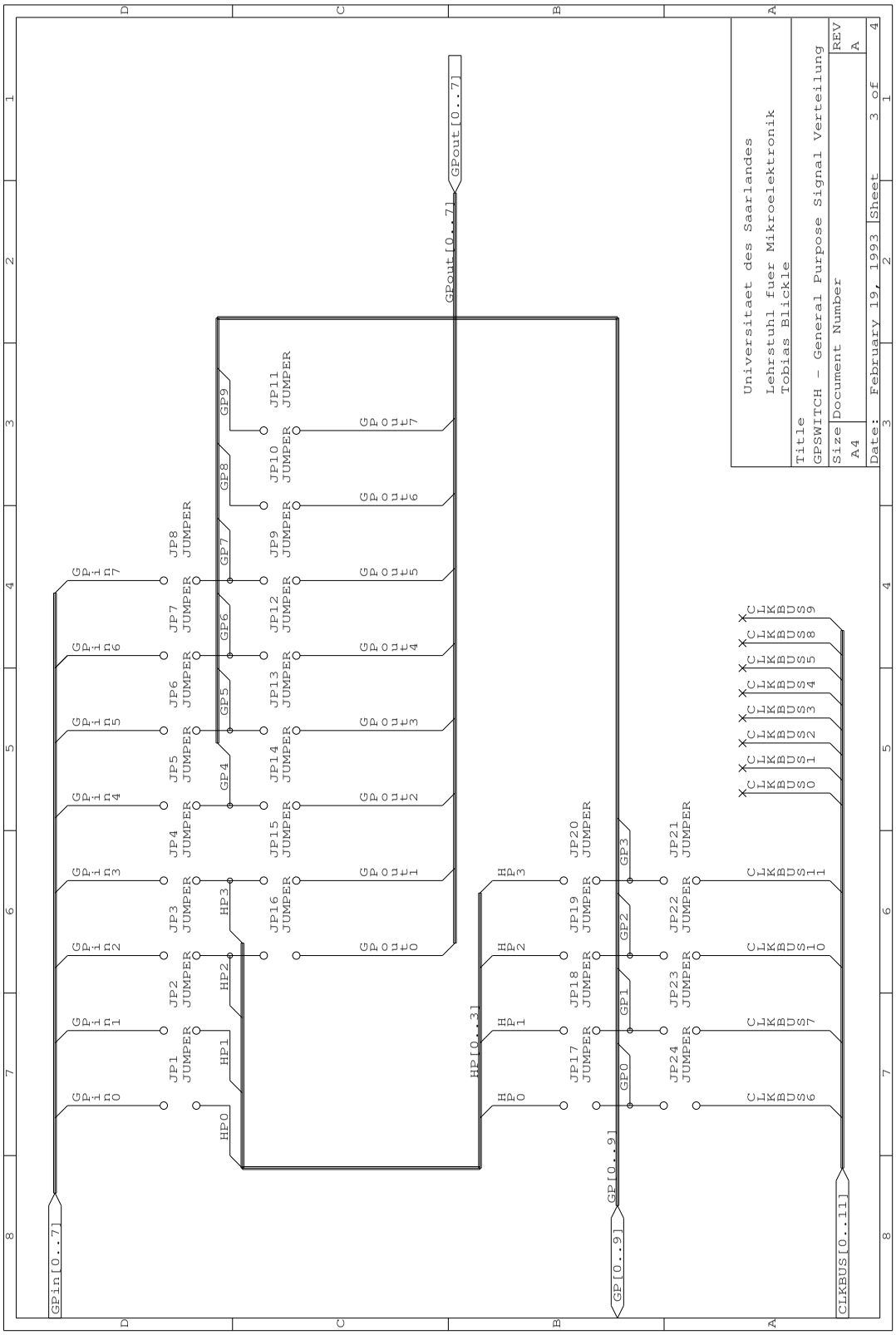

Bild 42 Schaltbild der General-Purpose-Bus-Verteilung

Interface-Adapter-Platine			
VG 64 Leiste männlich			
CN 1 Interfaceanschluß			
A1	GND	GND	C1
A2	IFIF9	IFIF8	C2
A3	IFIF7	IFIF6	C3
A4	IFIF5	IFIF4	C4
A5	IFIF3	FADR7	C5
A6	FADR6	FADR5	C6
A7	FADR4	FADR3	C7
A8	FADR2	FADR1	C8
A9	FADR0	READ1_ (IF Signal)	C9
A10	GPin7	GPin6	C10
A11	GPin5	GPin4	C11
A12	GPin3	GPin2	C12
A13	GPin1	GPin0	C13
A14	READ0	READ1	C14
A15	READ2	READ3	C15
A16	READ4	READ5	C16
A17	READ6	READ7	C17
A18	READ8	READ9	C18
A19	READ10	READ11	C19
A20	READ12	READ13	C20
A21	READ14	READ15	C21
A22	READ16	READ17	C22
A23	READ18	READ19	C23
A24	READ20	READ21	C24
A25	READ22	READ23	C25
A26	IFIF2	IFIF1	C26
A27	ComCLK(_1)	IFIF0	C27
A28	IFCLK	(RF_)GTS	C28
A29	$\overline{\text{DONE}}$ (PR8)	(RF_)GSR	C29
A30	$\overline{\text{PROG}}$ (PR2)	CCLK (PR3)	C30
A31	$\overline{\text{INIT}}$ (PR0)	DIN (PR1)	C31
A32	GND	GND	C32

Tabelle 9 Pinbelegung Interface-Adapter CN1

Interface-Adapter-Platine			
VG 64 Leiste männlich			
CN 2 Interfaceanschluß			
A1	GND	GND	C1
A2	IFIF9	IFIF8	C2
A3	IFIF7	IFIF6	C3
A4	IFIF5	IFIF4	C4
A5	IFIF3	IFIF2	C5
A6	IFIF1	IFIF0	C6
A7	N.C.	N.C.	C7
A8	RTRIG (PR7)	RDATA (PR6)	C8
A9	RCLK (PR5)	RIP (PR4)	C9
A10	GPout7	GPout6	C10
A11	GPout5	GPout4	C11
A12	GPout3	GPout2	C12
A13	GPout1	GPout0	C13
A14	WRITE0	WRITE1	C14
A15	WRITE2	WRITE3	C15
A16	WRITE4	WRITE5	C16
A17	WRITE6	WRITE7	C17
A18	WRITE8	WRITE9	C18
A19	WRITE10	WRITE11	C19
A20	WRITE12	WRITE13	C20
A21	WRITE14	WRITE15	C21
A22	WRITE16	WRITE17	C22
A23	WRITE18	WRITE19	C23
A24	WRITE20	WRITE21	C24
A25	WRITE22	WRITE23	C25
A26	N.C	CLKd_out	C26
A27	ComCLK(_1)	CLKc_out	C27
A28	IFCLK	CLKb_out	C28
A29	BSCLK	CLKa_out	C29
A30	RAMWE_out	BOE_out	C30
A31	ComCLK_out	GTBUF	C31
A32	GND	GND	C32

Tabelle 10 Pinbelegung Interface-Adapter CN2

Interface-Adapter-Platine
40pol Flachbandkabelanschluß
CN3 Rechenfeld-Anschluß
Belegung wie CN1 Rechenfeld (Tabelle 4)

Tabelle 11 Pinbelegung Interface-Adapter CN3

Interface-Adapter-Platine
60pol Flachbandkabelanschluß
CN4 Ramcontroller-Anschluß (IFBUS horizontal)
Belegung wie CN2 Ramcontroller (Tabelle 7)

Tabelle 12 Pinbelegung Interface-Adapter CN4

Interface-Adapter-Platine
60pol Flachbandkabelanschluß
CN5 Ramcontroller-Anschluß (IFBUS vertikal)
Belegung wie CN2 Ramcontroller (Tabelle 7)

Tabelle 13 Pinbelegung Interface-Adapter CN5

Anhang G Meßstecker

Meßstecker Typ A	
Program&Readback Signale	
1	DIN
2	$\overline{\text{PROG}}$
3	CCLK
4	RCLK
5	RTRIG
6	RDATA
7	RIP
8	$\overline{\text{INIT}}$
9	GND
10	GND

Tabelle 14 Pinbelegung Meßstecker Typ A

Meßstecker Typ B			
Datenbus			
1	DATA0	DATA1	2
3	DATA2	DATA3	4
5	DATA4	DATA5	6
7	DATA6	DATA7	8
9	DATA8	DATA9	10
11	DATA10	DATA11	12
13	DATA12	DATA13	14
15	DATA14	DATA15	16
17	DATA16	DATA17	18
19	DATA18	DATA19	20
21	DATA20	DATA21	22
23	DATA22	DATA23	24
25	GND	GND	26

Tabelle 15 Pinbelegung Meßstecker Typ B