Entwurf und Realisierung eines vierfach MIMD Prozessor Mikrochips für eine Anwendung in der Hochenergiephysik

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.- Ing. Falk Lesser
aus Göttingen

Mannheim, 2002

Dekan:	Professor Dr. Herbert Popp, Universität Mannheim
Referent:	Professor Dr. Volker Lindenstruth, Universität Heidelberg
Korreferent:	Professor Dr. Ulrich Brüning, Universität Mannheim

Tag der mündlichen Prüfung: 28. August 2002

Zusammenfassung

Inhalt dieser Dissertation ist der konzeptionelle Entwurf und die Realisierung eines Multiprozessorsystems. Primäres Einsatzgebiet ist das Datenverarbeitungssystem des Transition Radiation Detektors des ALICE[1] Experiments am CERN[2]. Konformität zu den Anforderungen der Datenverarbeitungskette unter Berücksichtigung fortschrittlicher Architekturkonzepte bei minimaler Chipfläche stellt dabei das grundsätzliche Entwurfsparadigma dar. Die Funktionalität des MIMD[3] Prozessors wird in einer Testumgebung auf mehreren Entwurfshierarchien nachgewiesen. Die Abbildung auf eine 0.18 µm CMOS Technologie liefert die notwendigen Ergebnisse zur Beurteilung der Leistungsfähigkeit des Entwurfs. Der MIMD Prozessor wurde in einen Prototypen (TRAP 1) integriert und innerhalb eines MPW[4]-Runs erfolgreich submittiert.

Abstract

The following thesis deals with the conceptual design and the realization of a multiprocessor system. Primary application is the readout system of the Transition Radiation Detector of the ALICE Experiment at CERN. As such, the basic design paradigm lies in obtaining specification conformity while focusing on progressive architecture concepts using a minimum of chip space. Furthermore, the functional performance of the MIMD processor core is shown in a test environment on several hierarchy levels. The mapping on a 0.18 µm CMOS technology delivers the results necessary in evaluating both performance and size of the MIMD processor. The MIMD processor was integrated in a prototype named Trap 1 and was succesfully submited in an MPW run.

---

1. **A** **L**agre **I**on **C**ollider **E**xperiment
2. **C**onseil **E**uropéenne pour la **R**echerche **N**ucléaire
3. **M**ultiple **I**nstruction **M**ultiple **D**ata
4. **M**ulti **P**roject **W**afer

Inhaltsverzeichnis

1 *Einleitung*

Multiprozessorsysteme erfahren gegenwärtig eine immer größere Akzeptanz, da die sequentielle Verarbeitung von Einprozessorsystemen nur maximal mit der Frequenz skaliert [12]. Tatsächlich kann durch geeignete Parallelisierung auf verschiedenen Entwurfsebenen eine Geschwindigkeitssteigerung erzielt werden. Bekannte Ebenen der Parallelisierung sind die Programm-, Prozeß-, Block- und Anweisungsebene [48], [27]. Angewandte Techniken hierfür sind die Rechner- bzw. Prozessorkopplung oder Techniken innerhalb der Prozessorarchitektur. Für die Prozessorkopplung werden meist speicher- oder nachrichtengekoppelte Systeme verwendet. Auf Prozessorebene sind Techniken, wie Befehlspipelining, superskalare und VLIW[1] Architekturen verbreitete Implementierungsvarianten. Eine Kombination dieser Eigenschaften bietet für spezielle Anwendungen oft den besten Mix.

Im Rahmen dieser Dissertation wurde ein MIMD Prozessor für die besonderen Bedingungen der Datenverarbeitungskette des ALICE Experiments [1], [2] entwickelt. Ferner wird ein Preprozessor vorgestellt, der digitalisierte Eingangsdaten mit einem festen Algorithmus vorverarbeitet und dem MIMD Prozessor die Resultate übergibt. Der Entwurf hat das Ziel, die Daten von insgesamt 21 analogen Datenkanälen zu verarbeiten und die Ergebnisse den weiteren Einheiten dieser Kette zur Verfügung zu stellen. Der Entwurf des MIMD Prozessors ist dabei auf die Bedürfnisse der Datenverarbeitungskette optimiert, kann durch seine generische Architektur aber in eine Vielzahl von Anwendungen integriert werden. Die verschiedenen Schnittstellen bieten zudem genügend Raum für Erweiterungen. Oberstes Entwurfsparadigma ist eine möglichst kompakte Architektur bei gleichzeitigem Höchstmaß an Flexibilität.

Die nachfolgende Abhandlung beginnt mit der Darstellung verschiedener Aspekte paralleler Prozessorarchitekturen. Anschließend wird in Kapitel 3 das gewählte Architekturkonzept vorgestellt, dem in Kapitel 4 die umfassende Beschreibung der im Vergleich entworfenen Prozessorarchitektur folgt. In Kapitel 5 wird die primäre Anwendung des konstruierten MIMD Prozessors beschrieben, der die in Kapitel 6 dargestellte Architektur des Preprozessors bestimmt. Eine Entwicklungsumgebung für den MIMD Prozessor wird in Kapitel 7 erläutert. In Kapitel 8 wird die zur Verifikation verwendete Testumgebung und die genutzten Testverfahren vorgestellt. Der technologieunabhängige Entwurf wurde in einem 0,18 µm CMOS Prozess mit sechs Lagen Metall submittiert. Die Ergebnisse der Hardwaresynthese und des Layouts werden in Kapitel 9 aufgezeigt und diskutiert. Eine Zusammenfassung schließt dieses Dokument ab. Zwei Prototypen, die im Verlauf dieser Arbeit entwickelt wurden, werden im Anhang vorgestellt.

1. **V**ery **L**arge **I**nstruction **W**ord

2 *Parallele Architekturen*

„Das Ziel der Entwicklung paralleler Rechnerarchitekturen ist die Überwindung der Einschränkung, die durch die sequentielle Verarbeitung nicht paralleler Architekturen vorgegeben sind“ [46]. Dieser Satz gibt die grundsätzliche Motivation für die Entwicklung paralleler Architekturen, sagt aber nichts zu den dadurch entstehenden Herausforderungen, die bei der Entwicklung vorhanden sind. Das sind hauptsächlich die Programmierbarkeit, die Speicherlatenz und die Synchronisation.

Dieses Kapitel beschreibt einige Grundlagen, die für die Entwicklung einer parallelen Prozessorsarchitektur notwendig sind und zeigt einen Ausschnitt zu den vorhandenen Konzepten. Einleitend werden zwei übliche Klassifikationssysteme vorgestellt, um im zweiten Teil das Konzept der speichergekoppelten Multiprozessorsysteme vorzustellen. Sie bilden die Grundlage für das entwickelte Prozessormodell.

2.1 Klassifikationen

Bei der Einteilung von Parallelrechnerarchitekturen wird nach verschiedenen Prinzipien unterschieden. Flynn [16] führt eine zweidimensionale Klassifikation nach dem Operationsprinzip ein. Demnach stellen Rechner Operatoren auf dem Instruktionsstrom und dem Datenstrom dar, woraus die folgenden Klassen resultieren:

	einfacher Instruktionsstrom	mehrfacher Instruktionsstrom
einfacher Datenstrom	SISD von Neumann	MISD
mehrfacher Datenstrom	SIMD Vektorprozessoren	MIMD Mehrprozessorrechner

Tab. 1: Klassifikation nach Flynn

Das Von-Neumann Rechnermodell (SISD) gilt als das grundlegende Rechnermodell und ist besonders wegen des Operationsprinzips bekannt und unterteilt einen Rechner in eine Zentraleinheit, einen Speicher und eine Eingabe/Ausgabeeinheit. Das Operationsprinzip teilt die Zentraleinheit in ein Rechen- und ein Steuerwerk, woraus wiederum die sequentielle Abarbeitung von Instruktionen resultiert. Für Parallelrechner finden sich diese Art von Prozessor als Verarbeitungseinheiten wieder.

Das SIMD-Architekturkonzept besitzt einen Informationsstrom, mit dem mehrere Datenelemente einer Datenstruktur verarbeitet werden. Dabei sind die Daten in Vektoren organisiert, so dass die Parallelität der Vektororganisation zur Verarbeitung ausgenutzt werden kann. Werden die Elemente der Reihe nach in einer Pipeline verarbeitet, wird dieser Rechner als Vektorrechner bezeichnet. Sind hingegen mehrere Verarbeitungseinheiten implementiert, so dass die Elemente des Vektors parallel verarbeitet werden können, wird von einem Feldrechner gesprochen. Vektorrechner besitzen sowohl eine Einheit zur Verarbeitung von Skalaren, als auch eine Einheit zur Verarbeitung von Vektoren [41]. Der Vorteil der Architektur liegt in dem nach außen sichtbaren Programmiermodell. Im Allgemeinen wird der Rechner sequentiell programmiert, nur die in den Vektoroperationen enthaltene Parallelität wird durch die einfache Hardwarestruktur der Pipeline effizient ausgenutzt. Die Verarbeitung des Vektors erfolgt nach dem Pipelineverfahren, wobei jede Instruktion in möglichst gleichlange Teiloperationen zerlegt wird. Die Beschleunigung liegt in der Tiefe der Pipeline. Dabei ist bei diesem Prozessortyp eine tiefe Pipeline durchaus zulässig, da jedes Ergebnis der Vektoroperation unabhängig ist. Die Vektoreinheit enthält Pipelines, welche die Operation ausführen. Als Zwischenspeicher werden Register verwendet, die in großer Anzahl vorhanden sind oder es erfolgt die Verarbeitung aus dem Hauptspeicher. Bei beiden Architekturen wird der Hauptspeicher meistens mehrfach verschränkt, um eine hohe Speicherbandbreite zu gewährleisten.

Bei der Konzeption eines Feldrechners [41] wird die Nebenläufigkeit von parallelen Verarbeitungseinheiten ausgenutzt. Die ausführenden Einheiten besitzen dabei keine Hardware, um den Programmfluss zu beeinflussen. Statt dessen wird in Feldrechnern die Möglichkeit eingebaut, durch Maskierung auf den Programmfluss Einfluss zu nehmen. Eine besondere Form ist das systolische Array, das meistens eine zwei oder dreidimensionale Gitteranordnung von Gitterelementen besitzt. Die Verarbeitung erfolgt im Pipelineverfahren taktsynchron.

Beim MIMD-Architekturkonzept arbeitet ein Satz von Prozessoren gleichzeitig und verarbeitet verschiedene Datensätze mit verschiedenen Instruktionssequenzen. Dabei wird häufig eine weitere Einteilung nach physikalischem Speicher, Adressraum, Kommunikationsmodell, Programmiermodell, Synchronisationsmechanismus und Latenzbehandlung getroffen, wobei der Art der Speicherkopplung die größte Bedeutung zukommt.

Ein weiteres Klassifikationssystem ist das Erlanger Klassifikationssystem (ECS) [7], das drei verschiedene Ebenen unterscheidet, die ihrerseits noch in Nebenläufigkeit und Pipelining unterteilt werden. Die resultierende Tripelnotation wird durch Operatoren, die zusammengesetzte Strukturen beschreiben, modifiziert. Diese Notation ergibt ein Maß für die Leistungsfähigkeit (max. Parallelität) als auch für die Flexibilität (Anzahl der verschiedenen Arbeitsmodi). Die vier wesentlichen Punkte dabei sind:

Serialität: liegt vor, wenn die auf einem bestimmten Abstraktionsniveau definierten Aktionen nicht gleichzeitig ausführbar sind, d.h. es ist genau eine Aktion ausführbar.

Parallelität: liegt vor, wenn die auf einem bestimmten Abstraktionsniveau definierten Aktionen gleichzeitig ausführbar sind.

Nebenläufigkeit: Die Parallelität wird als Nebenläufigkeit beschrieben, wenn die Ressourcen des Systems das gleichzeitige Ausführen vollständiger, auf diesem Niveau definierter Aktionen erlaubt.

Pipelining: liegt vor, wenn die auf einem bestimmten Abstraktionsniveau definierte Aktion in k Teilaktionen unterteilt werden kann, die nacheinander abgearbeitet werden.

Der Aufbau eines Rechners bzw. Prozessors gliedert sich dabei in die drei Hardwareelemente Leitwerk, Steuerwerk und Elementare Stelle, wobei das letztgenannte eine Einheit im Rechenwerk darstellt, welche die Operation auf genau eine Bitstelle im Datenwort ausführt. Insgesamt besteht jeder Rechner gemäß ECS aus einer bestimmten Anzahl von k Leitwerken, die ihrerseits eine bestimmte Zahl d von Rechenwerken mit einer bestimmten Zahl w von elementaren Stellen umfasst. Die Parameter k, d, w definieren den Umfang der Nebenläufigkeit des Rechners auf der jeweiligen Betrachtungsebene. Liegt Pipelineverarbeitung vor, können k' Teilleitwerke, d' Teilrechenwerke und w' elementare Teilwerke zusammenarbeiten. Daraus resultiert die Typbeschreibung t = (k*k', d*d', w*w'). Als Beispiel [48] wird der Rechner Intel Paragon XP/S verwendet. Dieser besitzt 1840 Prozessorknoten (k = 1840) von denen jeder aus zwei Prozessoren des Typs i860 besteht (k'=2). Der eine wird als Arbeits- der andere als Kommunikationsprozessor verwendet (Makropipeline). Jeder Prozessor kann im „Dual Instruction Mode" beschrieben werden, was einer zweistufigen Instruktionspipeline entspricht (d' = 2). Die ALU des i860 besitzt einen 64 Bit breiten Datenpfad (w = 64), die in drei Pipelinestufen aufgeteilt ist (w' = 3). Daraus ergibt sich eine Klassifikation gemäß ECS von t = (1840 * 2, 1 * 2, 64 * 3).

2.2 Speichergekoppelte Multiprozessoren

Speichergekoppelte Multiprozessoren bestehen aus mehreren (mindestens zwei) gleichen oder verschiedenen Recheneinheiten, die durch ein Netzwerk oder einen gemeinsamen Speicher miteinander verbunden sind. Im Gegensatz zu Feldrechnern können Multiprozessorsysteme pro Zeiteinheit unterschiedliche Befehle pro Recheneinheit bearbeiten. Beispiele sind TERA, HEP, usw. [12], [45], [48].

2.2.1 Allgemeines

Bei speichergekoppelten Systemen sind die betrachteten Prozessoren über gemeinsame Speicherbereiche miteinander verbunden. Die Kopplung kann über Busse, Crossbar oder Multiportspeicher geschehen. Speichergekoppelte Systeme verfügen in den meisten Fällen nur über einen gemeinsamen Speicher. Die Kommunikation wird dadurch realisiert, so dass der sendende Prozessor Daten in den gemeinsamen Speicher schreibt und der empfangende Prozessor die Daten liest [14].

Das größte Problem auf Software Ebene ist, eine einfache Programmierbarkeit zu gewährleisten. Die zu lösenden Probleme müssen unabhängig von der vorliegenden Hardwarekonfiguration zu lösen sein. „Als Stand der Technik beim Programmieren von MIMD Prozessoren gilt immer noch, dass der Programmierer die Parallelarbeit auf einer niedrigen Abstraktionsebene selbst organisieren muss, um ein effizientes, paralleles Programm zu erzielen" [45].

Auf Architekturebene stellt die Speicherlatenz und die Synchronisation der einzelnen CPU’s das größte Problem dar. Unter der Speicherlatenz versteht man die Zeit, die vergeht, bis ein benötigtes Datum von einem Speicher geliefert wird. Ist der Speicher nicht lokal, ist mit einer zusätzlichen Latenz durch das Kommunikationsnetzwerk zu rechnen. Die Zeit, die ein Prozessor auf einen Speicherzugriff warten muss, wird meistens durch Leerlauf des Prozesses (Busy Waiting) überbrückt. Leerlauf kann einfach implementiert werden, benötigt aber wertvolle Ressourcen, was für Echtzeitanwendungen Nachteile bringt.

Das zweite fundamentale Problem beim Entwurf von Multiprozessoren ist das Synchronisationsproblem, d.h. die Notwendigkeit, die Ordnung der Befehlsausführung gemäß der Datenabhängigkeiten zwischen den Befehlen einzuhalten. Normalerweise wird dies durch Einfügen von Synchronisationsprimitiven erreicht, was allerdings dazu führt, dass andere Prozesse warten müssen. Das Warten auf ein Synchronisationsereignis kann wieder durch Leerlauf des Prozesses erreicht werden.

2.2.2 Kommunikation

Ein wesentlicher Punkt bei der Bewertung der Leistungsfähigkeit von Parallelrechnern, ist der schnelle Austausch von Daten und nicht die numerische Leistungsfähigkeit der beteiligten Prozessoren [46]. Unter Kommunikation wird der Datenfluß innerhalb einer zu betrachtenden Architektur verstanden. Hiebei ist insbesondere die Kommunikation zwischen den Prozessorelementen, dem Prozessor und dem lokalen Speicher und dem Prozessor und dem globalen Speicher zu verstehen. Hierbei können die einzelnen Punkte über einfache Verbindungen oder durch komplexe Netzwerke realisiert werden. Häufig werden die Elemente über eine direkte Kopplung, durch Bussysteme oder über Netzwerke miteinander verbunden.

Man unterscheidet bei der direkten Kopplung zwischen den Kopplungsvarianten: Register, Speicher und Kanal. Abbildung 1 zeigt die verschiedenen Arten [14].

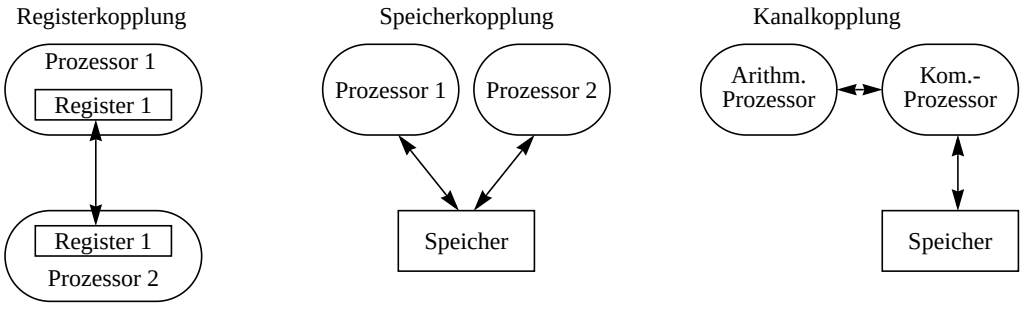


Abb. 1: Direkte Kopplung

Bei der Registerkopplung erfolgt die Kommunikation über gekoppelte Register, bei der Speicherkopplung über einen gemeinsamen Speicher und bei der Kanalkopplung über einen Kommunikationsprozessor, der dem Arbeitsprozessor den Datenaustausch abnimmt.

Bei der Registerkopplung werden die Kommunikationsregister der beteiligten Prozessoren durch Leitungen verbunden, über die der Inhalt ausgetauscht werden kann. Die Übertragung kann seriell

oder parallel stattfinden. Durch parallele Registerkopplung können im Allgemeinen Daten ohne Latenz übertragen werden. Ein Beispiel hierfür ist die Kopplung des sog. Kommunikations-Agenten und dem Prozessor-Agenten in der iWARP-Architektur [19]. Die Probleme, die bei dieser Registerkopplung auftreten sind:

- Wie werden die Daten zwischen zwei Knoten transferiert
- Wie informiert der eine den anderen, dass ein Wert gültig ist
- Wie kontrolliert der eine den anderen, dass nicht alle Daten gültig sind

Der iWARP-Prozessor besitzt ein spezielle Datenschnittstelle zwischen dem Kommunikations-Agenten und dem Prozessor. Diese beiden Register haben je zwei Ports zum Lesen und Schreiben, welche direkt in den Registersatz des iWARP-Prozessors abgebildet sind. Diese beiden Registerinhalte sind dem Kopf und dem Ende einer Kommunikations-Pipeline zugeordnet. Schreibt der Prozessor auf eines dieser Register, wird der entsprechende Wert der Kommunikations-Pipeline zugefügt, oder, falls der Prozessor von einer dieser Stellen liest, wird der entsprechende Wert aus der Pipeline entfernt.

Bei dem in Kap. 3 dargestellten MIMD Prozessor werden die beteiligten CPU’s über eine Registerkopplung miteinander verbunden. Erreicht wird dies über ein globales Register File. Allerdings erfolgt keine Kennzeichnung der Daten. Die Gültigkeit, wer der Empfänger ist, muss über Software sichergestellt werden. Außerdem werden die Werte des Preprozessors (siehe Kap. 6) in den Registersatz des MIMD Prozessors abgebildet, was ebenfalls einer Registerkopplung entspricht. Dadurch können Lade- und Speicherinstruktionen für die Daten des Preprozessors vermieden werden.

Bei der Speicherkopplung [14], [41] sind die beteiligten Prozessoren über einen gemeinsamen Speicher durch direkte Leitungen oder Busse miteinander verbunden. Alle Prozessoren können lesend oder schreibend auf den gemeinsamen Speicherbereich zugreifen, wobei jedoch Vorkehrungen getroffen werden müssen, um Zugriffskonflikte zu vermeiden, wodurch die Speicherkopplung i.d.R. langsamer ist als die Registerkopplung. Sie ist aber auch flexibler, da bei Verwendung von Multiportspeichern auch mehrere Prozessoren über ein und denselben Speicher kommunizieren können.

Der entwickelte MIMD Prozessor verfügt zusätzlich zur Registerkopplung über eine Speicherkopplung der vier CPU’s. Implementiert ist ein Vier-Port-Speicher, der allen CPU’s über Lade- und Speicherinstruktionen zugängig ist. Somit können die CPU’s mit einem Takt verzögert Daten untereinander austauschen.

Bei der Kanalkopplung wird die Verarbeitung der Daten in einem Prozessor und die Kommunikation durch einen weiteren geleistet. Dieser hat direkten Zugriff auf den Arbeitsspeicher. Der “Verarbeitungsprozessor” übergibt die Anfangsadresse und die Anzahl der Daten die übertragen werden sollen und widmet sich dann wieder der weiteren Verarbeitung. Die Kommunikation ist somit vom eigentlichen Prozessorelement entkoppelt. Die Geschwindigkeit zwischen dem Prozessorelement und dem Speicher ist im Allgemeinen sehr hoch, da dieser speziell für diese Aufgabe entworfen wurde.

Bei der Kopplung der Prozessoren über ein gemeinsames Bussystem kommunizieren mehrere Prozessoren über ein gemeinsames Medium mit dem Speicher. Meist ist der Zugriff erheblich langsamer als bei direkter Kommunikation, erlaubt aber auch die flexible Erweiterung der

Topologie. Es können schnell weitere Speicherelemente oder weitere CPU’s an das gemeinsame System angeschlossen werden. Prozessoren können Daten senden und empfangen oder eine Master-Funktion besitzen. Um Konflikte zu vermeiden wird häufig ein sog. Arbiter eingesetzt, der die Prozessoren vom Bus entkoppelt und diesen nach geeigneten Verfahren zuteilt. Bekannte Verfahren sind time-shared, token oder random access [44].

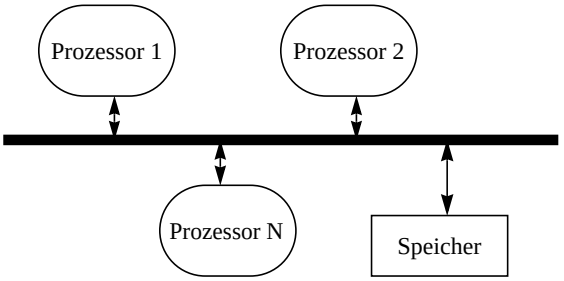


Abb. 2: Prinzip eines Bussystems

Verbindungsnetzwerke übernehmen das Routing der Daten und Befehle zwischen den angeschlossenen Prozessoren und Speicherelementen. Man unterscheidet zwischen statischen und dynamischen Netzwerken. Statische Netzwerke verbinden die beteiligten Elemente nach einem fest vorgegebenen Schema und einer fest vorgegebenen Transportzeit. Bekannte Verbindungsstrukturen sind Ring, Mesh, Cube, Baum usw. Dynamische Netzwerke verbinden die Elemente über Schalter, die beliebige Stellungen einnehmen können. Sind viele Teilnehmer angeschlossen ist der Hardwareaufwand unter Umständen sehr hoch und die Kommunikationszeit wird groß. Einstufige Netzwerke verbinden die Elemente über eine Schalterstufe. Sind viele Teilnehmer angeschlossen bietet sich das Hintereinanderschalten von mehreren Schaltern an, da sonst der Hardwareaufwand sehr hoch wird. Ein bekanntes Beispiel eines einstufigen Netzwerks ist der Crossbar Switch, für ein mehrstufiges Netzwerk lässt sich z.B. das Omega Netzwerk nennen. Abbildung 3 zeigt drei gebräuchliche Verbindungsnetzwerke [14].

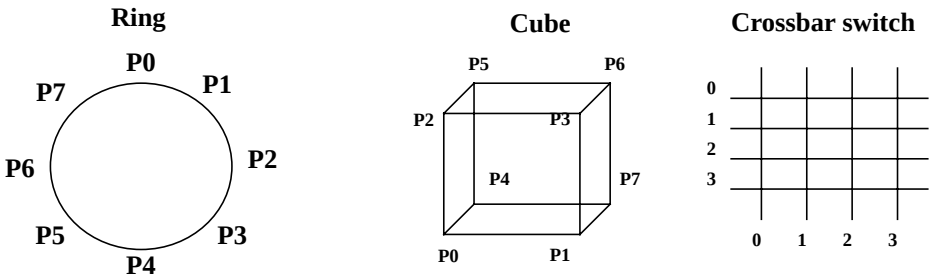


Abb. 3: Verbindungsnetzwerke

Der in Kap. 3 beschriebene MIMD Prozessor verfügt zusätzlich zur Register- und Speicherkopplung über ein Bussystem, an das busseitig verschiedene Peripherieelemente angeschlossen sind. Der Zugriff auf diesen Speicherbereich geschieht über Lade- und Speicher-Instruktionen. Die Zuteilung erfolgt über einen Arbiter mit absteigender Priorität. Weiter ist eine

Konfigurationseinheit an den Arbiter angeschlossen, die eine Master-Funktion besitzt. Im günstigsten Fall können zwei CPU's über diesen Speicher taktversetzt kommunizieren.

2.2.3 Synchronisation

Eine Synchronisation tritt auf, falls *Kooperation* oder *Konkurrenz* vorliegen, also falls Prozesse jeweils bestimmte Teilaufgaben innerhalb der Gesamtaufgabe erfüllen, was nach dem Erzeuger-/Verbraucher-System oder nach dem Auftraggeber-/Auftragnehmer-System geschehen kann [46]. Eine Konkurrenz liegt vor, falls die Aktivitäten eines Prozesses, die eines anderen zu behindern drohen. Die Koordination der Kooperation bzw. der Konkurrenz wird Synchronisation genannt, wobei die Unabhängigkeit der Abfolge von Aktivitäten verschiedener Prozesse eingeschränkt wird, so dass bestimmte Aktivitäten von Prozessen verzögert werden müssen. Um die gegenseitigen Aktivitäten verschiedener Prozesse untereinander abzustimmen, müssen sich die Prozesse gegenseitig mit Daten, sog. shared Variabels, versorgen, auf die Prozesse schreibend und lesend zugreifen können. Diese Daten werden für den Datenaustausch und die Synchronisation verwendet. Es wird zwischen der einseitigen Synchronisation, bei der zwei Prozesse insofern voneinander abhängig sind, als dass der erste Prozess die Voraussetzung für das richtige Ergebnis des zweiten Prozesses liefert und der mehrseitigen Synchronisation, bei der die zeitliche Abfolge der beteiligten Prozesse zwar gleichgültig ist, aufgrund von Schreib-/Schreib- oder Schreib-/Lese-Konflikten aber der erste Prozess nicht zusammen mit dem zweiten Prozess ausgeführt werden darf und umgekehrt, unterschieden. Im ersten Fall muss der zweite Prozess also solange warten bis der erste fertig ist. Im zweiten Fall muss entweder der erste oder der zweite warten bis der andere fertig ist. Allen Synchronisationsmaßnahmen ist gleich, dass Prozesse verzögert werden. Diese Verzögerung sollte eine endliche Dauer haben, da es sonst dazu kommen kann, dass ein Prozess auf ein Ereignis wartet, das nicht mehr eintreffen kann (engl. *deadlock*). Ein deadlock kann nur dann auftreten, wenn die vier folgenden Bedingungen erfüllt sind [46]:

- Die umstrittenen Betriebsmittel sind nur exklusiv nutzbar.
- Die umstrittenen Betriebsmittel können nicht entzogen werden.
- Die Prozesse belegen die schon zugewiesenen Betriebsmittel auch dann, wenn sie auf die Zuweisung weiterer Betriebsmittel warten.
- Es gibt eine zyklische Kette von Prozessoren, von denen jeder mindestens ein Betriebsmittel besitzt, das der nächste Prozess der Kette benötigt.

Demnach kann ein *deadlock* vermieden werden, indem ausgeschlossen wird, dass alle vier Zustände eintreten oder nach Eintritt eines *deadlocks* dieser beseitigt wird.

Zur Realisierung der Synchronisation sind verschiedenste Strategien bekannt. Drei gebräuchliche sollen hier vorgestellt werden. Diese stellen die Synchronisation aus Sicht des Programmierers dar. Es sind:

- Semaphor
- Schlossvariable
- Barriere

Ein Semaphor ist ein Datentyp, der aus einer positiven Integer-Variablen besteht, die über zwei Operationen - $P_{(S)}$ und $V_{(S)}$ - verändert werden kann [23], [46].

```
P(S): wenn S > 0
dann S:S-1,
sonst wird der Prozeß, der P(S) ausführt, suspendiert
V(S): S:S+1,
```

Die Operationen dürfen dabei nicht unterbrochen werden. Oft werden sie deshalb auch als unteilbare oder atomare Operationen bezeichnet. Ein wartender Prozess wird nach einer V-Anweisung reaktiviert, der seinerseits nach dem Erwecken die P-Operation erneut ausführt. Das nachfolgende Programmfragment [46] zeigt die Verwaltung eines beliebig großen Pufferspeichers mittels eines Semaphors. Die Operationen *get* und *put* sind unteilbare Operationen.

```
program erzeugerverbraucher
var n: semaphore;

procedure erzeuger;
begin
  repeat
   erzeuge;
   put;
   V(N)
  forever
  end;

procedure verbraucher;
begin
  repeat
   P(n);
   entnehme;
   verbrauche;
  forever;
end;
```

Das Semaphor kann als Differenz zwischen der Anzahl der von V erzeugten Signale und der Anzahl der von P verbrauchten Signale aufgefasst werden.

Bei Schlossvariablen[1] werden kritische Programmfragmente durch eine Schlossvariable verriegelt oder aufgesperrt. Ein Prozess, der einen kritischen Abschnitt betreten will, wartet solange, bis das Schloss offen ist, dann betritt er den Abschnitt und verriegelt das Schloss hinter sich, so dass kein weiterer Prozess den Abschnitt betreten kann. Nach Abarbeitung der Operationen wird das Schloss wieder geöffnet. Das Aufschließen muss von dem Prozess vollzogen werden, der auch das Schloss geschlossen hat. Ähnlich wie bei einem Semaphor besteht das Schloss aus einer, in diesem Fall, boolschen Variablen und den beiden Operationen lock und unlock. Zu den Variablen wird ein Satz an Funktionen definiert, welche die Funktionen lock und unlock ausführen. Hardwareseitig besteht die lock-Funktion aus den Operationen „Prüfen“ und „Setzen“, die als elementare Operation implementiert werden kann oder mittels einer swap-Anweisung zur Ausführung kommt. Grundsätzlich darf die Operation nicht unterbrochen werden [46].

1. Oft wird auch der Begriff Mutex (mutual exclusion) verwendet

Bei der Synchronisation mit Barrieren werden die beteiligten Prozesse, die an der Barriere ankommen, solange suspendiert, bis alle Prozesse an der Barriere angekommen sind. Oft wird so Verfahren, dass die Barrieren Variable mit der Zahl der Prozesse, die synchronisiert werden sollen, initialisiert wird. Wird nun eine *wait-barrier*-Anweisung von einem Prozess ausgeführt, wird dieser suspendiert und die Variable dekrementiert. Kommt der letzte Befehl an der Barriere an, wird die Variable auf den Initialisierungswert zurückgesetzt und die Suspendierung der Prozesse wird aufgehoben [41], [46].

Beispiele für die Hardwareimplementierung von Synchronisationsmechanismen speichergekoppelter Prozessoren sind TERA und HEP [45]. Beim HEP- Rechner sind spezielle Lade- und Speicherbefehle implementiert. Eine Ladeoperation kann so definiert werden, dass mit dem Zugriff auf diese Speicherstelle so lange gewartet wird, bis die Speicherstelle in den Zustand voll gewechselt hat. Ferner kann eine Instruktion ein Datum lesen und den Zustand auf ‘leer’ setzen. Trifft nun eine Speicheroperation auf diese Speicherstelle, wird diese Operation solange suspendiert, bis diese Stelle den Zustand leer erreicht bzw. falls dies der Fall ist, wird der Zustand auf voll gesetzt, so dass alle folgenden Speicherzugriffe warten müssen bis wieder der Zustand leer erreicht wird.

Im TERA-Rechner, der eine Weiterentwicklung des HEP-Rechners darstellt, wird jedem 64 Bit Datenwort ein Full/Empty Bit angehängt. Für die Lade- und Speicheroperationen sind unter Berücksichtigung dieses Flags die folgenden Zugriffsmodi implementiert [45]:

- Laden ohne Betrachten des ‘full’ bzw. ‘empty’ Bits.
- Warten auf Zustand ‘full’, nach dem Laden bleibt der Zustand ‘full’ erhalten.
- Warten auf Zustand ‘full’, nach dem Laden wird der Zustand auf ‘empty’ gesetzt.
- Speichern ohne Betrachten des ‘full’ bzw. ‘empty’ Bits.
- Warten auf Zustand ‘empty’, nach dem Speichern bleibt der Zustand ‘empty’ erhalten.
- Warten auf Zustand ‘empty’, nach dem Speichern wird der Zustand auf ‘full’ gesetzt.

Diese Operationen werden zur Synchronisation der Kontrollfäden verwendet. Ein eintretendes Warten auf einen Zustand ’full/empty’ geschieht durch sog. ‘busy waiting’, d.h. der Befehl, der die gesperrte Speicherstelle referenziert, wird solange wiederholt ausgeführt, bis der gewünschte Zustand erreicht ist. Zusätzlich kann ein Grenzwert definiert werden, so dass beim Erreichen eine Unterbrechung (trap) ausgeführt wird.

In dem in Kap. 3 dargestellten MIMD Prozessor wird die Synchronisation ebenfalls über Lese-/ Schreibinstruktionen auf gemeinsame Speicherbereiche gesteuert, wobei die Steuerung der Synchronisation über eine spezielle Maske erfolgt, die mit einer Instruktion in die sog. Synchronisationsregister geschrieben werden. Zur Steuerung der Synchronisation sind drei spezielle Befehle im Instruktionssatz vorgesehen. Insgesamt ähnelt das Verfahren dem Barrieren-Mechanismus, da die Synchronisation dadurch implementiert ist, dass die Verarbeitung suspendiert wird, sobald eine Barriere erreicht ist. Eine Beschreibung dieses Mechanismusses findet sich in Kap. 3.1.2.

2.2.4 Skalierbarkeit

Die Skalierbarkeit bezeichnet die Verkürzung der Verarbeitungszeit, die durch das Hinzufügen von Verarbeitungseinheiten entsteht, wobei das Programm unverändert bleibt. Wichtig ist eine

Mindestgröße der Problemstellung, da bei steigender Anzahl an Verarbeitungseinheiten ab einem bestimmten Punkt eine Sättigung eintritt. Zur Begriffsbestimmung sind die folgenden Variablen nötig [45]:

$P_{(1)}$: Anzahl der Operationen auf einem Einprozessorsystem

$P_{(N)}$: Anzahl der Operationen auf einem Mehrprozessorsystem

$T_{(1)}$: Ausführungszeit auf einem Einprozessorsystem in Taktzyklen

$T_{(N)}$: Ausführungszeit auf einem Mehrprozessorsystem mit N Prozessoren

Außerdem wird angenommen, dass $T_{(1)} = P_{(1)}$, da in einem Einprozessorsystem jede Operation in genau einem Schritt ausgeführt wird. Ferner kann $T_{(N)} \leq P_{(N)}$ angenommen werden, da in einem Multiprozessorsystem mit N Prozessoren mehr als eine Operation in einem Schritt ausgeführt werden kann. Daraus ergeben sich die Variablen $S_{(N)} = T_{(1)}/T_{(N)}$, die als Leistungssteigerung (engl. *speed up*) bezeichnet wird und die Effizienz $E_{(N)} = S_{(N)}/N$. Die Zahl N gibt die Anzahl der Prozessoren an, so dass die Effizienz eines Mehrprozessorsystems gerade so gut sein kann wie die eines Einzelprozessorsystems. Weitere Faktoren sind in diesem Zusammenhang der Mehraufwand für die Parallelisierung $R_{(N)} = P_{(N)}/P_{(1)}$, der den Mehraufwand für Organisation, Synchronisation und Kommunikation quantifiziert sowie der Parallelindex $I_{(N)} = P_{(N)}/T_{(N)}$, der den mittleren Grad der Parallelität angibt. Weiter ist die Auslastung $U_{(N)}=I_{(N)}/N$ zu nennen, die angibt, wieviele Operationen jeder Prozessor im Durchschnitt pro Zeiteinheit ausgeführt hat.

Die daraus resultierenden Größen können zur Beurteilung der Leistungsfähigkeit eines Mehrprozessorsystems im Vergleich zu einem Einzelprozessorsystem herangezogen werden.

3 *Eine neue MIMD Prozessor Architektur*

In diesem Kapitel wird die Hardwarestruktur eines neuen MIMD Prozessors erarbeitet. Die Entwicklung erfolgte als Designstudie, wobei die wesentlichen Eigenschaften und Elemente der Architektur durch eine Anwendung getrieben wurden (siehe Kap. 5).

Die Anforderungen, die an den Entwurf gestellt wurden sind:

- Maximal vier Datensätze parallel durch verschiedene Instruktionssequenzen zu verarbeiten.
- Die Datensätze in max. 1,8 µs zu verarbeiten, wobei pro Datensatz ca. 150 Takte benötigt werden.
- Daten zwischen den Verarbeitungseinheiten möglichst latenzfrei austauschen zu können.
- Algorithmen in Echtzeit auszuführen.
- Wenig Leistung zu verbrauchen.
- Wenig Chipfläche in Anspruch zu nehmen.
- Ein Bussystem für zukünftige Erweiterungen zu besitzen.

Daraus ergeben sich die folgenden abgeleiteten Anforderungen:

- Einen Synchronisationsmechanismus zur Verfügung zu stellen.
- Schnelle Arithmetik zu beinhalten.

Dementsprechend wird die nachfolgende Prozessorarchitektur als MIMD Architektur implementiert, die mit vier identischen CPU's maximal vier Datensätze verarbeiten kann. Nachfolgend werden die charakteristischen Merkmale dieser Architektur aufgezeigt, um dann die Funktionsblöcke des MIMD Prozessors 'topdown' vorzustellen. Im nächsten Kapitel wird die Funktionalität und die Architektur des RISC Prozessor Kerns beschrieben.

3.1 Charakteristische Eigenschaften

In diesem Kapitel werden die charakteristischen Eigenschaften des MIMD Prozessors beschrieben. Eingangs wird ein Überblick gegeben, um dann den Synchronisationsmechanismus und die Kommunikationsmöglichkeiten vorzustellen. Dann wird auf die Architektur eingegangen.

3.1.1 Übersicht

Das primäre Einsatzgebiet sieht für die Verarbeitung der maximal vier Datensätze ca. 1,8 µs Verarbeitungszeit vor. Pro Datensatz müssen Algorithmen ausgeführt werden, die etwa 150 Taktzyklen verbrauchen, wobei für jeden Datensatz unter Umständen verschiedene Algorithmen anzuwenden sind. Die Ergebnisse müssen am Ende der 1,8 µs vorliegen, weshalb der MIMD

Prozessor aus vier 32 Bit RISC CPU’s aufgebaut wird, die mit jedem Takt eine Instruktion verarbeiten können. Die relativ geringe Zielfrequenz von 120 MHz ermöglicht es, eine zweistufige Pipeline zu verwenden, wodurch Datenabhängigkeiten vermieden werden können.

Für den Instruktionsspeicher und den Datenspeicher wird jeweils ein Vier-Port-Speicher implementiert, so dass alle CPU’s zu jeder Zeit unabhängig voneinander auf die beiden Speichermodule zugreifen können. Durch die Verwendung von Speichermodulen mit vier Ports kann einerseits Chipfläche eingespart werden [36] und andererseits können die CPU’s über den Datenspeicher Daten untereinander austauschen. Ein Cache Speicher ist in diesem Zusammenhang nicht erforderlich, da der Zugriff auf die Speichermodule unmittelbar erfolgt. Der Zugriff auf den Datenspeicher erfolgt über Lade- und Speicherbefehle. Zur verzögerungsfreien Kommunikation ist ein globales Register File (GRF) vorgesehen, auf das alle CPU’s Zugriff haben.

Ferner verfügt der MIMD Prozessor über einen Synchronisationsmechanismus, mit dem zwei, drei oder vier CPU’s auf einen Zeitpunkt synchronisiert werden können. Ausnahmebehandlungen werden von jeder CPU separat vorgenommen, d.h. für jede CPU ist eine Interrupt Einheit implementiert, die den Programmfluss in maximal zwei Takten wechselt. Der Rücksprung erfolgt unmittelbar.

Die Arithmetisch-Logische-Einheit (ALU) innerhalb der CPU’s stellt 16 verschiedene Operationen auf Festkommadaten zur Verfügung. Bis auf die Division können alle Operationen in einem Takt ausgeführt werden. Für die Division wurde ein Radix-4 Verfahren implementiert, das einen minimalen Hardwareaufwand hat und in 18 Taktzyklen eine 64:32 Division durchführt. Die ALU ist auf Fläche optimiert.

Zum Anschluß verschiedener Peripheriegeräte sind ein globaler und lokale Busse implementiert. Über die lokalen Busse kann jede CPU unmittelbar mit einer Netzwerkschnittstelle kommunizieren. Der globale Bus ist über einen Arbiter erreichbar, der nach absteigender Priorität den Bus an die CPU’s oder die Konfigurationseinheit verteilt. An den globalen Bus sind im Wesentlichen die Interrupt-Einheiten, die Konstantenspeicherelemente, die Netzwerkschnittstelle, die Konfigurationsparameter und die Datenspeicher für die Rohdaten des Preprozessors angehängt.

Die Sprungbehandlung wird im MIMD Prozessor ohne Branch Prediction durchgeführt. Stattdessen wird hinter jeder dekodierten Sprunginstruktion eine NOP-Instruktion eingefügt, so dass eine Sprunginstruktion in zwei Taktzyklen verarbeitet wird.

Weiterhin ist eine Konfigurationseinheit zur Initialisierung des Prozessors implementiert. Mit ihr kann über den globalen Bus auf den Instruktionsspeicher sowie auf den Datenspeicher zugegriffen werden. Zusätzlich können die Konfigurationsregister der CPU’s sowie des Preprozessors (siehe Kapitel 6) durch diese Einheit konfiguriert werden.

Abbildung 4 gibt einen Überblick über die implementierten Blöcke.

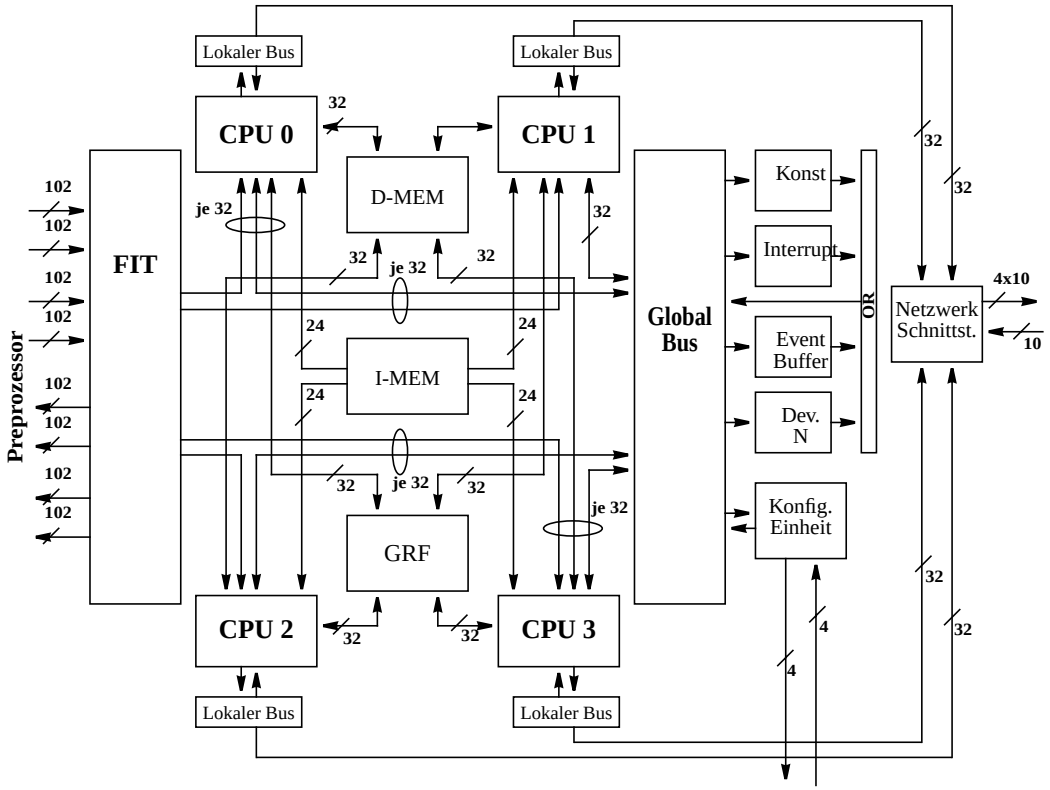


Abb. 4: Blockdiagramm des MIMD Prozessors

Dargestellt sind die wesentlichen Komponenten des MIMD Prozessors. Die vier CPU's können über das GRF und den gemeinsamen Datenspeicher kommunizieren. Ferner ist ein globaler Bus vorgesehen, über den die CPU's ebenfalls Daten austauschen können. Die Schnittstelle zum Preprozessor ist das sog. FIT Register File. Die Schnittstelle zur Außenwelt wird durch eine Netzwerkschnittstelle zur Verfügung gestellt. Der Prozessor wird durch eine Konfigurationseinheit initialisiert.

Die CPU's beziehen ihre Daten aus dem privaten oder globalen Registersatz (PRF bzw. GRF), aus dem Registersatz zwischen Preprozessor und MIMD Prozessor (FIT) oder aus dem Registersatz, der die Konstanten speichert (CONST). Ferner können Daten über Ladebefehle aus dem Datenspeicher oder vom globalen Bus geladen werden. Das PRF ist in jede CPU integriert und nur für diese sichtbar. Hier werden die lokalen Daten gespeichert, die anderen CPU's nicht zugängig sein müssen. Das GRF dient dem latenzfreien Datenaustausch der CPU's untereinander, kann aber ebenso zur globalen Speicherung von Daten verwendet werden. Das Fit Register ist die Schnittstelle zwischen dem Preprozessor und dem MIMD Prozessor. Es wird vom Preprozessor verwendet, um dort die Ergebnisse seiner Berechnung abzulegen, um nach erfolgter Verarbeitung dem MIMD Prozessor einen ausgewählten Datensatz zur Verfügung zu stellen. Durch die Projektion der Daten des Preprozessors in den Registersatz des MIMD Prozessors ist es möglich, unmittelbar mit der Verarbeitung der Daten zu beginnen. Für den MIMD Prozessor ist dieser Registersatz nur lesbar. Das CONST-Register beinhaltet Konstanten, die im Verlauf der Berechnung immer wieder gebraucht werden (0,1,2,usw.). Für jede CPU ist ein Registersatz zur Speicherung der Konstanten vorgesehen. Diese befinden sich als Peripheriegeräte im Adressraum

des globalen Busses und werden über die Konfigurationseinheit beschrieben. Die einzelnen CPU's greifen auf ihren Registersatz, der die Konstanten enthält, direkt zu, können aber andererseits diese auch über den globalen Bus auslesen oder beschreiben. Der gemeinsame Datenspeicher ist ausschließlich über Lade- und Speicherbefehle den CPU's zugängig. Dadurch, dass dieser Speicher vier unabhängige Ports sowohl zum Lesen als auch zum Schreiben besitzt, kann jede CPU mit jedem Takt auf diesen Speicher zugreifen. Der Instruktionsspeicher hat die gleiche Architektur wie der Datenspeicher, nur, dass die Breite der Datenwörter und die Tiefe des Speichers variiert. Außerdem verfügt dieser nur über einen Schreibport, worüber das Programm durch die Konfigurationseinheit in den Speicher geschrieben wird (siehe Kap. 3.2.3).

Der MIMD Prozessor ist als Registermaschine entworfen, d.h. alle Operationen werden zwischen Registern ausgeführt. Der Zugriff auf den Datenspeicher erfolgt über Load/Store Operationen. Externe Peripheriegeräte werden über den globalen Bus angesprochen, der auch für zukünftige Anwendungen ausreichend Erweiterungsspielraum läßt. Eine leistungsfähige 32 Bit ALU erlaubt auch komplizierte arithmetische Operationen auszuführen, wobei auf eine kompakte Architektur geachtet wurde. Die Kommunikation mit einer Netzwerk-Schnittstelle findet über einen lokalen Bus statt, den jede CPU besitzt. In diesen Speicherbereich kann mit jedem Takt ein Datum geschrieben werden. Alle Schnittstellen können nach außen über ein JTAG-Interface kontrolliert werden. Die Aktivität des MIMD Prozessors wird über eine Power-Management- Einheit gesteuert. Sie kontrolliert die Taktsignale aller integrierten Einheiten, so dass der Leistungsverbrauch minimal gehalten werden kann.

3.1.2 Der Synchronisationsmechanismus

Mit Hilfe des Synchronisationsmechanismus können die CPU's auf einen Punkt synchronisiert werden bzw., falls erforderlich, auch einzelne CPU's untereinander. Die Idee des Mechanismus beruht auf dem Barrierenkonzept. Die beteiligten CPU's, welche die Barrierensynchronisation über eine spezielle Instruktion aufrufen, warten so lange, bis die letzte CPU die Barriere erreicht, auf die gewartet werden soll. Diese CPU hebt daraufhin die Barriere auf.

Für die Implementierung wurde das globale Register File um vier Register erweitert, welche mit den Speicherstellen des GRF korrespondieren. Logisch wird für diesen Mechanismus das GRF in vier Teile eingeteilt. Jede CPU ist demnach vier Registern zugeordnet.

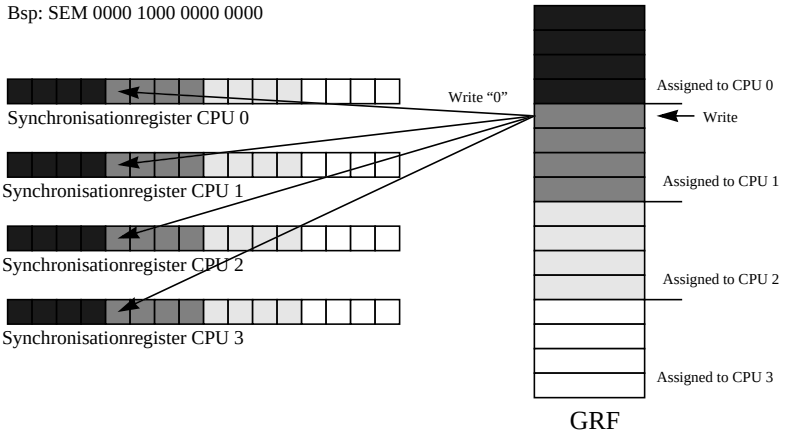


Abb. 5: Synchronisationsmechanismus

Die Synchronisationsregister sind jeweils einer CPU zugeordnet. Eine Schreibaktion auf ein Register des GRF setzt das zugeordnete Flip-Flop der vier Synchronisationsregister zurück. Hier werden durch die Instruktion *SEM* mit dem Argument “0000 1000 0000 0000” die Synchronisationsregister beschrieben. Wird mittels einer Schreibinstruktion auf die fünfte Speicherstelle des GRF’s geschrieben (GRF[4]) wird das gesetzte Bit zurückgesetzt, also in diesem Fall „0000 0000 0000 0000“.

Eine Schreibaktion einer CPU auf ein Register im GRF erzeugt das Zurücksetzen der entsprechenden Bitstelle innerhalb jedes der vier Synchronisationsregister. Gesetzt werden die Register durch eine spezielle Instruktion (*SEM*). Als Argument erhält der Befehl ein 16 Bit breites Wort, welches den Wert darstellt, der in das Synchronisationsregister geschrieben werden soll. Durch eine weitere Instruktion (*SYN*) wird der Programmzähler der entsprechenden CPU angehalten, solange nicht die bitweise ODER-Verknüpfung des Synchronisationsregisters ’1’ ergibt. Mit Hilfe der Instruktion *SYT* kann der Inhalt des Synchronisationsregisters ausgelesen werden. Die Synchronisation verläuft nach folgendem Schema:

Zur Synchronisation wird erst das Synchronisationsregister beschrieben. Das zeigt an welche CPU’s beteiligt sind. Sobald eine CPU die Barriere in Form der SYN-Instruktion erreicht, suspendiert sie ihren Programmzähler. Die Verarbeitung wird durch die letzte CPU freigegeben, die das SYN-Register zurücksetzt.

Seitens der implementierten Hardware ist der Aufwand für den Synchronisationsmechanismus gering. Nur vier Register und einige Oder-Gatter sind im Wesentlichen erforderlich, um den Mechanismus zu implementieren.

3.1.3 Kommunikation der CPU's

Die CPU's können über drei Wege Daten miteinander austauschen:

- über das Globale Register File (GRF),
- über den globalen Datenbus,
- über den Datenspeicher (D-MEM).

Die Kommunikation über das GRF entspricht einer Registerkopplung wie sie in Kap. 2 dargestellt wurde. Der Austausch von Daten erfolgt unmittelbar, d.h. falls eine CPU zu einem Zeitpunkt einen Wert in das GRF schreibt, kann die zweite CPU diesen Wert bereits im nächsten Takt lesen.

Der Datenaustausch über den Datenspeicher verläuft über Lade- und Speicherinstruktionen, d.h. die erzeugende CPU schreibt einen Wert in den Datenspeicher, den eine andere CPU über eine Ladeinstruktion in den eigenen Registersatz holt.

Die Kommunikation über den globalen Datenbus wird durch den Arbiter reglementiert, kann aber im günstigsten Fall ebenso verlaufen, wie bei der Kommunikation über den Datenspeicher. Verwendet aber beispielsweise CPU 0 ständig den globalen Bus, kommen die anderen CPU's nicht zum Zug und der Datenaustausch kann im ungünstigsten Fall gar nicht durchgeführt werden.

Bei allen Verfahren muss den CPU's angezeigt werden, dass der von CPU x geschriebene Wert gültig ist und für wen dieser Wert bestimmt ist. Außerdem ist der Zeitpunkt des Datenaustauschs zu beachten, da verschiedene Algorithmen unter Umständen verschiedene Verarbeitungszeiten erfordern und apriori nicht feststeht, welche CPU zuerst an den Punkt des Datenaustauschs kommt. Der Prozessor stellt hierfür den oben beschriebenen Synchronisationsmechanismus zur Verfügung. Zusätzlich können die in Kapitel 2.2.3 beschriebenen Mechanismen implementiert werden. Grundsätzlich lassen sich Prozeduren für die Synchronisation mittels Semaphore, Lock Variablen oder Barrieren implementieren.

3.2 Architektur des MIMD Prozessors

In diesem Kapitel wird die Architektur und die Funktion einzelner Bausteine beschrieben. Es werden die Schnittstellen und die gemeinsamen Speicher erläutert.

3.2.1 Globales Register File (GRF)

Das GRF besteht aus 16 Einträgen mit einer Wortbreite von 32 Bit. Es hat vier Lese- und vier Schreibports, über welche die CPU's auf das GRF zugreifen können. Für den Fall, dass zwei CPU's zur gleichen Zeit auf eine Speicherstelle zugreifen, ist eine Prioritätslogik integriert, welche derjenigen CPU mit der kleineren Identitätsnummer das Recht zum Schreiben einräumt. Dadurch wird vermieden, dass beide Schreibvorgänge erfolglos sind. Das Schreiben wird über vier separate *write enable* Leitungen ermöglicht. Zum Lesen sind vier Leseports implementiert. Diese sind als unkritisch zu betrachten, da beim Lesen keine Konflikte auftreten können. Die Synchronisation der vier CPU's wird ebenfalls durch das GRF zur Verfügung gestellt. Hierfür verfügt der Block über weitere Ein- und Ausgänge zum Lesen und Beschreiben der Synchronisationsregister. Insgesamt

kann jede CPU mit jedem Takt vom GRF lesen bzw. darauf schreiben. Das GRF verwendet als Speicherelemente Flip-Flops. Das GRF mit seinen Ports und dem internen Aufbau ist in Abbildung 6 dargestellt.

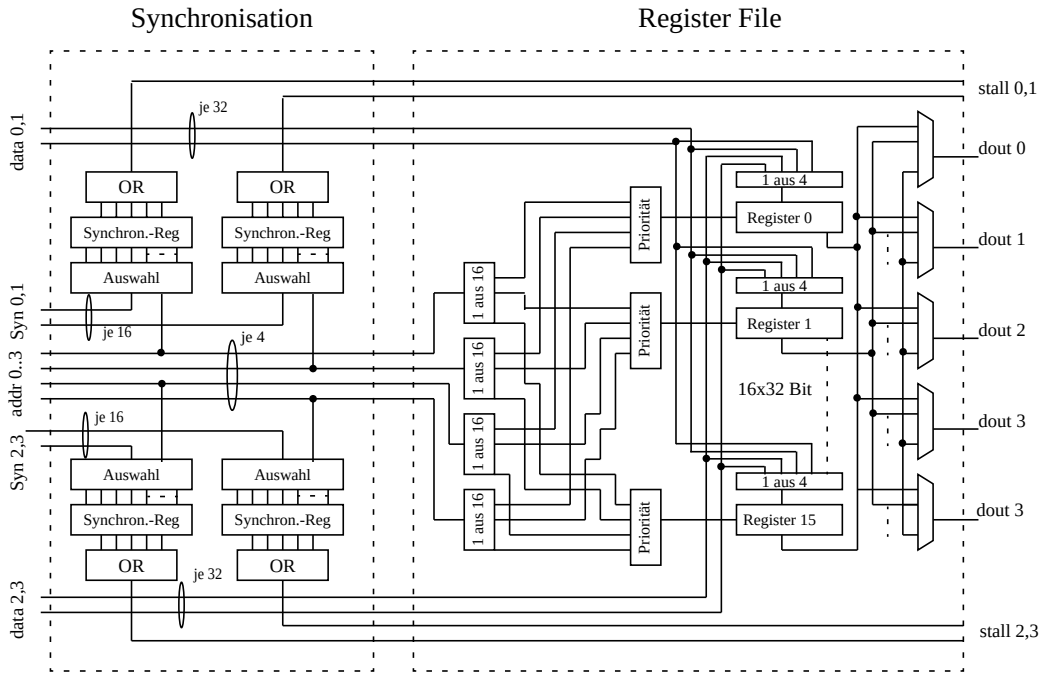


Abb. 6: Globales Register File

Das GRF teilt sich in den Register Block und in den Synchronisationsblock. Die Signale “stall” geben ein Flag an die Programmzähler der entsprechenden CPU, welche die Verarbeitung unterbrechen, sobald eine SYN Instruktion dekodiert wurde und das Flag gesetzt ist.

Die Abbildung zeigt auf der rechten Seite das Register File mit den Datenspeichern und den Adressmultiplexern. Ferner sind die Prioritätslogiken dargestellt. Links ist die Synchronisationseinheit dargestellt. Sie besteht aus den vier Synchronisationsregistern und einer Auswahllogik, welche die entsprechende Bitstelle zurücksetzt, falls auf ein Register innerhalb des GRF’s geschrieben wurde. Die logische ODER-Verknüpfung setzt das stall-Signal, was den Programmzähler der entsprechenden CPU anweist, den Programmzähler nicht mehr zu inkrementieren, insofern ein SYN-Befehl zur Ausführung kommt.

3.2.2 Der Datenspeicher und Instruktionsspeicher

Der Datenspeicher dient dem MIMD Prozessor zum Speichern der Daten. Er besitzt vier unabhängige Lese- und vier Schreibports. Durch die Vier-Port-Realisierung kann jede CPU zu jedem Zeitpunkt von dem Speicher lesen bzw. auf ihm schreiben. Wollen mehrere CPU’s im gleichen Takt auf die gleiche Speicherstelle schreiben, wird an diese Stelle ein ungültiger Wert gespeichert. Jedes Speicherwort ist 39 Bit breit und beinhaltet 32 Bit Daten und 7 Bit Hamming-kodierte Kontrollstellen. Die vier Hamming-Encoder und -Decoder befinden sich im Eingangs-

bzw. Ausgangspfad des Speichers, so dass die Hamming Kodierung für die CPU's nicht sichtbar ist. Der Hamming-Status der einzelnen Decoder wird nach außen geführt und durch die Power-Management-Einheit ausgewertet. Daraufhin kann bei einem Bitfehler ein Interrupt auslöst werden.

Der Speicher hat ein asynchrones Verhalten beim Lesen und ein synchrones Verhalten beim Schreiben. Dadurch kann die zweistufige Pipelineverarbeitung auch beim Zugriff auf den Speicher unverändert fortgeführt werden. Andernfalls müssten diese Spezialfälle berücksichtigt werden oder es wären zwei weitere Pipelinestufen notwendig.

Innerhalb der ersten Realisierung des MIMD Prozessors (Trap 1) besteht der Datenspeicher aus 3x64 Wörtern und 1x32 Wörtern, wobei der letztgenannte Block mit Flip-Flops und die drei anderen Speicher durch SRAM Zellen implementiert wurden. Der Aufbau des Datenspeichers ist in Abbildung 7 dargestellt und zeigt die vier Speicherblöcke und die Blöcke zur Hamming-Kodierung bzw. Dekodierung. Der Adressdecoder wird zum Lesen und Schreiben verwendet. Es erfolgt keine Zwischenspeicherung der Daten. Ein Lese- bzw. Schreibzugriff erfolgt innerhalb eines Taktes.

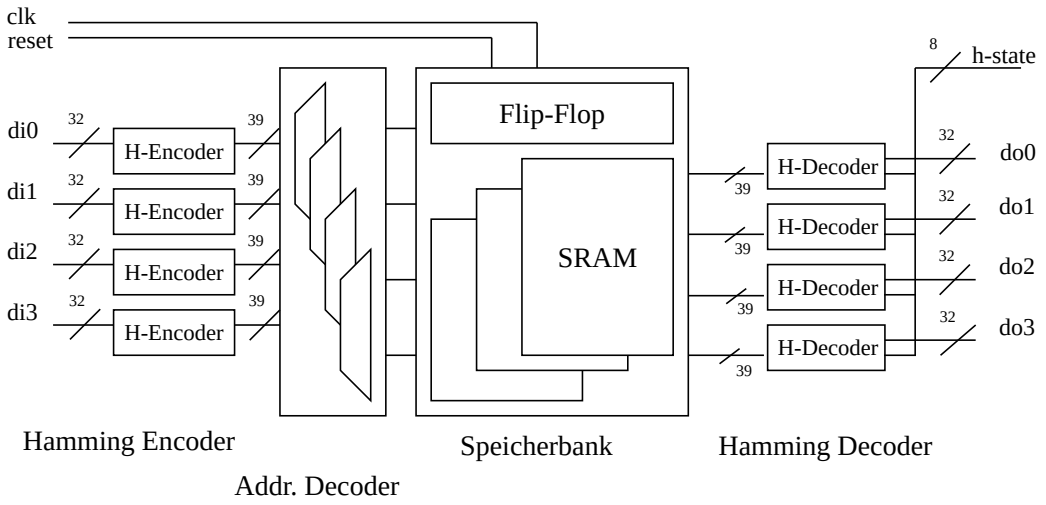


Abb. 7: Datenspeicher mit Hamming En- und Decodern
Die Speicherelemente, die durch Flip-Flops implementiert sind, haben separate Adressdecoder, die hier nicht dargestellt sind.

Der Vier-Port-Speicher ist eine Variante für die Implementierung eines gemeinsamen Speicherbereichs. Eine weitere Möglichkeit ist ein interner Bus mit vier Speicherbänken, die zwar leichter zu implementieren sind, aber den Nachteil der Arbitrierung haben und somit unter Umständen immer mit einer Verzögerung zu rechnen ist. Diesen Nachteil hat ein Speicher mit vielen Eingängen nicht. Außerdem sind die Dimensionen von mehreren Speichermodulen immer größer als eine Multi Port Variante, was den Stromverbrauch ansteigen läßt. Allerdings steigt die Signallaufzeit linear mit jedem Port [36], was auf die Zeitkonstante zurückzuführen ist, die durch

die Kapazität der Bit/Nichtbit-Leitung und dem Widerstand des beteiligten NMOS Transistors bestimmt wird. Deshalb kann ein Speicher mit vielen Ports nur endliche Dimensionen annehmen. Für eine Vier-Port Variante und einer Taktzyklusszeit von 8,33 ns ist das unproblematisch.

Abbildung 8 zeigt einen Lesezyklus gefolgt von einem Schreibzyklus. Der Lesezyklus beginnt mit dem Vorladen der Bit/Nichtbit-Leitung bis zur fallenden Taktflanke auf den Wert (V_{dd}-V_t). Anschließend wird die Adresse durch ein 8-fach Und-Gatter dekodiert (6 Bit Adresse, 1 Bit invertierter Takt, 1 Bit invertiertes Block Enable). Eine Verzögerungseinheit wartet 1,2 ns, so dass sich auf der Bit- und Nichtbit-Leitung eine Spannungsdifferenz von ca. 100 mV aufbauen kann. Dann werden die Ladungsverstärker aktiviert, die in einen definierten Zustand kippen und den Wert auf den Ausgang propagieren. Für den verwendeten 0,18 µm CMOS Prozess (UMC0.18) ergibt sich somit eine Zugriffzeit von ca. 5,5 ns. Die restliche Zeit des Taktes wird dazu verwendet, die Werte zum MIMD Prozessor zu übertragen.

Das Schreiben auf den Speicher verläuft analog zum Lesen, nur dass hier kein Vorladen der Bit- bzw. Nichtbit-Leitung notwendig ist. Der angelegte Wert treibt nach dem Dekodieren, was in der ersten Takthälfte liegt, die selektierte Bitzelle, die ihrerseits in den gewählten Zustand kippt. Der Speicher akzeptiert wieder Änderungen bis zur fallenden Taktflanke und übernimmt 1,2 ns später die Werte in die Speicherelemente.

Das Schreiben und Lesen auf dem Speicherbereich, der aus Flip-Flops besteht, ist als unkritisch zu betrachten, da hier die gesamte Taktperiode zur Verfügung steht. Durch die geringe Tiefe des Speicherbereiches ist die Dekodierung der Adressen relativ kurz (ca. 1 ns), so dass die Zugriffszeit auf einen Speicherwert bei ungefähr 2 ns liegt.

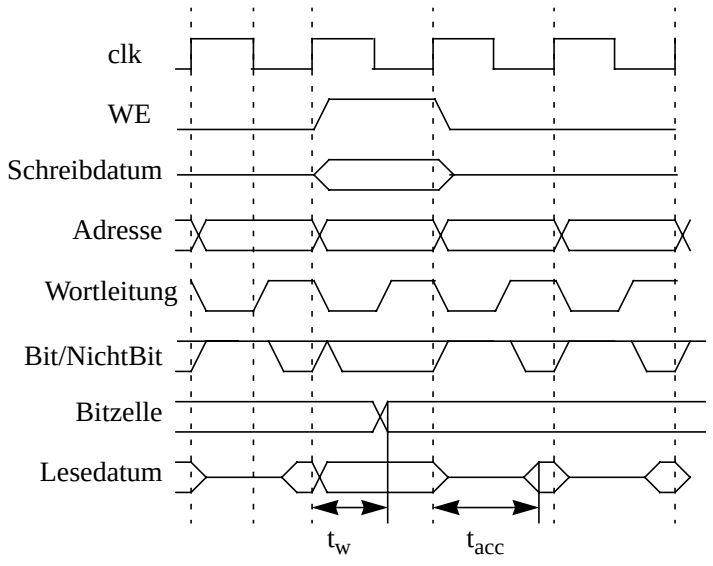


Abb. 8: Lese- und Schreibzugriff auf den Datenspeicher
Die Variable t_w ist die Zeit, bis ein Datum gültig in den Speicher geschrieben ist. Die Zeit t_{acc} ist die Zugriffszeit des Speichers.

Der Instruktionsspeicher wurde analog zum Datenspeicher als Vier-Port-Speicher implementiert. Allerdings verfügt er nur über einen Dateneingang, über den der Instruktionsspeicher initialisiert

wird und vier Datenausgänge, worüber die CPU’s die Instruktionen erhalten. Um 1-Bit Fehler korrigieren zu können, wird das 24 Bit Speicherwort durch eine Hamming-Kodierung auf 30 Bit erweitert. Zur Erhöhung der Packungsdichte und Senkung des Stromverbrauchs wurde der größte Teil des Speichers innerhalb des Prototypen Trap 1 als Full Custom Speicher implementiert [36]. Die weiteren Speicherzellen wurden wie beim Datenspeicher durch Flip-Flops realisiert. Insgesamt ist der Speicher in Blöcke zu je 64 Wörtern organisiert. Dabei entfallen die Worte der Speicheradressen 0x0000 bis 0x003F auf die erste Implementierungsvariante, der Rest, also bis zur Speicheradresse 0x00FF entfällt auf die Full Custom Implementierung.

Der Instruktionsspeicher hat ein asynchrones Verhalten beim Lesen und ein synchrones Verhalten beim Schreiben. Gleichzeitiges Lesen von einer Speicherstelle ist möglich. Beschrieben wird der Speicher von der Konfigurationseinheit (Kap. 3.2.3).

Beide Speicherblöcke verwenden Hamming-Kodierung [25], [38] um 1 Bit Fehler korrigieren und 2 Bit Fehler erkennen zu können. Auf diese Weise ist es möglich, die Ausbeute (*engl. Yield*) bei der Produktion zu erhöhen. Abbildung 9 zeigt das implementierte Verfahren für den Instruktionsspeicher. Genutzt wurde ein (29,24) Hamming-Code, der aus einem gekürzten (31,26) Code hervorgegangen ist. Der Code hat einen Hamming-Abstand von d=3. Durch Erweitern der H-Matrix um eine Zeile (nur Einsen) zum (30,24)-Code erhält man einen Hamming-Abstand d=4, so dass zusätzlich 2 Bit Fehler erkannt werden können. Dies wird durch die XOR-Verknüpfung aller Datenbits erreicht und repräsentiert die Parität (do29). Der Hamming-Encoder verknüpft das erzeugte Syndrom über ODER-Gatter und bildet mit der XOR-Verknüpfung aller eintreffenden Stellen (Parity Check) den Hamming-Status. Dieser ist folgendermaßen zu interpretieren: ’00’ - kein Fehler, ’01’ - Paritätsfehler, ’10’ - 2 Bit Fehler, 1 Bit Fehler (korrigiert). Er wird an die Power-Management-Einheit des MIMD Prozessors übergeben. Alle Fehler werden separat gezählt und lösen bei Überschreitung einer einstellbaren Schwelle einen Interrupt aus. Die drei Zählerstände befinden sich im Adressbereich des globalen Busses, so dass diese von den CPU’s bzw. von der Konfigurationseinheit ausgelesen werden können. Die Fehlerbehandlung wird über Software gesteuert.

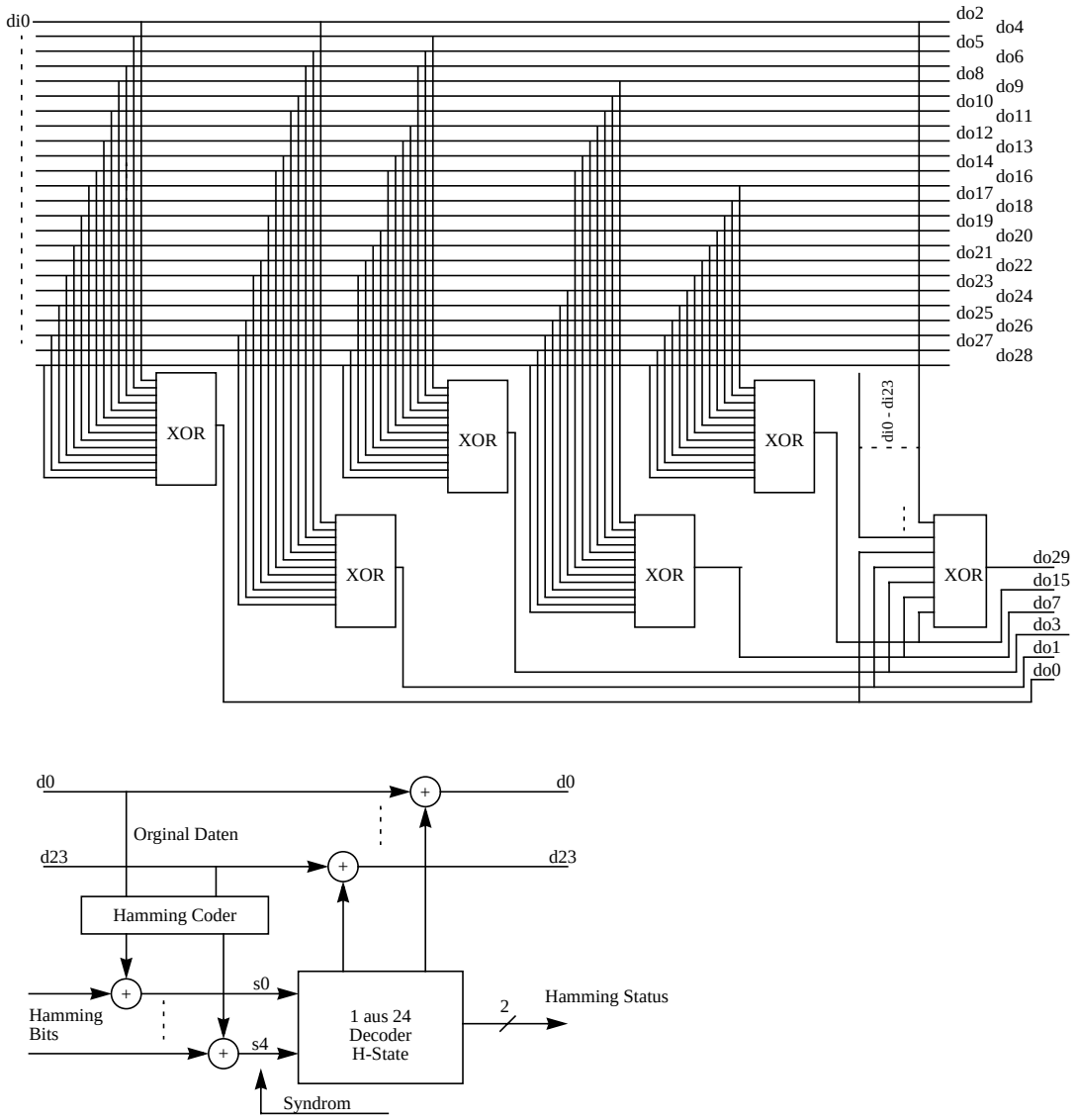


Abb. 9: Hamming-Kodierung bzw. Dekodierung [38] für den Instruktionsspeicher

3.2.3 Schnittstellen

Der MIMD Prozessor verfügt über vier Schnittstellen, um mit seiner Außenwelt zu kommunizieren. Das ist die Schnittstelle zum Preprozessor mit Registerkopplung zum MIMD Prozessor, die Schnittstelle zum globalen Bus, vier lokale Busse und die Konfigurationseinheit.

Die erstgenannte Schnittstelle dient der Registerkopplung des Preprozessors mit dem MIMD Prozessor. Die zweite Schnittstelle dient dem Anschluß externer Peripheriegeräte an den Prozessor. Je ein lokaler Bus für jede CPU dient der verzögerungsfreien Kommunikation mit der Netzwerkschnittstelle. Die Konfigurationseinheit hat Zugriff auf alle Module, die sich im Adressraum des globalen Busses befinden sowie den Daten- und den Instruktionsspeicher.

3.2.3.1 Busschnittstelle des MIMD Prozessors

Die Busschnittstelle des MIMD Prozessors dient dem Anschluß externer Peripheriegeräte an den Prozessor. Von Seiten des MIMD Prozessors sind die vier CPU's und eine Konfigurationseinheit an den globalen Bus über einen Arbiter angeschlossen. Dieser entscheidet welche CPU den Bus zum Lesen bzw. zum Schreiben zur Verfügung gestellt bekommt. Der Arbiter verwaltet einen 16 Bit breiten Adressraum, der in 16 Segmente aufgeteilt ist. Die Aufteilung ist in Tabelle 2 dargestellt:

.

Adresse	Segment
0x0000 - 0x0FFF	Lokaler Bus
0x1000 - 0x1FFF	Interrupt Controller 0..3
0x2000 - 0x2FFF	Event Buffer (Preprozessor)
0x3000 - 0x3FFF	Konfigurierbare Parameter des Preprozessors
0x4000 - 0x4FFF	Konstanten der CPU's 0..3
0x5000 - 0x5FFF	Powermanagement des Gesamtsystems
0x6000 - 0x6FFF	Zum allgemeinen Gebrauch
0x7000 - 0x7FFF	Netzwerk Interface
0x8000 - 0xEFFF	unbenutzt
0xF000 - 0xFFFF	Reserviert für spezielle Werte

Tab. 2: Aufteilung des Adressbereichs des globalen Busses

Über den Speicherbereich des lokalen Busses können die vier CPU's direkt mit der Netzwerk Schnittstelle [1], [39] kommunizieren. Auf diesen Speicherbereich kann jede CPU direkt, also ohne Arbitrierung und somit ohne Verzögerung, zugreifen. Physikalisch ist dieser Speicherbereich mit dem globalen Bus gekoppelt. Eine Unterscheidung findet über separate Lade-Speicherinstruktionen statt.

"Interrupt Controller 0..3" bedeutet, dass dieser Speicherbereich von den Interrupt Controllern belegt wird. Dort sind die Interrupt-Vektor-Tabellen (IVT) und die Konfigurationswerte der Interrupt Controller abgelegt. Das Segment "Event Buffer" enthält die Daten, die der Preprozessor von den ADC's erhält. Die Konfigurationswerte des Preprozessors und der Power-Management-Einheit befinden sich ebenfalls im Adressraum des globalen Busses. Im Segment „Konstanten der CPU's 0..3" sind die Konstanten der CPU's abgelegt, die im Programmverlauf häufig verwendet werden. Außerdem ist dort die CPU-Id enthalten.

Der globale Bus hat gegenüber dem MIMD Prozessor ein synchrones Verhalten beim Schreiben und ein asynchrones Verhalten beim Lesen, womit der Zugriff ein analoges Verhalten zum Datenspeicher aufweist (vgl. Kap. 3.2.2). Eine CPU kann mit jedem Takt auf eine Speicherstelle zugreifen, die an diesem Bus angeschlossen ist, insofern sie den Bus durch den Arbiter zugeteilt bekommen hat. Das Lesen erfolgt unmittelbar, d.h. beim Anlegen der Adresse antwortet die angesprochene Einheit noch im selben Takt, so dass die CPU den gelesenen Wert in ein internes Register mit der nächsten steigenden Taktflanke übernehmen kann. Will eine CPU durch eine

Leseinstruktion auf den Bus zugreifen, wird dies in der ersten Pipelinestufe im Befehlsdecoder anhand der dekodierten Instruktion festgestellt. Die Anforderung wird direkt an den Arbiter übergeben ohne in ein Pipelineregister geschrieben zu werden. Dieser entscheidet bei mehreren gleichzeitigen Anforderungen, welche CPU den Zugriff erhält. Dabei ist die Priorität mit absteigender CPU Nummer sortiert. Diejenigen CPU's, welche den Bus nicht zugeteilt bekommen, erhalten ein Flag (B) gesetzt, das signalisiert, dass der Zugriff erfolglos war. Auf das Flag kann im weiteren Programmverlauf gesprungen werden. Dieses sog. Non Blocking Verhalten verhindert, dass die Verarbeitung einer CPU aufgrund eines verweigerten Zugriffs aussetzt. Vielmehr arbeiten die CPU's weiter und es kommt nie zu einem Stillstand. Die Konfigurationseinheit hat höchste Priorität auf den Arbiter, d.h. sie bekommt, bei erfolgter Anforderung, zu jeder Zeit den Bus zugeteilt und kann im folgenden Takt auf dem Bus schreiben.

Um interne Tristate Busse im Chip zu vermeiden, sind separate Busse zum Lesen und Schreiben implementiert. Die Datenausgänge der Module, die an den globalen Bus angeschlossen sind werden ODER-verknüpft, bevor sie mit dem Bus des lokalen Busses ODER verknüpft werden, um dann an den MIMD Prozessor übergeben zu werden. Deshalb liefern alle Einheiten am Ausgang seitens des Busses einen Nullvektor, insofern sie nicht adressiert wurden. Einzig das angesprochene Element liefert Daten. Die Architektur der Schnittstelle ist in Abbildung 10 dargestellt:

.

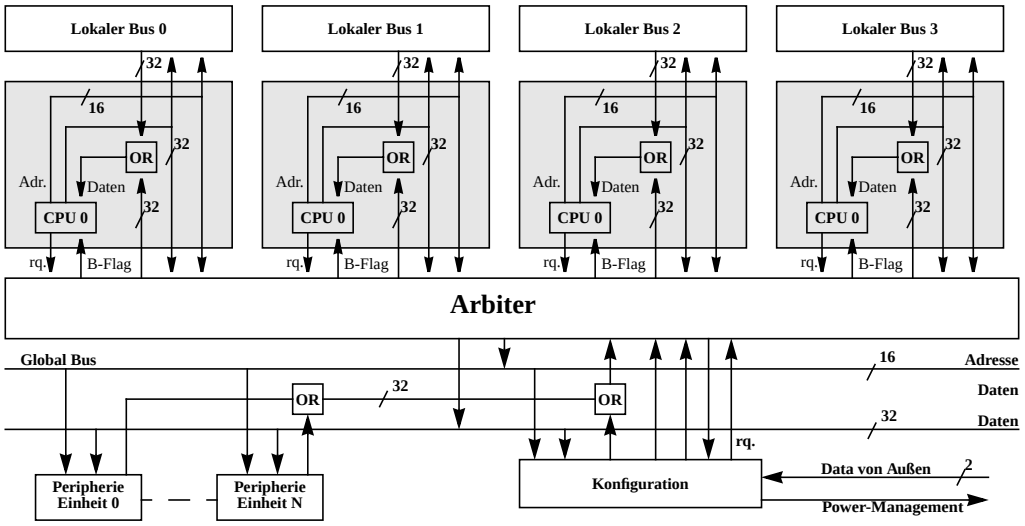


Abb. 10: Schnittstelle der CPU's zum lokalen und globalen Bus

Die obige Abbildung zeigt die vier CPU's mit den lokalen Bussen, die physikalisch mit dem globalen Bus, der zum Arbiter führt, verbunden sind. Der Arbiter trennt die CPU's von den angeschlossenen Peripheriegeräten. Diese können nur Daten empfangen bzw. Daten auf Anforderung senden. Die Konfigurationseinheit kann mit der höchsten Priorität auf den Bus zugreifen.

3.2.3.2 Schnittstelle zum Preprozessor

Ein wesentliches Detail bei der Implementierung des MIMD Prozessors stellt die Registerkopplung zwischen dem Preprozessor und dem MIMD Prozessor dar. Die Schnittstelle besteht aus einem Registersatz, in dem die Daten des Preprozessors akkumuliert werden und nach erfolgter Verarbeitung dem MIMD Prozessor als Subsatz zur Verfügung gestellt werden. Dadurch kann ein wesentlicher Kommunikationsaufwand zwischen den beiden beteiligten Modulen vermieden werden und unmittelbar mit der Verarbeitung der Daten durch die vier CPU's des MIMD Prozessors fortgefahren werden. Die Auswahl wird durch eine Einheit vorgenommen, welche maximal vier der neunzehn Datensätze auswählt (siehe Kap. 6.6). Jeder Datensatz besteht aus acht Werten, wobei jeder CPU je zwei Datensätze zur Verfügung gestellt werden, aus denen die CPU unmittelbar auswählen kann. Die Architektur wird in Kapitel 6 beschrieben, da sie Bestandteil des Preprozessors ist.

3.2.4 Konfiguration des MIMD Prozessors

Dieses Kapitel beschreibt den Aufbau der Konfigurationseinheit, mit der die internen Speicher des MIMD Prozessors und alle Module, die am globalen Bus angeschlossen sind, beschrieben und wieder ausgelesen werden können. Es wird die Architektur beschrieben und auf das Protokoll zur Kommunikation mit der Außenwelt eingegangen. Mit der vorgestellten Netzwerktechnologie lassen sich Ketten (engl. *daisy chains*) aus den oben beschriebenen MIMD Prozessoren bilden. Eine genaue Beschreibung der Architektur kann in [18] nachgelesen werden.

Mit der Konfigurationseinheit wird der MIMD Prozessor und die angeschlossenen Peripheriegeräte initialisiert und bei Bedarf wieder ausgelesen. Außerdem kann der Instruktions- sowie der Datenspeicher konfiguriert werden. Der Zugang erfolgt über den Arbiter des globalen Busses. Alle Geräte die konfiguriert werden sollen, müssen sich im Adressbereich des globalen Busses befinden, nur für den Daten- und den Instruktionsspeicher wird ein besonderer Datenpfad verwendet.

Die Kommunikation erfolgt auf Basis eines Paket basierten Netzwerk Protokolls. Die Übertragung findet seriell und asynchron über LVDS[1] statt.

Nach außen besitzt die Einheit vier Leitungen (zwei Eingänge und zwei Ausgänge), so dass die Prozessoren unabhängig voneinander in einer Kette in zwei Richtungen betrieben werden können. Damit lassen sich Ketten aus MIMD Prozessoren bilden, die von einer externen Einheit mit Daten versorgt werden. Ein FPGA der Altera Excalibur Familie [55] wird als Master verwendet. Die physikalische Verbindung der Prozessoren untereinander erfolgt über LVDS. Demnach verfügt jede Einheit über je zwei Sende- und Empfangszellen. Für den Fall, dass ein Prozessor defekt ist, besteht die Möglichkeit, die Verbindung zu überbrücken, so dass ein fehlerhafter Chip isoliert werden kann und die verbleibenden Chips unbeeinträchtigt weiter betrieben werden können. Falls mehr als ein Prozessor defekt sein sollte, verliert man die Prozessoren zwischen den defekten Chips. Die implementierte Netzwerkstruktur ist in Abbildung 11 [18] dargestellt.

1. **L**ow **V**oltage **D**ifferential **S**ignal

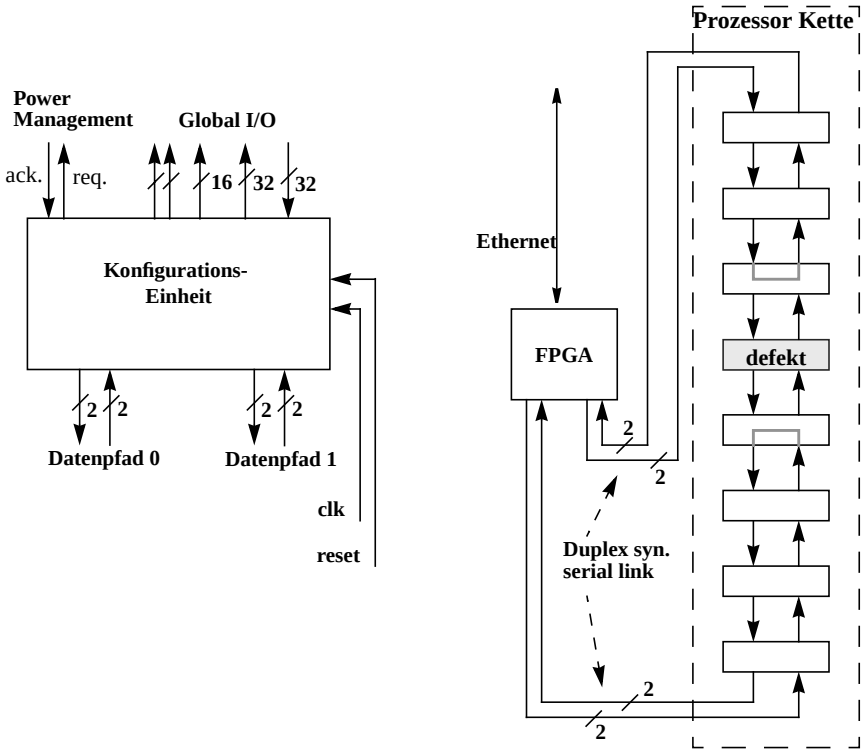


Abb. 11: Konfigurationseinheit und MIMD Prozessoren in einer Daisy Chain

3.2.4.1 Schichtenmodell der Konfigurationseinheit

Die Implementierung der Konfigurationseinheit ist in Anlehnung an die OSI[1] Lagen (Paket basierte Kommunikation) [44] in vier Stufen aufgebaut. Diese sind

- Physical Layer
- Data Link Layer
- Network Layer
- Application Layer

Abbildung 12 [18] zeigt die einzelnen Schichten in einer Übersicht:

1. **O**pen **S**ystem **C**ommunication

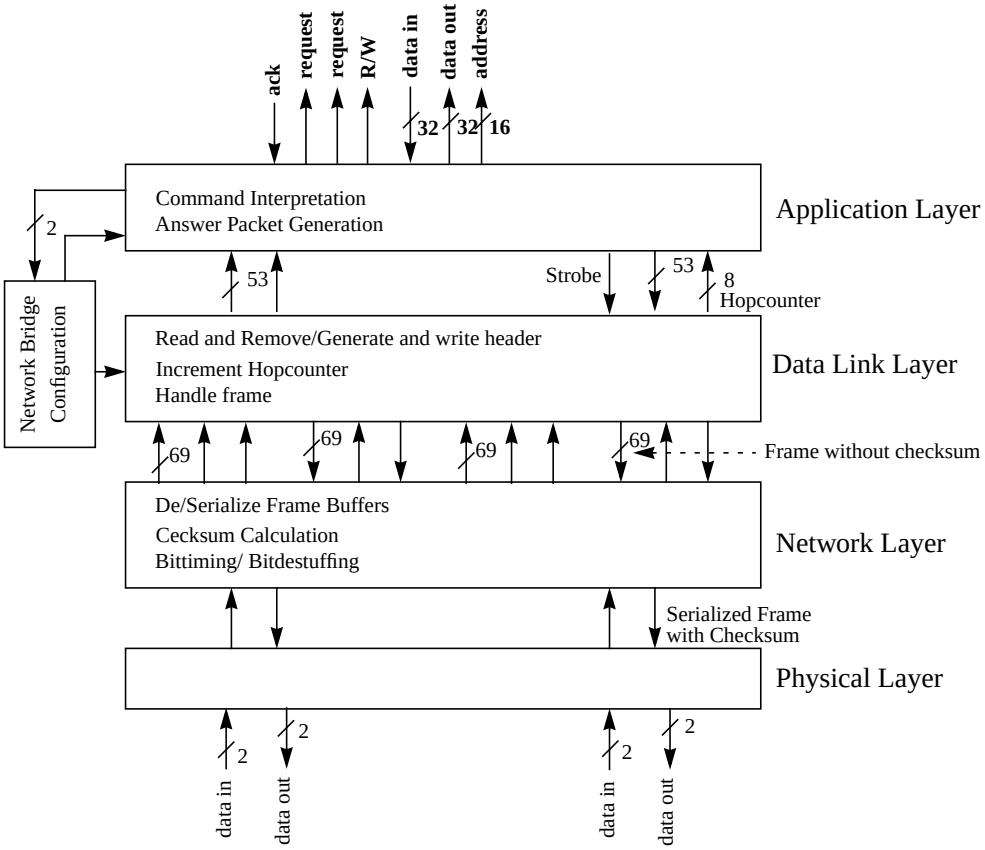


Abb. 12: Schichtenmodell des Netzwerk Interfaces

Der *Physical Layer* führt eine Tiefpass-Filterung der Eingangsdaten durch und synchronisiert die einkommenden Daten mit der internen Taktrate von 120 MHz. Implementiert wird das durch eine konfigurierbare Anzahl an Flip-Flops, deren Ausgänge miteinander XNOR verknüpft sind. Die Synchronisation mit der internen Clock wird durch Flip-Flops erreicht.

Der *Data Link Layer* serellisiert bzw. deserellisiert die einkommenden bzw. ausgehenden Daten (24 MBit/sec) und überprüft die Checksumme, die mit übertragen wird. Außerdem werden redundante Stuff Bits aus dem Datenstrom entfernt bzw. für den ausgehenden Datenstrom werden Stuff Bits eingefügt. Ferner ist im *Data Link Layer* die sog. *Network Brigdes* implementiert, die es ermöglicht, den eingehenden Datenstrom auf den Ausgang zu schalten, was angewendet wird, falls ein Prozessor innerhalb einer Kette fehlerhaft ist und dieser somit ausgeschlossen werden muss.

Der *Network Layer* hat die Aufgabe, den Kopf (engl. *Header*) des Datenpakets zu lesen bzw. zu entfernen und die spezifizierten Aktionen auszuführen. Außerdem wird der *Hop Counter*[1] inkrementiert und es wird entschieden, ob ein Datenpaket intern weiterverarbeitet oder weitergeleitet wird.

Der *Application Layer* führt die Aktionen zum globalen Bus aus, wofür diese Einheit eine Anforderung an den Arbiter stellt und im nächsten Takt auf den Bus die Daten legt bzw. von der selektierten Adresse liest. Ferner kann direkt der Instruktionsspeicher beschrieben werden und im Falle des Datenspeichers kann der Datenpfad von CPU 0 auf die Konfigurationseinheit umgelenkt werden, so dass diese den Datenspeicher beschreiben kann. Dies geschieht, während sich die CPU's im Ruhe-Modus (engl. *idle mode*) befinden, d.h. die Taktsignale der CPU's sind von der Power-Management-Einheit ausgeschaltet.

3.2.4.2 Netzwerkprotokoll der Konfigurationseinheit

Die Kommunikation zwischen dem "Master" und den angeschlossenen Prozessoren erfolgt nach einem Paket basierten Protokoll, womit die einzelnen Chips innerhalb einer Kette einfach angesprochen und konfiguriert werden können. Eine Transaktion kann dabei nur von dem Master eingeleitet werden. Die angeschlossenen Chips empfangen entweder Daten oder senden auf Anforderung Daten aus dem eigenen Speicherbereich.

Jede Transaktion beginnt mit dem Versenden eines Start-Bits auf der Datenleitung. Dadurch wird den Empfängern signalisiert, dass eine Übertragung beginnt. Außerdem wird ein interner Zähler zurückgesetzt, der zur Synchronisation verwendet wird. Dabei werden innerhalb eines Frames insgesamt 86 Bit Daten übertragen. Jede Flanke des übertragenen Datensignals wird zur Resynchronisation verwendet. Falls nicht spätestens nach dem achten Datenbit ein Null-Eins Wechsel aufgetreten ist, wird ein sog. Stuff Bit eingefügt. Dadurch wird spätestens nach acht übertragenen Bits eine Synchronisation erzwungen. Allerdings verringert sich dadurch der Datendurchsatz. Somit variiert die übertragene Datenmenge zwischen 86 und 96 Bits (maximal 10 Stuff Bits). Eine übertragene Sequenz und die korrespondierende Datensequenz ist in Abbildung 13 dargestellt.

1.Der Hop Counter ist eine 8 Bit breite Zahl, die angibt, an wievielen Prozessoren das Paket bisher vorbeigekommen ist.

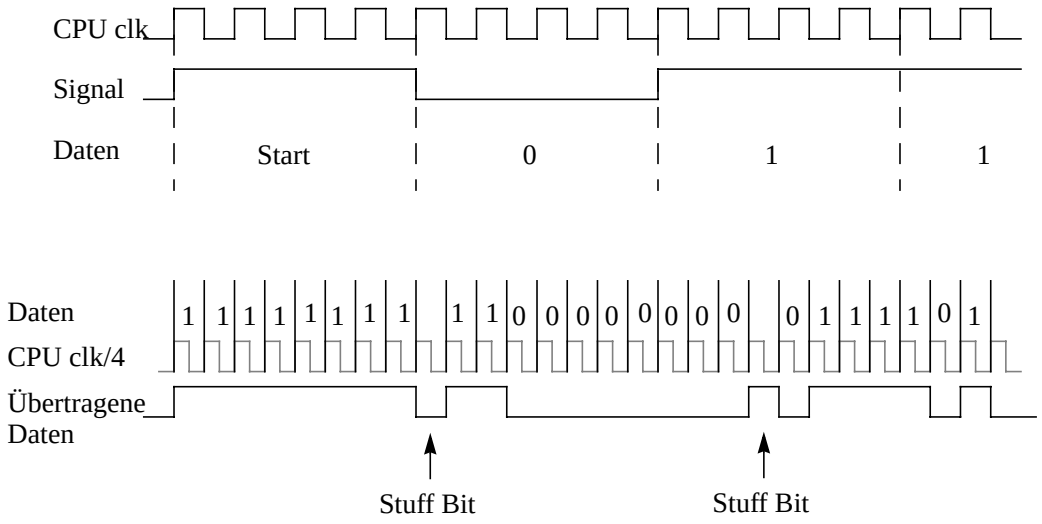


Abb. 13: Protokoll des Netzwerkes

Im oberen Teil der Abbildung ist das Taktverhältnis CPU-Takt zum Wert, der auf dem Netzwerk sichtbar ist, dargestellt. Im unteren Teil ist der Vorgang des Bit Stuffing dargestellt. Findet nicht spätestens nach dem achten Datenbit ein Null-Eins bzw. ein Eins-Null Übergang statt, wird ein Übergang erzwungen und ein Stuff Bit eingefügt.

Die Daten werden nach dem *Cyclic Redundant Code* (CRC) kodiert [37], [44]. Bei dieser zyklischen Redundanzprüfung werden die Bits der empfangenen bzw. zu sendenen Datenfolge als binäre Koeffizienten eines Polynoms aufgefaßt. Dieses Polynom wird durch das sog. Generatorpolynom $G_{(x)}$ dividiert. In diesem Fall wird das Polynom $x^{16}+x^{12}+x^5+1$ benutzt, dass auch als CRC-CCITT bekannt ist und vom Comité Consultatif International Téléphonique et Télégraphique als Norm eingeführt wurde und sich als Industriestandard durchgesetzt hat. Der Rest der Division ist ein Polynom 15. Grades, dessen binäre Koeffizienten als Prüfbits verwendet werden. Beim Versenden der Daten über die Zweidrahtverbindung werden die Prüfbits den eigentlichen Datenbits angehängt. Beim Empfangen der Daten werden die Prüfbits erneut berechnet und mit den empfangenen Daten verglichen. Stimmen die Werte nicht überein lag ein Fehler bei der Übertragung vor. Die Polynomdivision wird durch ein rückgekoppeltes Schieberegister vorgenommen, dessen Rückkopplungen durch die Koeffizienten des Generatorpolynoms festgelegt sind. In Abbildung 14 ist das rückgekoppelte Schieberegister für das oben genannte Polynom skizziert.

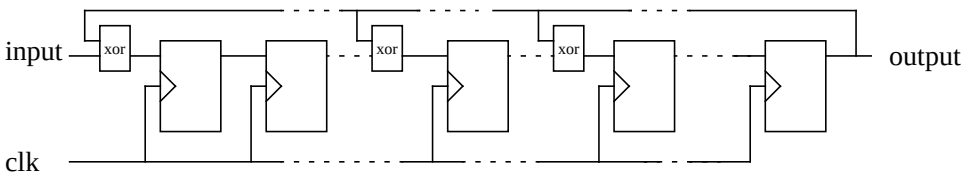


Abb. 14: Hardwarerealisierung der Polynomdivision für das CRC-CCITT

Um effizient mit den angeschlossenen Chips kommunizieren zu können unterstützt das Netzwerkprotokoll verschiedene Kommandos. Ein Netzwerk Error wird immer dann versendet, wenn ein Fehler beim Senden oder Empfangen aufgetreten ist. Der “Master” beginnt dann mit dem Versenden der letzten Transaktion für ein weiteres Mal. Das Setzen bzw. Rücksetzen der Brücke (engl. *Bridge)* bewirkt, dass die eingehenden Daten unverändert auf den zweiten Datenpfad umgeleitet werden, womit defekte Chips innerhalb einer Kette ausgeschlossen werden können. Das NOP Kommando führt keine Aktion durch. Insgesamt unterstützt das Protokoll die in Tabelle 3 aufgeführten Kommandos.

Hex	Binary	Beschreibung
0x0	0000	Netzwerk Error (Checksumme, Bitstuff, timout)
0x1	1000	Lesen
0x2	0100	Schreiben
0x3	1100	Lesen broadcast
0x4	0010	Setzen/Zurücksetzen der Brücke
0x7	1110	NOP
0x8	0001	Lesen - request an Power-Management
0x9	1001	Schreiben - request an Power-Management
0xA	0101	Lesen broadcast - request an Power-Management
0xE	0111	reserviert
0xF	1111	reserviert

Tab. 3: Kommandos innerhalb des Netzwerkprotokolls

Die Daten, die über das Netzwerk übertragen werden, sind in sog. *Frames* verpackt. Ein Datensatz besteht aus insgesamt 86 Bit, wovon 53 Bit für Daten und der Rest zur Steuerung verwendet wird. Dabei signalisiert das erste übertragene Bit den Beginn eines Frames, dann folgt der Datensatz und eine 16 Bit breite Checksumme, die sicherstellt, dass auftretende Übertragungsfehler erkannt werden. Der Datensatz besteht aus einem 16 Bit Header, wobei 7 Bit auf die CRC-Kodierung entfallen, ein Bit, das die Senderichtung angibt und einen 8 Bit breiten *Hop Counter*, der die Anzahl der bisher bei der Übertragung erreichten Stationen zählt. Letztendlich folgt der Datensatz, der in den Speicherbereich der CPU’s geschrieben wird. Dieser besteht aus einem 5 Bit breiten Kommando, einer 16 Bit breiten Adresse sowie 32 Bit Daten. Abbildung 15 zeigt die Struktur eines Frames:

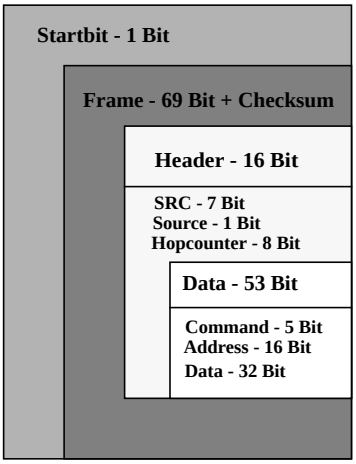


Abb. 15: Aufbau eines Datensatzes

Der Datenspeicher und der Instruktionsspeicher werden über separate Datenpfade beschrieben bzw. wieder ausgelesen. Dafür signalisiert die Konfigurationseinheit der Power-Management-Einheit, dass eine Lese- oder Schreibaktion auf einen der beiden Speichermodule erfolgen soll und schaltet die Datenpfade der ersten CPU um, auf Speicherstellen, die sich im Adressbereich des globalen Busses befinden. Der Schreibport des Instruktionsspeichers muss nicht umgeschaltet werden, da die CPU's nicht auf den Instruktionsspeicher schreiben können. Innerhalb der Konfigurationseinheit, im Speicherbereich des globalen Busses, sind je zwei Register für den Instruktionsspeicher und Datenspeicher vorgesehen. Dabei repräsentieren die unteren 16 Bit die Adresse und die obersten beiden Bits ein Enable (MSB) und das darauf folgende den Lese-Schreibschalter (R/W). Das zweite Register speichert die Daten. Nach erfolgter Anforderung an die Power-Management-Einheit und gesetztem Enable, legt die Einheit die Daten im nächsten Takt an das Speichermodul. Einen Takt später übernimmt der Speicher die Daten. Anschließend wird die Startadresse inkrementiert, so dass im nächsten Takt bereits auf die nächste Speicherstelle geschrieben werden kann (autoincrement). Die Transaktion endet, sobald das Enable Bit im ersten Register von der Konfigurationseinheit zurückgesetzt wird.

4 *Entwicklung einer RISC CPU*

In den folgenden Kapiteln wird die Funktionalität des RISC Prozessors erläutert. Einleitend wird eine Motivation für den Entwurf der Architektur behandelt. Die Beschreibung der Spezifikationen beschränkt sich auf den Befehlssatz, die Adressierungsarten sowie die Befehls- und Datenformate. Diese Merkmale sind nach außen hin sichtbar und prägen somit die Vorgaben, aus denen die Architektur der CPU entwickelt wurde. Auf die interne Struktur wird im zweiten Teil eingegangen.

4.1 Motivation

Der Grund für die Entwicklung einer neuen Prozessor-Architektur liegt in den speziellen Anforderungen, die sich aus der primären Anwendung ergeben (siehe Kap. 5) und der Kopplung der CPU’s zu dem in Kap. 3 beschriebenen MIMD Prozessor. Daraus lassen sich die folgenden Anforderungen ableiten:

- Kompakte Architektur
- Flache Pipeline
- Schnelle Arithmetik
- Konfigurierbare Schnittstellen
- Erweiterbarkeit

Dementsprechend wird eine Harvard-Architektur entwickelt, welche die oben beschriebenen Eigenschaften besitzt. Integraler Bestandteil ist der minimale Hardwareaufwand bei einem 32 Bit breiten Datenpfad mit flacher Pipeline.

4.2 Befehls- und Datenwortbreite

Das Befehlsformat der CPU basiert auf 24 Bit breiten Worten. Auftretende Operanden sind in die Instruktion kodiert und haben eine feste Länge. Die binäre Darstellung der Opcodes kann Tabelle 4 entnommen werden. Die verschiedenen Befehlsformate sind in Abbildung 16 dargestellt. Dabei ist zu erkennen, dass die meisten Befehle 3-Adress-Befehle sind, also die Adresse der ersten und zweiten Operanden, sowie die Adresse des Resultats in die Instruktion kodiert ist. Es kommen aber auch 2- 1- und 0-Adress-Befehle vor. Das implementierte Befehlsformat bietet genug Raum für zukünftige Erweiterungen. Die Kodierung der Befehle ist einer möglichst einfachen Decodierung, bei gleichzeitig kurzer Wortbreite gefolgt.

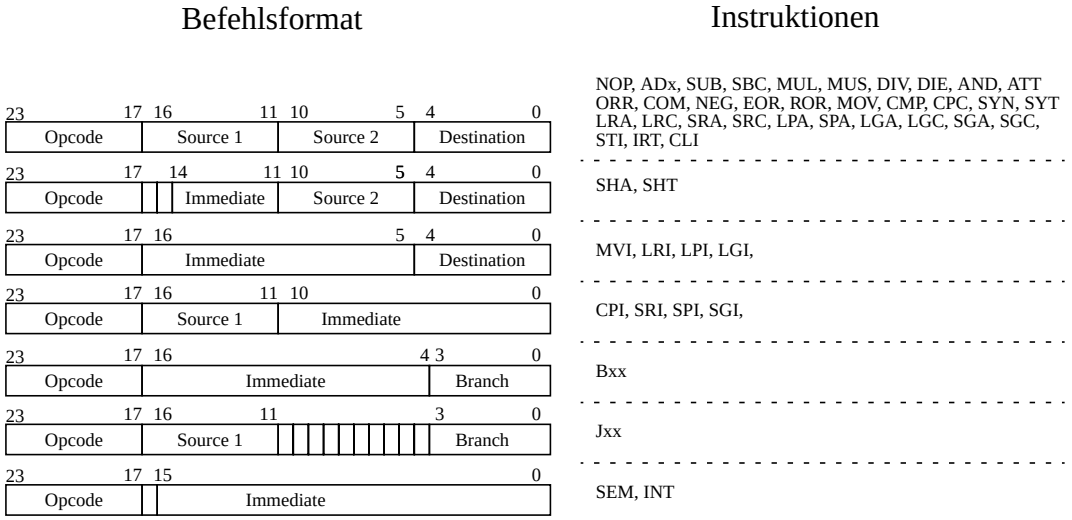


Abb. 16: Befehlsformate und Instruktionen

Im rechten Teil ist dargestellt, welche Befehle welches Befehlsformat verwenden

Das Datenformat der CPU basiert auf 32 Bit breiten Worten. Die Reihenfolge, in der die Datenwörter des Typs Long (64 Bit) abgelegt werden, dass bei der Multiplikation vorkommt bzw. dass bei der Division verwendet wird, wird folgendermaßen festgelegt: Die oberen 32 Bit der Multiplikation werden im privaten Register File (PRF) an der Speicherstelle 0x0 abgelegt. Für die Division kommen ebenfalls die oberen 32 Bit des Dividenden aus diesem Register.

4.3 Instruktionssatz

Der Befehlssatz bestimmt die Größe und Mächtigkeit des Befehlsvorrats, der dem Programmierer zur Verfügung steht. Der Befehlssatz der CPU besteht aus 72 Maschinenbefehlen, die sich in sieben Gruppen einteilen lassen. Dabei unterstützt die CPU alle RISC-ähnlichen Instruktionen auf Festkommadaten. Die nunmehr implementierten Opcodes sind in Tabelle 4 dargestellt:

#	Opcode	Format	Op 1	Op 2	Op 3	Wirkung
0	NOP	0000 0000 0000 0000 0000 0000	-	-	-	No Operation
1	ADD	1000 001s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Addition
2	ADC	1000 001s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Addition mit Übertrag
3	SUB	1000 011s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Subtraktion
4	SBC	1000 111s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Subtraktion mit Übertrag
5	MUL	1001 001s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Multiplikation
6	MUS	1001 011s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Multiplikation
7	DIV	1001 100s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	-	Division
8	DIE	1001 1110 0000 0000 000d dddd	-		PRF, GRF	Ergebnis der Division sp.
9	AND	1010 011s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Logisches UND
10	ATT	1010 010s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	-	Logisches UND ohne WB

Tab. 4: Instruktionssatz der CPU

11	ORR	1010 101s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Logisches ODER
12	COM	1011 1110 0000 0ttt tttd dddd	-	GRF, PRF, FitReg, Const	PRF, GRF	Zweierkomplement
13	NEG	1010 1110 0000 0ttt tttd dddd	-	GRF, PRF, FitReg, Const	PRF, GRF	Einerkomplement
14	EOR	1010 001s ssss sttt tttd dddd	PRF, FitReg	GRF, PRF, FitReg, Const	PRF, GRF	Logisches Exklusiv ODER
15	SHA	1011 0110 0iii ittt tttd dddd	implicit	GRF, PRF, FitReg, Const	PRF, GRF	Arithmetisches Schieben
16	SHT	1011 0010 0iii ittt tttd dddd	implicit	GRF, PRF, FitReg, Const	PRF, GRF	Logisches Schieben
17	ROR	1011 1010 0000 0ttt tttd dddd	-	GRF, PRF, FitReg, Const	PRF, GRF	Rechts Rotieren
18	MOV	1100 0010 0000 0ttt tttd dddd	-	GRF, PRF, FitReg, Const	PRF, GRF	Verschieben
19	MVI	1100 011i iiii iiii iiid dddd	-	implicit	PRF, GRF	Verschieben implizit
20	CMP	1000 100s ssss sttt ttt0 0000	PRF, FitReg	GRF, PRF, FitReg, Const	-	Vergleichen
21	CPI	1100 100s ssss siii iiii iiii	PRF, FitReg	implicit	-	Vergleichen implizit
22	CPC	1000 110s ssss sttt ttt0 0000	PRF,	GRF, PRF, FitReg, Const	-	Vergleichen mit Übertrag
23	BSS	0000 010k kkkk kkkk kkk0 0100	-	implicit	-	Sprung implizit
24	JSS	0000 100s ssss s000 0000 1100	PRF,	-	-	Sprung Register relativ
25	BSC	0000 010k kkkk kkkk kkk0 0100	-	implicit	-	Sprung implizit
26	JSC	0000 100s ssss s000 0000 0100	PRF,	-	-	Sprung Register relativ
27	BZS	0000 010k kkkk kkkk kkk0 1001	-	implicit	-	Sprung implizit
28	JZS	0000 100s ssss s000 0000 1001	PRF,	-	-	Sprung Register relativ
29	BZC	0000 010k kkkk kkkk kkk0 0001	-	implicit	-	Sprung implizit
30	JZC	0000 100s ssss s000 0000 0001	PRF,	-	-	Sprung Register relativ
31	BVS	0000 010k kkkk kkkk kkk0 1011	-	implicit	-	Sprung implizit
32	JVS	0000 100s ssss s000 0000 1011	PRF,	-	-	Sprung Register relativ
33	BVC	0000 010k kkkk kkkk kkk0 0011	-	implicit	-	Sprung implizit
34	JVC	0000 100s ssss s000 0000 0011	PRF,	-	-	Sprung Register relativ
35	BNS	0000 010k kkkk kkkk kkk0 1010	-	implicit	-	Sprung implizit
36	JNS	0000 100s ssss s000 0000 1010	PRF,	-	-	Sprung Register relativ
37	BNC	0000 010k kkkk kkkk kkk0 0010	-	implicit	-	Sprung implizit
38	JNC	0000 100s ssss s000 0000 0010	PRF,	-	-	Sprung Register relativ
39	BCS	0000 010k kkkk kkkk kkk0 1000	-	implicit	-	Sprung implizit
40	JCS	0000 100s ssss s000 0000 1000	PRF,	-	-	Sprung Register relativ
41	BCC	0000 010k kkkk kkkk kkk0 1000	-	implicit	-	Sprung implizit
42	JCC	0000 100s ssss s000 0000 0000	PRF,	-	-	Sprung Register relativ
43	BBS	0000 010k kkkk kkkk kkk0 1101	-	implicit	-	Sprung implizit
44	JBS	0000 100s ssss s000 0000 1101	PRF,	-	-	Sprung Register relativ
45	BBC	0000 010k kkkk kkkk kkk0 0101	-	implicit	-	Sprung implizit
46	JBC	0000 100s ssss s000 0000 0000	PRF,	-	-	Sprung Register relativ
47	BRA	0000 010k kkkk kkkk kkk0 1111	-	implicit	-	Sprung implizit
48	JMP	0000 100s ssss s000 0000 1111	PRF,	-	-	unbedingter Sprung
49	SYN	0000 1100 0000 0000 0000 0000	-	-	-	Synchronisation
50	SYT	1100 1110 0000 0000 000d dddd	-	-	PRF, GRF	Lese Syn Register
51	SEM	0001 0000 iiii iiii iiii iiii	-	implicit	-	Setze Syn-Maske
52	LRA	1101 0010 0000 0ttt tttd dddd	-	PRF	PRF, GRF	Lade aus DMEM
53	LRC	1111 0010 0000 0000 000d dddd	-	implicit	PRF, GRF	Lade aus DMEM postinc.
54	LRI	1101 0110 kkkk kkkk kkkd dddd	-	implicit	PRF, GRF	Lade aus DMEM implizit
55	SRA	0001 010s ssss sttt ttt0 0000	PRF,	PRF	-	Speichere in DMEM
56	SRC	0011 100s ssss s000 0000 0000	PRF,	implicit	-	Speichere in DMEM post
57	SRI	0001 100s ssss skkk kkkk kkkk	PRF,	-	PRF, GRF	Speichere in DMEM implizit
58	LPA	1110 0010 0000 0ttt tttd dddd	-	PRF		Lade vom privaten Bus
59	LPI	1110 0010 0000 0ttt tttd dddd	-	implicit		Lade von privaten Bus implizit
60	SPA	0001 110s ssss sttt ttt0 0000	PRF,	PRF		Speichere in privaten Bus
61	SPI	0010 000s ssss skkk kkkk kkkk	PRF,	-		Speichere im privaten Bus
62	LGA	1110 1010 0000 0ttt tttd dddd	-	PRF		Lade aus globalen Bus
63	LGC	1111 0110 0000 0000 000d dddd	-	implicit		Lade aus globalen Bus
64	LGI	1110 1110 kkkk kkkk kkkd dddd	-	implicit		Lade aus globalen Bus
65	SGA	0010 010s ssss sttt ttt0 0000	PRF,	PRF		Speichere im globalen Bus
66	SGC	0011 110s ssss s000 0000 0000	PRF,	implicit		Speichere im globalen Bus
67	SGI	0010 100s ssss skkk kkkk kkkk	PRF,	-		Speichere im globalen Bus
68	CLI	0010 1100 0000 0000 0000 0000	-	-		Clear Interrupt
69	STI	0011 0000 0000 0000 0000 0000	-	-		Store Interrupt

Tab. 4: Instruktionssatz der CPU

70	INT	0011 0010 iiii iiii iiii iiii	-	implicit		Software Interrupt
71	IRT	0011 0100 0000 0000 0000 0000	-	-		Rücksprung aus Interrupt

Tab. 4: Instruktionssatz der CPU

Die zweite Spalte zeigt die mnemonische Darstellung der Instruktion, wie sie auch von dem entwickelten Assembler (siehe Kap. 7.1) verwendet wird. Die dritte Spalte zeigt die Binärdarstellung. Die dargestellten Buchstaben kennzeichnen die verwendeten Adressfelder. Dabei steht ‘s’ für die erste Quelle einer Operation (*Source 1*), der Buchstabe ‘t’ kennzeichnet die zweite Quelle (*Source 2*), ‘d’ wird zur Kodierung des Zielregisters verwendet (*Destination*) und ‘i’ zeigt an, dass Daten direkt in die Instruktion kodiert wurden. Der Buchstabe ‘k’ bedeutet, dass eine Adresse Bestandteil der Instruktion ist, was von Sprung-, Lade- und Speicherinstruktionen verwendet wird.

4.4 Adressraum und Adressierungsarten

Der Adressraum entspricht der Anzahl an Speicherwörtern, die ein Prozessor adressieren kann. Der Adressraum des Datenspeichers und des globalen Busses ist 16 Bit breit. Somit können 65536 Worte adressiert werden. Dabei unterstützt der Prozessor fünf verschiedene Adressierungsarten die zum Laden und Speichern von Daten aus den Datenspeichern verwendet werden. Unter Adressierungsarten werden jene Verfahren verstanden, über die ein Maschinenbefehl ihre Operanden referenziert.

Es wird immer auf den Datenspeicher, den globalen oder den lokalen Adressraum referenziert. Ist ein Lese- bzw. Schreibzyklus notwendig, wird, dem Lade- bzw. Speicherverfahren entsprechend, entweder direkt adressiert oder relativ zu einem Register des PRF’s. Soll auf den globalen Bus zugegriffen werden, wird dies über einen Arbiter getan, der nach Prioritäten geordnet den Bus zuteilt. Ein erfolgloser Zugriff wird der CPU über ein spezielles Flag angezeigt. Der Prozessor unterstützt die in den folgenden Abschnitten dargestellten Adressierungsarten [3].

4.4.1 Unmittelbare Adressierung

Bei dieser Adressierungsart ist der Operand eine Konstante, die Teil der Instruktion ist. Sie wird verwendet, um Konstanten in spezielle Register des Prozessors zu verschieben. Beispiele sind die Instruktionen *SEM* und *INT*. Bei ihnen wird das Argument, eine 16 Bit breite Zahl, übergeben. Dabei handelt es sich entweder um einen Interrupt-Vektor oder, im Falle von *SEM*, um den Vektor zur Synchronisation der Prozessoren.

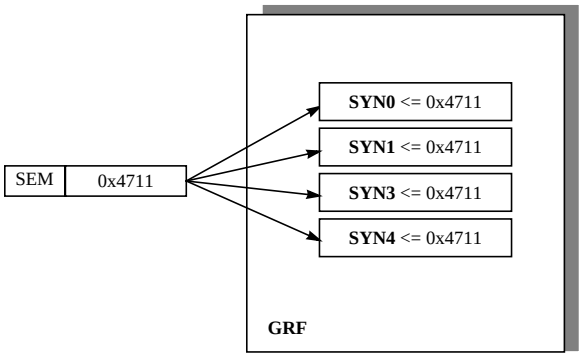


Abb. 17: Unmittelbare Adressierung

Als Beispiel dient die Instruktion *SEM*. Hier wird das 16 Bit breite Argument in die Synchronisationsregister der CPU's transferiert.

4.4.2 Absolute Adressierung

Das Adressierungsverfahren wird verwendet, wenn die effektive Adresse des Operanden als absolute Adresse Bestandteil der Instruktion ist. Sie wird bei allen Lade- und Speicherbefehlen verwendet, die direkt das angesprochene Datenwort referenziert.

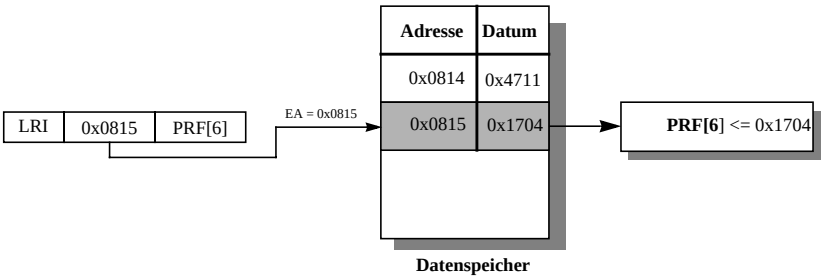


Abb. 18: Absolute Adressierung

Als Beispiel dient die logische Operation *LRI*. Sie lädt das durch die Adresse referenzierte Datenwort aus dem Speicher direkt in den Registersatz der CPU.

4.4.3 Registeradressierung

Bei dem Adressierungsverfahren befindet sich der Operand in einem CPU Register. Die Adresse steht als Registeradresse im Opcode.

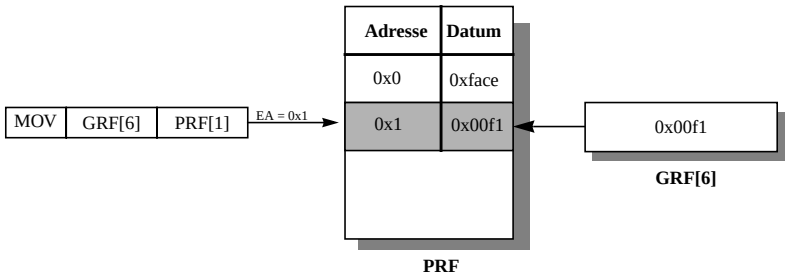


Abb. 19: Registeradressierung

Als Beispiel dient der Verschiebebefehl *MOV*, der das referenzierte Wort, das sich in einem CPU Register befindet an die Stelle eines CPU Registers schreibt, das durch die Adresse des Opcodes ausgewählt wurde.

4.4.4 Registerindirekte Adressierung

Bei dieser Adressierungsart steht die effektive Adresse in einem Register und der Operand im Speicher. Das Register, welches die Adresse enthält, wird als Zeiger verwendet. Diese Adressierungsart wird eingesetzt, wenn innerhalb eines Programmteils häufig auf dieselbe Adresse zugegriffen wird.

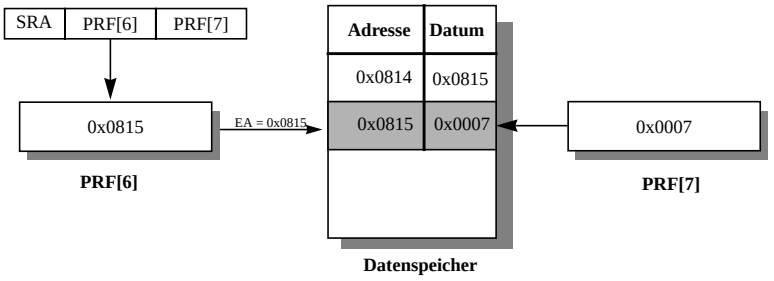


Abb. 20: Registerindirekte Adressierung

Als Beispiel dient der Speicherbefehl *SRA*, der das referenzierte Wort, das sich in einem CPU Register befindet an die Stelle des Datenspeichers schreibt, das durch den Inhalt des CPU Registers referenziert wird und durch die Adresse des Opcodes ausgewählt wurde.

4.4.5 Registerindirekte Adressierung mit Postinkrement

Dieses Adressierungsverfahren eignet sich besonders für die Bearbeitung von Datenfeldern in einer Schleife und wird für Lade-/Speicher-Instruktionen verwendet, die auf den Datenspeicher oder den externen Datenbus zugreifen.

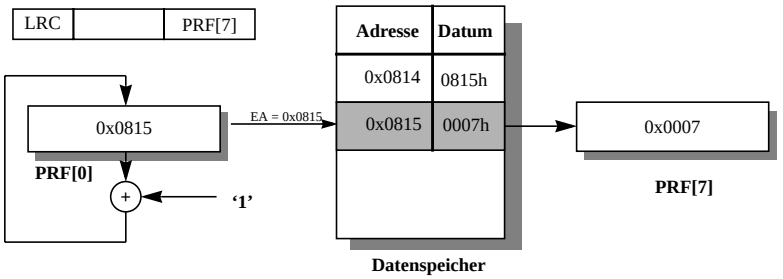


Abb. 21: Registerindirekte Adressierung mit Postinkrement

Als Beispiel dient die Instruktion *LRC*, mit der Werte aus dem globalen Speicherbereich in den Registersatz des Prozessors transferiert werden. Anschließend wird der Wert der Adresse um den Wert eins inkrementiert. Bei dieser Adressierungsart steht implizit fest, dass sich die Adresse in dem ersten Register des privaten Registersatzes befindet.

4.5 Die Architektur einer CPU

In diesem Kapitel wird die Hardwarestruktur der RISC CPU dargestellt. Dabei ist das Augenmerk auf eine möglichst kompakte Architektur gerichtet, wobei die Verarbeitungsgeschwindigkeit und der Datendurchsatz die weiteren Entwurfskriterien darstellen. Dementsprechend wird der MIMD Prozessor aus vier RISC Prozessor Kernen aufgebaut, die über eine zweistufige Pipeline verfügen. Datenspeicher und Instruktionsspeicher werden als Vier-Port-Speicher implementiert, was sicherstellt, dass alle CPU’s gleichzeitig auf den Daten- sowie den Instruktionsspeicher zugreifen können.

Eine CPU des MIMD Prozessors besteht im Wesentlichen aus zwei Pipelinestufen, um der Zielvorgabe einer minimalen Taktrate von 120 MHz bei einer 0,18 µm CMOS Technologie gerecht zu werden. Gegenüber einer Einzyklenarchitektur kann eine Steigerung des Datendurchsatzes um den Faktor zwei erzielt werden. Die flache Pipeline ermöglicht die schnelle Verarbeitung der Daten trotz der geringen Taktfrequenz. Ferner können Datenabhängigkeiten vermieden werden.

Die erste Stufe lädt die Instruktion aus dem Instruktionsspeicher, decodiert die aktuelle Instruktion und stellt die Steuerwörter der zweiten Stufe zur Verfügung. Die zweite Stufe holt die Daten aus den referenzierten Datenspeichern, führt Operationen auf diesen durch und schreibt die Ergebnisse wieder in einen Datenspeicher zurück. ALU-Operationen werden zwischen Registern ausgeführt, d.h. bei einer Operation, in der zwei Werte durch eine ALU Operation miteinander verknüpft werden, kommen die Daten grundsätzlich aus einem Register und werden auch wieder in ein Register zurückgeschrieben. Der Zugriff auf den Datenspeicher oder dem angeschlossenen Bus erfolgt über Lade- und Speicherbefehle. Einen Überblick über die Architektur liefert Abbildung 22:

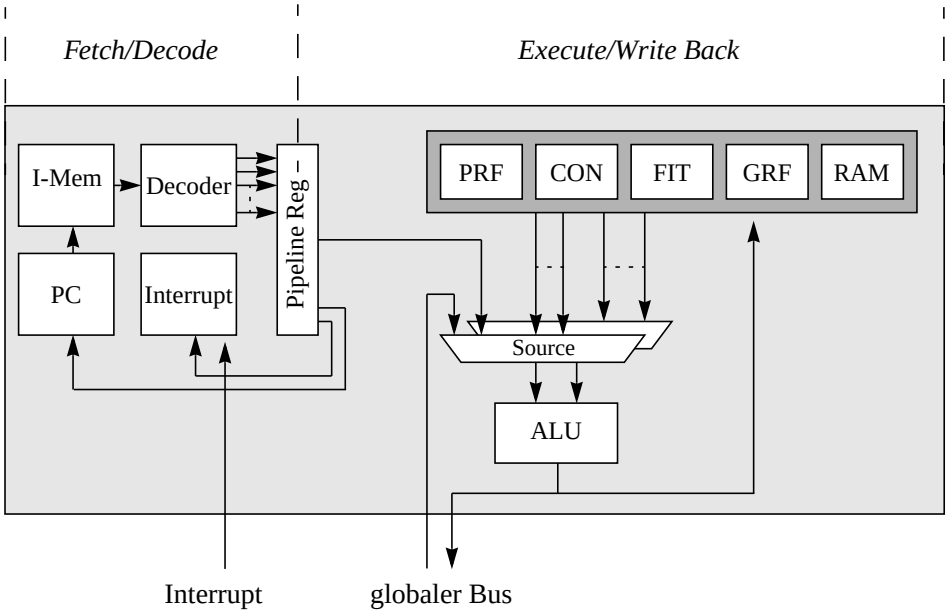


Abb. 22: Architektur einer CPU
Alle dargestellten Datenpfade sind 32 Bit breit

4.5.1 Die erste Pipelinestufe

Die erste Pipelinestufe hat die Funktion, Opcodes aus dem Instruktionsspeicher zu laden und zu decodieren sowie dem Opcode entsprechend Instruktionswörter zu generieren. Die PC-Logik liefert die aktuelle Adresse des Programmzählers und berechnet nach erfolgtem Fetch-Zyklus die Folgeadresse der Instruktion. Im Falle eines Interrupts bzw. eines Sprunges wird die gegenwärtige Adresse gerettet und durch die Sprungadresse bzw. Adresse der Interrupt-Vektor-Tabelle (IVT) des Interrupt Controllers ersetzt.

In der ersten Stufe befinden sich der Programmzähler (PC), die Dekodiereinheit und der Instruktionsspeicher, wobei dieser zusätzlich den drei weiteren CPU's zur Verfügung steht.

4.5.1.1 Der Programmzähler

Im Programmzähler wird der Wert der aktuellen Programmadresse gehalten und verwaltet. Die implementierten Register besitzen eine Breite von 16 Bit. Bei fortlaufender Programmfolge wird der Wert des PC-Registers um den Wert eins erhöht. Im Falle von Programmsprüngen oder Interrupts wird der gegenwärtige Wert durch die Sprungadresse bzw. durch die Startadresse der Interrupt Prozedur ersetzt. Zusätzlich wird der aktuelle Wert in einem Register zwischengespeichert, so dass beim Rücksprung der Programmverlauf fortgesetzt werden kann.

Eine Instruktion hat eine feste Länge von 24 Bit und belegt im Instruktionsspeicher genau eine Zeile. Befehle, die für Programmsprünge verantwortlich sind, bestehen aus dem Sprungbefehl

selbst und der Sprungadresse, sowie im Falle von registerrelativen Sprüngen, aus der Adresse des Registers, in dem sich die Sprungadresse befindet. Die erste Pipelinestufe detektiert, dass es sich um eine Sprunginstruktion handelt und stellt ggf. der zweiten Stufe die Sprungadresse zur Verfügung. Dort werden die selektierten Flags ausgewertet, ob der Sprung ausgeführt werden soll. Dann wird die Sprungadresse an den Programmzähler übergeben und der Sprung wird ausgeführt. Ist die Sprungbedingung nicht erfüllt, wird die sequentielle Programmabarbeitung fortgesetzt. Der PC-Logik wird über ein Steuersignal angezeigt, ob anstatt des inkrementierten Programmzählers die Sprungadresse in dem Programmzähler gespeichert wurde oder, im Falle eines Interrupts, die Adresse der Interrupt Prozedur. Die möglichen Varianten sind in Tabelle 5 dargestellt:

Programmverlauf	Neuer PC-Wert	Erklärung
Sequentiell	PC = PC +1	PC inkrementieren
bedingte Sprünge	PC = Branch-Addr	PC durch Sprungadresse ersetzen
unbedingte Sprünge	PC = Branch-Addr	PC durch Sprungadresse ersetzen
Register-relativer Sprung	PC = PRF[Addr]	PC durch Sprungadresse ersetzen, dabei die unteren 16 Bit
Interrupt	PC = IVT	PC durch Startadresse der Interrupt Prozedur ersetzen
Rücksprung aus Interrupt	PC = tempaddr	PC durch gespeicherte Rücksprungadresse ersetzen

Tab. 5: Änderung des Programmzählers

4.5.1.2 Datenpfad und Befehlsdecoder

Nach dem Laden der Instruktion aus dem Instruktionsspeicher erfolgt die Decodierung durch den Befehlsdecoder. Dieser generiert Steuerwörter, die von der zweiten Stufe verarbeitet werden. Die während eines Fetch-Zyklusses geladenen bzw. erzeugten Daten werden in die Pipelineregister der ersten Stufe geschrieben. Parallel zu jedem Ladezyklus weist die Kontrolleinheit den Programmzähler an, den aktuellen Wert des PC-Registers zu inkrementieren bzw. im Falle eines Sprunges oder Interrupt, die neuen Werte in den Programmzähler zu übernehmen.

In klassischen Pipelinearchitekturen ist für die Dekodierlogik eine separate Pipelinestufe vorgesehen. Bei dem Entwurf der Prozessorarchitektur konnte davon ausgegengen werden, dass die Zugriffszeit des integrierten Instruktionsspeichers hinreichend klein ist und deshalb der Befehlsdecoder in die erste Stufe integriert werden kann. Die Zugriffszeit des implementierten Speichers beträgt etwa 5,5 ns [61].

Für das Laden von Instruktionen ergibt sich folgender Verlauf:

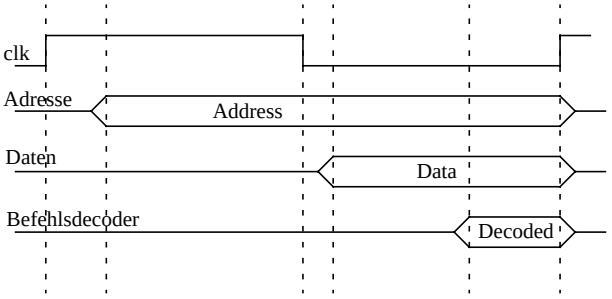


Abb. 23: Verlauf der Instruktionsverarbeitung innerhalb der ersten Stufe

Der Ladezyklus beginnt mit einer Ladeaufforderung an den Instruktionsspeicher, dafür wird die aktuelle Adresse des PC-Registers an den Speicher übergeben. Der Speicher übernimmt die Adresse und übergibt das angeforderte Datum an den Befehlsdecoder, der die generierten Steuerwörter in insgesamt 23 Pipelineregister schiebt. Anschließend wird bei sequentiellem Programmverlauf zur nächsten Instruktion übergegangen.

Der Decoder besteht aus kombinatorischer Logik mit kurzer Signallaufzeit. Am Ende des Taktes stehen die generierten Steuerwörter zur Verfügung und es kann damit begonnen werden, die nächste Instruktion zu laden. Somit kann mit jedem Takt eine Instruktion aus dem Speicher geladen und dekodiert werden.

Der Datenpfad der ersten Pipelinestufe beinhaltet die Pipelineregister für die Steuerwörter des Befehlsdecoders, sowie ggf. die zugehörigen Operanden. Wurde ein Sprung bzw. ein Interrupt detektiert, wird der nachfolgende Befehl durch einen NOP ersetzt, insofern die Sprungbedingung erfüllt ist, d.h. die Instruktion nach dem Sprungbefehl wird nicht ausgeführt.

Der Befehlsdecoder generiert abhängig vom geladenen Opcode mehrere Steuerwörter, welche innerhalb der zweiten Pipelinestufe ausgewertet werden. Tabelle 6 zeigt die generierten Steuerwörter und die Funktion.

Steuerwort	Breite	Erklärung
sl1	4 Bit	Auswahl der Speicherzeile innerhalb des ausgewählten Registerblocks (Op. 1)
sl2	4 Bit	Auswahl der Speicherzeile innerhalb des ausgewählten Registerblocks (Op. 2)
dl	4 Bit	Auswahl der Speicherzeile innerhalb des ausgewählten Registerblocks (Op. 3)
sb1	3 Bit	Auswahl des Registerblocks (Op.1)
sb2	3 Bit	Auswahl des Registerblocks (Op.2)
sb3	2 Bit	Auswahl des Registerblocks (Op.3)
db	1 Bit	Auswahl des Registerblocks (Op.3)
gate	1 Bit	Anhalten der weiteren Verarbeitung
wb	1 Bit	Write Back; Auswahl, ob Daten zurückgeschrieben werden
branch_sel	1 Bit	Auswahl der Adressquelle für Sprung
branch	1 Bit	Instruktion ist Sprunginstruktion
no_flags	1 Bit	Flags der ALU bleiben bei Instruktion unverändert

Tab. 6: Instruktionswörter des Befehlsdecoders

Steuerwort	Breite	Erklärung
branch_op	4 Bit	Auswahl der Flags, die für die Sprungbedingung ausgewertet werden
immediate	32 Bit	Implizierter vorzeichenerweiterter Wert
branch_pc	16 Bit	Vorzeichenerweiterte Sprungadresse
mem_addr	16 Bit	Vorzeichenerweiterte Speicheradresse
sras	1 Bit	Auswahl der Adressquelle für den Datenspeicher
sioas	1 Bit	Auswahl der Adressquelle für den globalen Bus
request	1 Bit	Flag signalisiert die Anforderung, den globalen Bus zu benutzen
alu_op	4 Bit	Instruktion ist ALU Operation
cli_sti_irt	2 Bit	Steuerwörter zur Verwaltung von Interrupts
int_req_soft	16 Bit	Interruptvektor aus Instruktion
we_prf0	1 Bit	Write Enable auf Zeile 0 des Privaten Register File (PRF)

Tab. 6: Instruktionswörter des Befehlsdecoders

Die Steuerwörter *sl1, sl2* und *dl* selektieren die Speicherzeile innerhalb der Registerbänke. Dabei spricht *sl1* die Blöcke PRF und FIT, *sl2* die Blöcke PRF, GRF, CONST und FIT und *dl* die Blöcke GRF und PRF an. Die beiden Erstgenannten selektieren die Quelloperanden, die anschließend in der ALU miteinander assoziiert werden. Das Wort *dl* selektiert die Zeile innerhalb des Blocks PRF oder GRF, in dem das Resultat der Operation gespeichert werden soll. Die Werte *sb1, sb2, sb3* und *db* selektieren die Blöcke, aus welchen die Werte stammen, die verknüpft werden bzw. den Block, in dem die Werte zurückgeschrieben werden sollen. Dafür kommen PRF, GRF, CONST und FIT in Frage. Das Signal *gate* wird verwendet, um die weitere Verarbeitung zu unterbrechen. Es wird gesetzt, falls der Befehl *SYN* dekodiert wurde, der anzeigt, dass der Programmzähler suspendiert werden soll. Das Signal wird an den Programmzähler übergeben, der im nächsten Takt die Verarbeitung unterbricht, insofern die Synchronisationsbedingung erfüllt ist. Das Flag *wb* unterscheidet, ob Resultate in einen Registersatz zurückgeschrieben werden oder nicht. Die gleiche Funktion hat das Signal *branch*, das anzeigt, dass es sich um eine Sprunginstruktion handelt. Das Steuerwort *branch_op* kodiert binär die Flags der CPU, auf die bedingte Sprünge ausgeführt werden können. Die Auswertung der ALU Flags erfolgt aufgrund der übergebenen Sprungflaggen, die unverändert aus der Instruktion übernommen werden. Die Quelle der Sprungadresse wird über das Signal *branch_sel* angezeigt. Mögliche Quellen sind entweder das PRF oder „Immediate“, was heißt, dass die Sprungadresse direkt aus der Instruktion stammt. Das Signal *no_flags* zeigt explizit an, dass die Flags der ALU bei der Verarbeitung der gegenwärtigen Instruktion unverändert bleiben sollen. Das ist notwendig, da grundsätzlich alle Daten durch die ALU geführt werden, bevor sie gespeichert werden, was einen einheitlichen Datenpfad schafft. Die Signale *mem_addr* und *branch_pc* sind Adressen, die Bestandteil der Instruktion sind und an die zweite Pipelinestufe übergeben werden, um entweder als vorzeichenerweiterte Lade- oder Speicheradresse verwendet zu werden oder den neuen Wert des Programmzählers darzustellen. Die Werte *sras* und *sioas* selektieren die Quelle der Adresse für den Datenspeicher bzw. den globalen Bus. Vorher wird durch das *request* Flag dem Arbiter angezeigt, dass die CPU auf den globalen Bus zugreifen will. Dieses Signal wird als einziges nicht in Pipelineregister zwischengespeichert, da die Entscheidung, welche CPU den Bus erhält, bereits im gegenwärtigen Takt getroffen werden muss. Das Signal *alu_op* enthält binär kodiert die ALU Operation, die durch die gegenwärtige Instruktion ausgeführt werden soll. Das Steuerwort *cli_sti_irt* kann drei verschiedene Werte annehmen und steuert das Verhalten des Interrupt Controllers. Es zeigt dem Controller an, wann die Interrupt Prozedur beendet ist (*irt*), wann Interrupts mit niedriger Priorität verboten sind (*cli*) und wann Interrupts mit niedriger Priorität wieder erlaubt sind (*sti*). Der Wert *int_req_soft* ist das Argument des Software Interrupts und wird direkt aus der Instruktion

extrahiert und dient dem Interrupt Controller als Eingang. Mit dieser Instruktion können gleichzeitig mehrere Interrupts ausgeführt werden. Das Signal *we_prf0* wird gesetzt, falls das erste Register des PRF beschrieben werden soll.

Der Aufbau des Datenpfades der ersten Pipelinestufe ist in Abbildung 24 schematisch dargestellt:

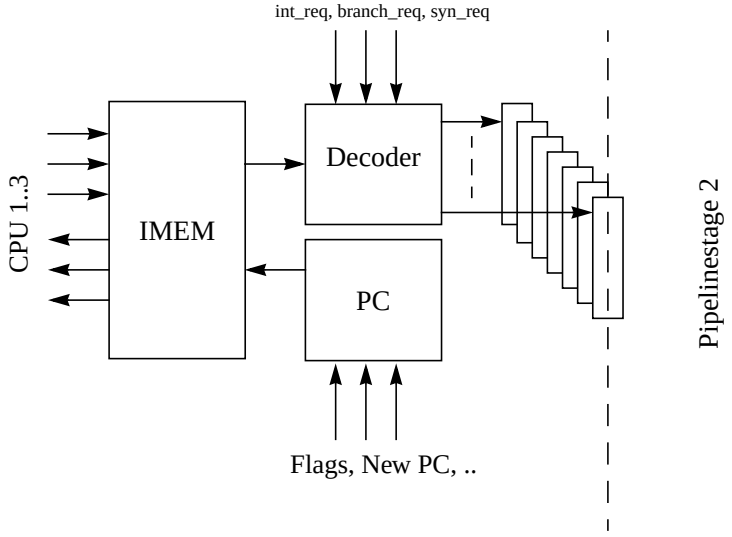


Abb. 24: Vereinfachter Datenpfad der ersten Pipelinestufe

4.5.2 Die zweite Pipelinestufe

Im Datenpfad der zweiten Pipelinestufe werden die Operanden selektiert, welche in der Arithmetisch-Logischen-Einheit (ALU) miteinander assoziiert werden. Ferner beinhaltet die zweite Stufe Multiplexer, um Speichermodule der Instruktion entsprechend korrekt zu adressieren. Außerdem werden hier die Sprungbedingung ausgewertet, ob im Falle einer Sprunginstruktion der Sprung ausgeführt wird. Deswegen werden auch hier die Flags der ALU gespeichert. Physikalisch befindet sich das PRF und die ALU in der zweiten Stufe. Die Architektur ist in der folgenden Abbildung dargestellt. Sie zeigt die Register Blöcke aus denen die Daten stammen, die in der ALU miteinander verknüpft werden. Ferner ist der Datenspeicher dargestellt, auf den mit Lade- und Speicherbefehlen zugegriffen wird. Der Konfigurationspfad, über den der Datenspeicher initialisiert wird, ist angedeutet. Die Flags der ALU werden in zwei Registern gespeichert. Im Falle eines Interrupts werden die Flags in einem temporären Register zwischengespeichert, um nach erfolgtem Rücksprung den alten Zustand wieder herzustellen.

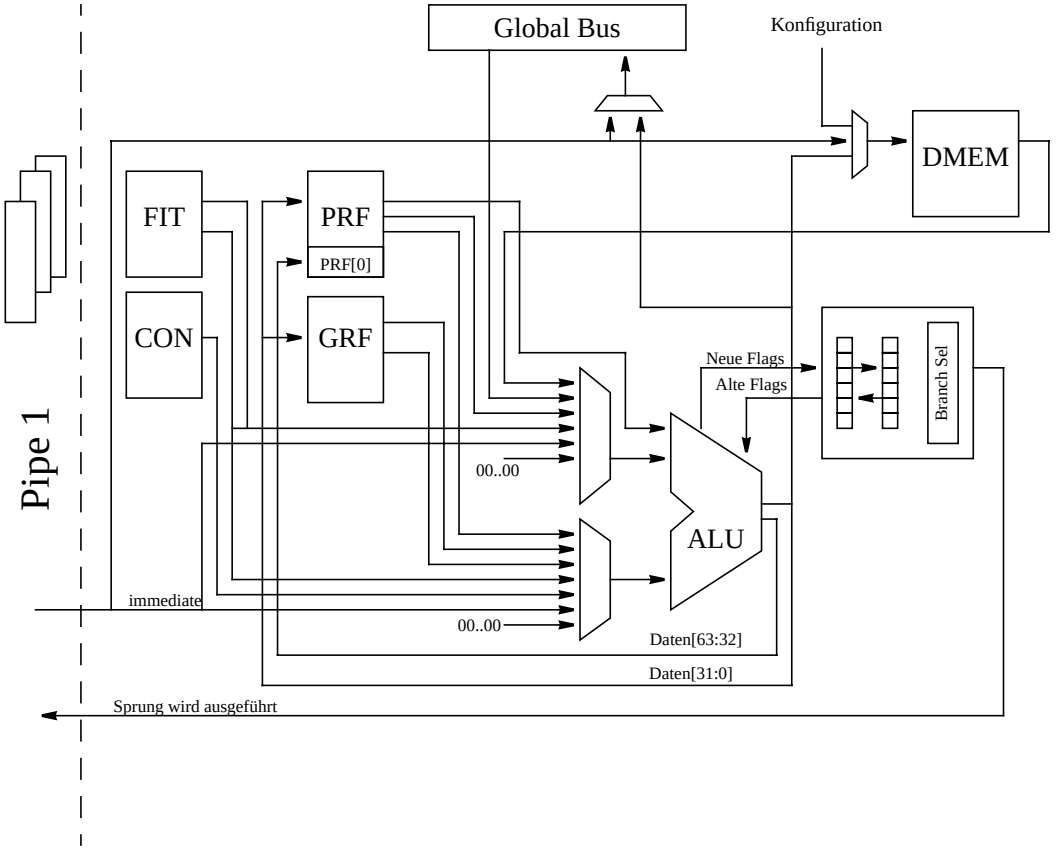


Abb. 25: Die zweite Pipelinestufe

Die Steuerwörter, welche die Multiplexer in der zweiten Stufe steuern, werden im Befehlsdecoder der ersten Stufe generiert. Der Datenpfad der zweiten Stufe ist so kurz wie möglich gehalten, da innerhalb eines Taktzyklusses die Daten aus einem Registerfile (PRF, GRF, CONST, FIT) selektiert, durch die ALU assoziiert und anschließend mit der steigenden Taktflanke wieder in ein Register (siehe Instruktionssatz) zurückgeschrieben werden. In dem Fall von Lade- oder Speicherbefehlen wird die Adresse unter Umständen aus einem lokalen Register geholt. Der kritische Pfad erstreckt sich in diesem Fall von der Selektion der privaten Registerstelle durch die ALU und den Arbiter zum gemeinsamen Datenspeicher.

Der Ablauf, in welcher Art die Daten in der zweiten Pipelinestufe verarbeitet werden, läßt sich in die drei folgenden Blöcke teilen:

- Selektion der Quelldaten,
- Verarbeiten der Daten,
- Selektion des Speicherortes und Zurückschreiben der Daten.

Für das Selektieren der Quelldaten werden die Steuerwörter *sb1* (select data from blocks to ALU source 1) und *sb2* verwendet, welche die Daten der möglichen Blöcke auswählen. Die Adresse, welche die Zeile innerhalb der Blöcke auswählt, wird durch das Steuerwort *sl1* und *sl2* (select Line in source block 1 or 2) bewerkstelligt. Diese Signale werden an alle in Frage kommenden Blöcke gelegt. Im Fall einer Speicherinstruktion wird zusätzlich zu den Daten, welche in das Zielregister geschrieben werden, die Adresse, welche den Speicherort selektiert, durch das Flag

sioas (select I/O address source) festgelegt. Dabei selektiert eine ‘0’ eine *immediate* Adresse, d.h. die Adresse ist Bestandteil der Instruktion und kommt aus der ersten Pipelinestufe. Eine ‘1’ bedeutet, dass die Adresse des Speicherortes sich im privaten Registersatz befindet.

Das Verarbeiten der Daten wird durch die ALU durchgeführt. Sie unterstützt 16 Befehle für die Verarbeitung von Daten im Festkommaformat. Alle Operationen können innerhalb eines Taktzyklusses durchgeführt werden. Die Division wird durch einen Radix-4 Dividierer implementiert, der für die Verarbeitung einer 64 Bit auf 32 Bit Division 18 Taktzyklen benötigt. Die Architektur der zweiten Stufe ist so konzipiert, dass grundsätzlich alle Daten durch die ALU geführt werden, gleichgültig ob Werte assoziiert werden oder nicht.

Die Selektion des Speicherortes wird durch die Steuerworte *db* (destination block), *sb3* (select block 3) und das Flag *wb* (write back) sichergestellt. Dabei wählt das Flag *wb* aus, ob Daten gespeichert werden sollen. Das Steuerwort *sb3* generiert die write enable Flags zu den Speicherblöcken: Datenspeicher, Globales/Lokales Register File sowie im Fall eines Synchronisationsbefehles das write enable (*we*) für dieses Register. Im Fall einer Speicher-Instruktion wird analog zu einer Ladeinstruktion die Adresse an den Speicherblock übergeben.

4.5.2.1 Die Arithmetisch-Logische-Einheit

Die Arithmetische-Logische-Einheit (ALU) verarbeitet 16 Instruktionen auf Festkommawerte. Die unterstützten Instruktionen sind in Tabelle 7 dargestellt [13]:

Befehl	Binär	Aktion	Ergebnis
ADD	0000	Addition	A + B
ADC	0001	Addition mit Carry	A + B + C
SUB	0010	Subtraktion	A - B
SBC	0011	Subtraktion mit Carry	A - B - C
MUL	0100	Multiplikation	A * B
MULS	0101	Multiplikation Neg.	A * B signed
DIV	0110	Division	A / B
DIVEND	0111	Ende der Division	Write Back A / B
EOR	1000	Exklusiv Oder	A xor B
AND	1001	Logisch Und	A and B
OR	1010	Logisch Oder	A or B
NEG	1011	Negation	0x0000 - B
SH	1100	Logisches Schieben	B << A
SHA	1101	Arithmetisches Schieben	B << A
ROR	1110	Rechts Rotieren durch Carry	ROR B, C
COM	1111	Komplementieren	$FFFF - B

Tab. 7: Alu-Befehle und ihre Wirkung

Alle dargestellten Operationen können innerhalb eines Taktzyklusses ausgeführt werden. Einzig die Division wird in 18 Zyklen ausgeführt, da für sie ein Radix-4 Verfahren implementiert ist. Das

Zurückschreiben des Ergebnisses wird durch eine separate Instruktion angewiesen. Das Einfügen dieser Instruktionen an der richtigen Stelle des Programmcodes muss durch den Benutzer geschehen. Nachdem eine Division angestoßen wurde, kann die ALU auch weiterhin verwendet werden. Zwischenzeitliche Operationen, die das Ergebnis der Division verwenden möchten (engl. Read after Write), werden nicht abgefangen. Alle anderen Operationen werden anhand des übergebenen ALU-Opcodes selektiert, wodurch die ALU angewiesen wird, das Ergebnis der entsprechenden arithmetischen Einheit an den Ausgang der ALU zu legen. Handelt es sich um eine Multiplikation, werden die oberen 32 Bit des Ergebnisses an den Ausgang *mul64* geschaltet, der direkt mit dem ersten Register des privaten Registersatzes (PRF[0]) verbunden ist, d.h. im Fall einer Multiplikation wird das Ergebnis in das erste Register des privaten Register-Files geschrieben. Deshalb sollte bei einer Multiplikation nicht das erste Register für das Ergebnis ausgewählt werden, da ansonsten das Ergebnis der Operation ungültig ist. Handelt es sich um eine Division, kommen die oberen 32 Bit des Dividenden aus diesem Register.

Für die Implementierung der ALU werden Standardelemente verwendet. Die logischen Verknüpfungen sind durch Gatter realisiert, die übrigen Operationen durch Elemente der Synopsys Standardbibliothek [53]. Der Multiplizierer und der Dividierer sind als zeitkritisch anzusehen und sind deshalb durch spezielle Strategien implementiert. Diese werden in den beiden folgenden Kapiteln beschrieben. Eine Übersicht zur Architektur der ALU liefert Abbildung 26.

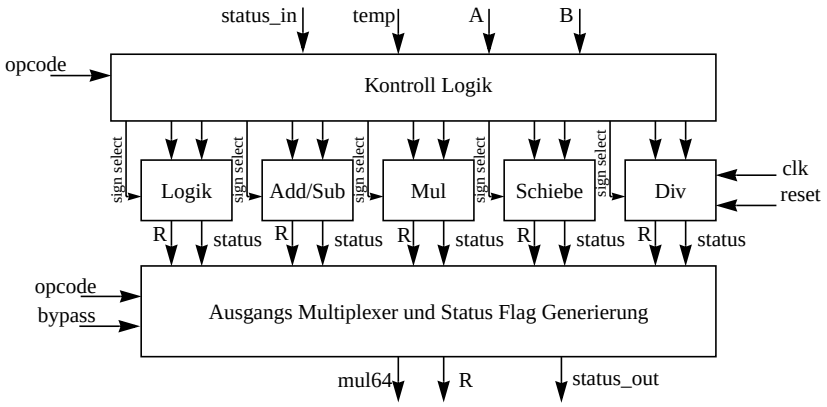


Abb. 26: Architektur der Arithmetisch Logischen Einheit

Das Resultat (R) einer arithmetischen Operation wird nicht in einem Register gespeichert, sondern direkt auf den Ausgang propagiert. Ein Multiplexer wählt dem Opcode entsprechend den Wert aus, der auf den Ausgang der ALU geschaltet wird. Die Eingangssignale *status_in* und *status_out* stellen die Flags dar, die in der ALU berechnet, aber nicht hier gespeichert werden. Stattdessen werden sie zusammen mit den anderen Flags im Datenpfad der zweiten Pipelinestufe gespeichert. Das Signal *bypass* steuert den Vorgang der Berechnung der Flags, d.h. ist das Signal gesetzt, werden die Werte, die am Eingang *status_in* liegen, unverändert auf den Ausgang *status_out* geschaltet. Das 4 Bit breite Eingangswort *opcode* steuert die Aktion der ALU.

Im beiden nachfolgenden Kapiteln werden Multiplikations- und Divisionsverfahren vorgestellt und hinsichtlich der Eignung für den Einsatz in der CPU untersucht. Dabei wird insbesondere auf

Geschwindigkeit und Flächenbedarf geachtet. Sequentielle Methoden haben zwar einen geringen Flächenbedarf, sind aber wegen der Vielzahl an benötigten Taktzyklen meist ungeeignet. Beispiele finden sich in [30],[32],[43]. Parallele Multiplikationsverfahren bilden das Produkt innerhalb eines Taktzyklusses, wobei die schnelle Ausführung zu Lasten der Chipfläche geht.

4.5.2.1.1 Multiplikation

Die Multiplikation wird in allen modernen Prozessoren eingesetzt, wobei der Vorgang der Multiplikation auf eine Folge von Additionen zurückgeführt wird. Serielle und parallele Verfahren unterscheiden sich in der Reihenfolge der Ausführung der Addition. In den folgenden Beispielen wird davon ausgegangen, dass beide Faktoren die gleiche Länge haben und die Zahlen im Zweierkomplement dargestellt sind. Das Produkt zweier N-stelliger Zahlen läßt sich grundsätzlich, gleichgültig in welchem Zahlenformat dargestellt, in einem Wort der Länge 2*N darstellen. Sequentielle Verfahren bilden für jede Stelle des Multiplikators das Partialprodukt und akkumulieren das Produkt in einem Register. Der Hardwareaufwand ist gering und die Verarbeitungszeit ist hoch. Feldmultiplizierer bilden das Partialprodukt der Bitstellen gleichzeitig und führen die Addition ebenfalls gleichzeitig aus. Der Hardwareaufwand steigt verglichen mit den sequentiellen Multiplizierern quadratisch, hat aber dafür eine schnellere Verarbeitungszeit, da die gesamte Verarbeitung innerhalb eines Taktes stattfindet. Für große Wortbreiten oder kurze Taktzyklen müssen Pipelineregister eingefügt werden. Dann steigt wiederum die Verarbeitungszeit. Parallele Multiplizierer bilden die Partialprodukte parallel und addieren diese anschließend mit Hilfe eines Multi Operanden Addierers (Carry Save Adder Tree). Im Allgemeinen läßt sich die Geschwindigkeit durch eine Reduzierung der Partialprodukte erreichen oder durch eine schnellere Addition der Partialprodukte. Abbildung 27 [50] zeigt die Multiplikationstechniken in einer Übersicht.

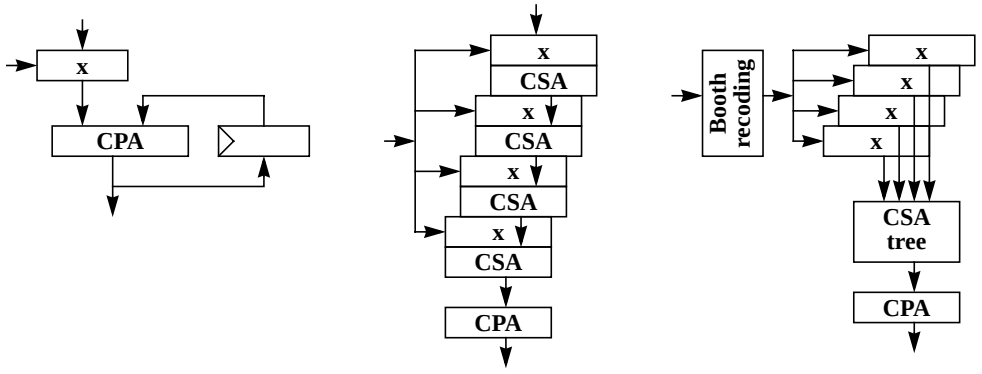


Abb. 27: Möglichkeiten der Multiplikation

Das erste Verfahren scheidet aus, da in diesem Fall 32 Taktzyklen für eine Multiplikation benötigt werden. Das zweite Verfahren kann innerhalb eines Taktes multiplizieren, benötigt dafür allerdings ca. 10 ns, was nicht der gewünschten Taktfrequenz von 120 MHz entspricht. Deshalb wird ein paralleles Verfahren ausgewählt und die Anzahl der Partialprodukte durch eine geeignete

Multiplikatorkodierung reduziert. Die Addition wird durch einen Carry Save Adder Tree beschleunigt. Am Ende der Berechnung werden die Summe und die Überträge zum Ergebnis mit Hilfe eines schnellen Addierers verknüpft.

Bei der Multiplikatorkodierung werden die Anzahl der Multiplikationszyklen reduziert, indem geeignete Vielfache des Multiplikanden gebildet werden. Das verbreitetste Beispiel ist das Multiplikationsverfahren nach Booth. Das Verfahren beruht auf der Tatsache, dass sich Multiplikationszyklen zusammenfassen lassen, falls das i-fache des Multiplikanden ($0<i<2^h-1$) zur Verfügung steht. Denselben Effekt erreicht man durch Verwendung betragskleinerer Vielfacher V des Multiplikanden ($-2^{h-1} < V < 2^{h-1}$). Enthält der Multiplikator einen Block von Nullen der Länge k, dann kann die Multiplikation durch Verschieben des Partialproduktes über k Stellen um k Additionsschritte beschleunigt werden. Bei einem Einsblock im Multiplikanden können wegen

$$w(00011110000) = 2^{v+1} - 2^{u}$$

v u

die v-u+1 Additionen des Multiplikanden durch eine Subtraktion und eine Addition ersetzt werden. Operationen (Sub und Add) sind nur an den 01- bzw. 10- Übergängen erforderlich.

Anschließend werden die Partialprodukte mit einem Walace tree (Carry Save Adder Tree) addiert. Dabei werden jeweils drei benachbarte Zeilen einer Matrix M in einem CSA-Addierer [50] zusammengefaßt. Am Ende werden noch die Summe mit den Überträgen mit Hilfe eines Carry-Propagate-Adders addiert. Tabelle 8 zeigt die Verarbeitungszeiten der in Abbildung 27 dargestellten Multiplizierern:.

	Sequentiell	Feld (CSA)	Parallel (Wallace)
Maximale Taktfrequenz	32 Takte	102 MHz	275 MHz
Fläche	minimal	0,11 mm^2	ca. 0,09 mm^2

Tab. 8: Gegenüberstellung verschiedener Implementierungen eines 32 Bit Multiplizierers in 0,18 µm CMOS Technologie

Tabelle 8 zeigt die Eignung des Multiplikationsverfahrens für diese Anwendung. Das Zeitverhalten genügt, bei gleichzeitig akzeptablem Flächenbedarf, der Anforderung in signifikant weniger als 8 ns zu multiplizieren. Deshalb ist als Multiplizierer ein Modul nach Wallace eingebaut.

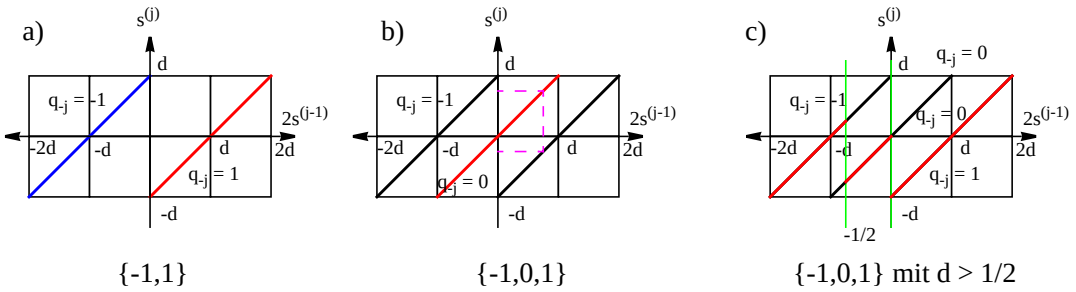
4.5.2.1.2 Division

Die Division ist ein vollkommen anders geartetes Problem, da die einzelnen Schritte der Division nicht parallelisiert werden können. Im Allgemeinen benötigt man k Rechenschritte, falls der Dividend die Bitbreite 2k und der Divisor die Breite k besitzt. Mit jedem Divisionsschritt wird der Teilrest u um eine Stelle verschoben, durch Vergleich mit dem Divisor d eine weitere Quotientenziffer gebildet und deren Produkt mit dem Divisor vom Teilrest subtrahiert. Dieses Verfahren, was einer schriftlichen Division entspricht, läßt sich durch sequentielle- oder parallele Verfahren implementieren. Sequentielle Verfahren bieten sich an, falls große Bitbreiten verarbeitet werden müssen oder eine besonders kompakte Architektur gefordert ist. Nachteil ist der unter

Umständen große Bedarf an Taktzyklen, bis das Ergebnis vorliegt. Falls negative Zahlen (im Zweierkomplement) verarbeitet werden sollen, wird häufig ein sog. Nonrestoring Verfahren verwendet, das anstatt der Ziffernmenge {0,1}, die Menge {-1,1} für den Quotienten verwendet. Dieser speichert ebenfalls den Teilrest u-2^kd, bemerkt aber im gleichen Takt, dass dies falsch war und führt im nächsten Takt eine Addition statt einer Subtraktion durch. Falls das Vorzeichen des Quotienten am Ende nicht übereinstimmt, ist noch ein Korrekturschritt notwendig. Am Ende muss der Quotient noch ins Binärformat gewandelt werden.

Parallele Verfahren reihen i.d.R. die Elemente des sequentiellen Dividierers hintereinander und produzieren somit den Quotienten unmittelbar. Allerdings ist der kritische Pfad dieser Implementierungsstrategie besonders lang und hat einen Flächenbedarf der quadratisch mit der Bitbreite steigt [43]. Deshalb ist dieses Verfahren besonders für kleine Bitbreiten geeignet. Außerdem ist das Verfahren leicht zu implementieren und es lassen sich beliebig viele Pipelinestufen einfügen.

Prinzipiell kann der Quotient zweier Zahlen aus einer Look Up Table (LUT) berechnet werden. Die Division wird dann auf das Aufsuchen eines Tabellenwertes zurückgeführt. Da jedoch der Umfang der Tabelle exponentiell mit der Länge des Divisors bzw. Dividenden steigt, ist eine vollständige Anwendung dieser Methode aus Gründen des Speicheraufwandes und der Tabellensuchzeit unrealistisch. Sinnvoll ist hingegen der Einsatz einer vereinfachten Tabelle, zur Berechnung mehrerer Quotientenbits in einem Divisionszyklus, die als Eingangsdaten nur die signifikantesten Bits des Divisors und des Dividenden enthält. Vorher muss festgelegt werden, wieviel Stellen gleichzeitig verarbeitet werden sollen. Aus Aufwandsgründen ist es zweckmäßig, die Anzahl der betragsverschiedenen Vielfachen auf höchstens 3 zu beschränken, wobei jeweils eines dieser Vielfachen dem Betrage nach größer, kleiner oder gleich 1 ist [17]. Bei dieser Wahl ist man beim Radix-4 Dividierer angekommen, der jeweils 2 Bits des Divisors zu einer Radix-4 Zahl zusammenfaßt und die Division somit um den Faktor vier gegenüber der sequentiellen Division beschleunigt. Um nicht das dreifache des Divisors zur Verfügung stellen zu müssen, wird der Zahlenbereich so eingeschränkt, dass man mit dem maximal zweifachen auskommt. Gewählt wird die Schranke 8/3d. Der Vorgang soll anhand einiger Beispiele erklärt werden. Die Darstellung des partiellen Divisionsrestes (p) und dem Divisor (d) erfolgt im sog. p-d Plot, womit die Auswahl des Quotientenbits bildlich dargestellt werden kann. Abbildung 28 [32] zeigt den Divisionsverlauf für die Ziffernmengen {-1,1}, {-1,0,1} und {-1,0,1} mit der Einschränkung, dass der geschobene Teilrest nicht größer als 1/2 ist.



{-1,1} {-1,0,1} {-1,0,1} mit d > 1/2

Abb. 28: p-d Plot für Ziffernmenge {-1,1} und {-1,0,1}

Die dargestellten Plots sind folgendermaßen zu interpretieren: Man nimmt den geschobenen partiellen Rest $2s^{(j-1)}$ und projiziert ihn auf die dargestellten Geraden. In Plot a) erreicht man genau eine Linie, die anzeigt, ob als Quotientenbit -1 (blau) oder 1 (rot) gewählt wird. Von dort aus projiziert man auf die y-Achse und liest den Rest der Operation ab, der im nächsten Schritt um eine Binärstelle geschoben wird und für die nächste Auswahl verwendet wird. Bei Verwendung der Ziffernmenge {-1,0,1} (Abb. 29 b) ist die Auswahl nicht mehr eindeutig und es kann für den Bereich [0,d] das Quotientenbit ’0’ oder ’1’ gewählt werden. Analog dazu für den Bereich [-d,0] das Bit ’-1’ oder ’0’. Der Vorteil ist, dass sozusagen ein Fehler bei der Auswahl zulässig ist. Das ist daran erkennbar, dass sich die Auswahllinien überlappen. Hat der geschobene partielle Rest beispielsweise den Wert 3/4d (magenta), kann als Quotientenbit der Wert ’0’ oder ’1’ ausgewählt werden. Trifft man zusätzlich die Einschränkung das d>1/2 ist (Abb. 29 c), erhält man die Quotientenbits durch zwei einfache Vergleiche. Das Bit ’-1’ wird gewählt, falls der partielle Rest kleiner als -1/2 ist, eine ’1’ falls der Rest größer als ’0’ und für alle weiteren Fälle wird ’0’ als Quotientenbit gewählt. Abbildung 28 c zeigt das sog. Radix-2 Verfahren. Beim Radix-4 Verfahren werden je zwei Stellen zu einer Radix-4 Ziffer zusammengefasst, d.h es werden die Quotientenziffern {-3,-2,-1,0,1,2,3,} verwendet. So gesehen benötigt man das 1- 2- und 3-fache des Divisors. Damit der Aufwand überschaubar bleibt, schränkt man die Quotientenziffern auf {-2,-1,0,1,2} ein, was möglich ist, falls d>1/2. Daraus folgt sofort, dass der Wert s (geschobener Wert des partiellen Restes) - maximal den Wert 2/3 d annehmen kann. Der Sachverhalt ist in Abbildung 29 [32] dargestellt:

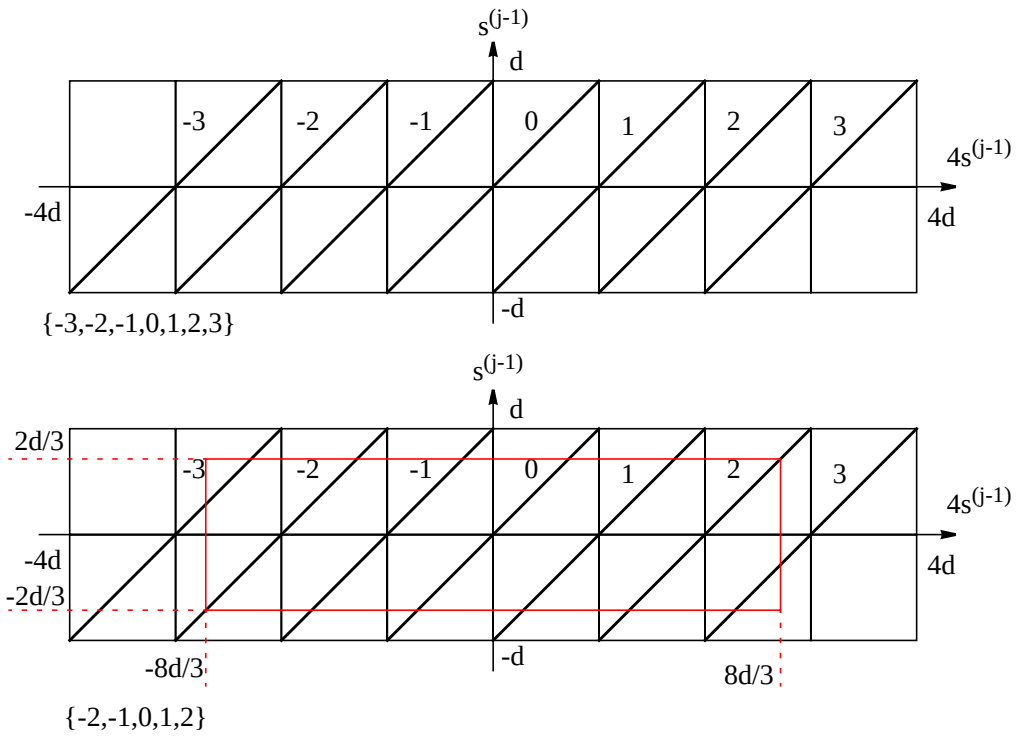


Abb. 29: Radix-4 mit Ziffernmenge {-3,-2,-1,0,1,2,3} und {-2,-1,0,1,2}

Der rot dargestellte Bereich zeigt, dass der gesamte Bereich des Quotienten abgedeckt ist. Das eigentliche Problem bei dieser Art der Division ist die „schnelle“ Auswahl des richtigen Quotientenbits. Letztendlich muss ein Verfahren gefunden werden, so dass durch die Verknüpfung

von wenigen Stellen bereits entschieden werden kann, welches Quotientenbit gewählt werden muss. Deshalb wird der partielle Rest der Division in zwei getrennten Registern gespeichert. Das sind die Summe und die Überträge, also in der sog. Stored Carry-Save-Form. Die Verknüpfung geschieht mit einem Carry-Save-Adder, der mit jedem Takt die beiden Teile des partiellen Rests, Summe und Übertrag, sowie den ausgewählten Vielfachen des Devisors miteinander verknüpft. Die Auswahl des Quotientenbits wird durch eine LUT vorgenommen die insgesamt sieben Stellen des partiellen Restes und vier Stellen des Divisors für die Auswahl benötigt. Da die Auswahl des Quotientenbits durch die signifikanten Bits des partiellen Restes (engl. *partial remainder*) p und des Divisors d erfolgt, wird als Darstellungsweise für die Auswahl des Quotientenbits häufig der sog. p-d Plot verwendet. Das Prinzip wird anhand der Radix-2 Division (Abb. 29 c) erläutert. Abbildung 30 [32] zeigt das Verfahren:

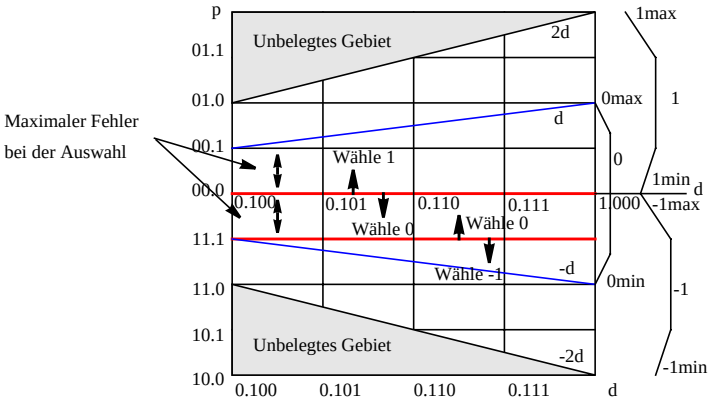


Abb. 30: p-d Plot für Radix-2 Division

An der y-Achse sind die drei signifikanten Stellen des partiellen Divisionsrestes dargestellt. An der x-Achse analog dazu die vier signifikanten Bits des Divisors. Die rot dargestellten Linien kennzeichnen die Grenzen der Wahl des Quotientenbits. Die signifikanten Stellen des Divisors sind für die Auswahl des Quotientenbits in diesem Fall nicht erforderlich, was an den horizontal verlaufenden Auswahllinien abgelesen werden kann. Die gefüllten Bereiche sind unzulässig, da der partielle Rest nicht größer als das Zweifache des Divisors (oben) oder kleiner als das negativ Zweifache des Divisors werden kann. Die blauen Linien zeigen die Grenzen des Bereichs, wann noch eine ’0’ als Quotientenbit gewählt werden kann. Deshalb muss die Auswahlgrenze so gewählt werden, dass durch den entstehenden Fehler bei der Auswahl durch die (wenigen)

signifikanten Stellen, nicht der zulässige Bereich verlassen wird. Die Darstellung kann dazu verwendet werden, die LUT zu erzeugen.

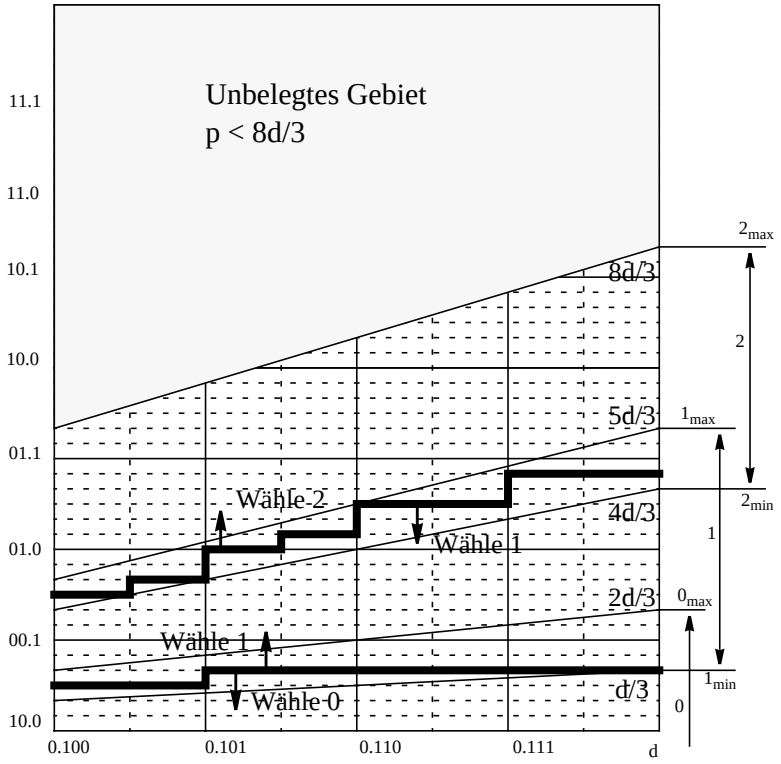


Abb. 31: p-d Plot für Radix-4 SRT Division mit Quotientenbits [-2,2]

In Abbildung 31 ist ein p-d Plot für ein Radix-4 SRT Verfahren dargestellt. Es ist erkennbar, dass die Auswahllinien nun treppenförmig verlaufen, da die Toleranzbereiche enger geworden sind. Deshalb sind in diesem Fall auch die fünf signifikanten Stellen des Divisors erforderlich, wobei vier Stellen für die Auswahl und eine weitere, um die unterste Stelle zu erzeugen, erforderlich sind. Von dem geschobenen Rest werden insgesamt sieben Stellen für die Auswahl benötigt. Die erzeugte LUT hat dementsprechend 2048 Einträge. Das Quotientenbit wird in drei Bits kodiert. Der gesamte Divisionsverlauf hat den folgenden Ablauf:

- Zuerst wird der Divisor solange nach links verschoben, bis die zweite Stelle des Divisors gesetzt ist. Im Falle eines negativen Divisors wird solange geschoben, bis die zweite Stelle eine Null enthält. Der Dividend wird um die gleiche Distanz verschoben wie der Divisor.
- Dann folgen 16 Divisionszyklen (64 Bit durch 32 Bit), so dass sich am Ende der Quotient im Quotientenregister befindet.
- Abschließend wird unter Umständen der Quotient noch korrigiert, falls der Rest und der ursprüngliche Dividend unterschiedliche Vorzeichen haben.

Die Architektur des Radix-4 Dividieres zeigt Abbildung 32:

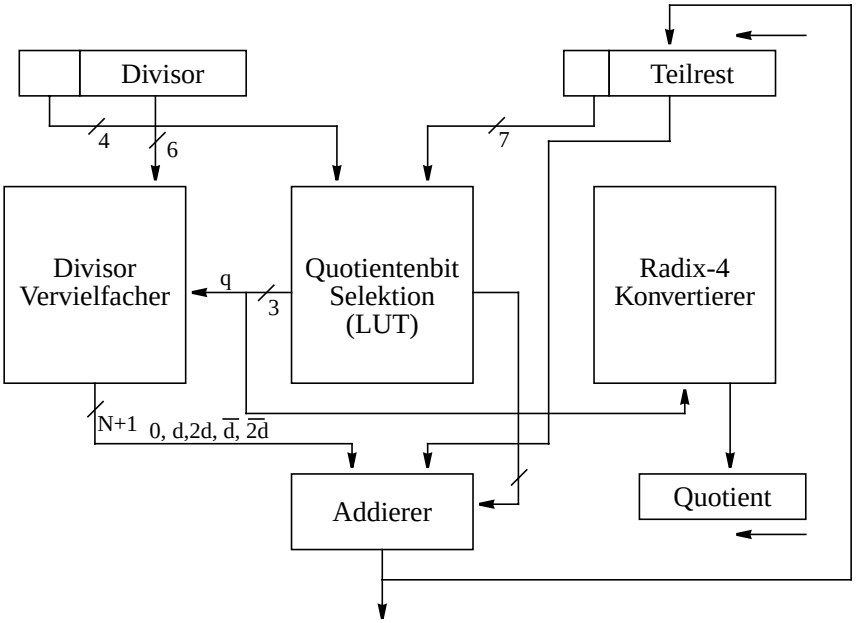


Abb. 32: Blockdiagramm des Radix-4 Dividierers

Das Prinzip soll an einer Division 16 Bits durch 8 Bits erläutert werden. Dabei werden die Ziffern $z=149_{10}$ (Dividend) und $d=5_{10}$ (Divisor) verwendet. Zuerst wird die binäre Darstellung des Divisors um vier Stellen nach links verschoben, so dass die zweite Stelle gesetzt ist. Um die gleiche Anzahl wird der Dividend geschoben. Dann erfolgt die Division, die analog zum Nonrestoring-Verfahren verläuft, nur dass hier die Ziffernmenge der Quotientenbits größer ist. Im Beispiel muss am Ende noch ein Korrekturschritt durchgeführt werden, womit der Fehler der letzten Operation ausgeglichen wird.

```
z          0000000010010101
d          00000101              \\ schiebe z und d um vier Stellen

S(0)       0000100101010000      \\ Wähle q(0)= 0 oder 1, wähle 0, schiebe um 2
4S(0)      0010010101000000
d          01010000

4S(1)      1001010100000000      \\ wähle q(1) = 2 => sub 2d
          +011000000
Rest:      111101010

4s(2)      1101010000            \\ schieben und q(2) = 0 oder -1 wählen, wähle 0

4s(3)      0101000000            \\ schieben und q(3) = -2 => add 2d
          +1010000000
Rest       1111000000            \\ Rest ist negativ => sub 1 von Q

         Q=[020-2]-1=30-1=29     \\ Korrekturschritt
```

4.5.3 Privater Registersatz

Der private Registersatz ist Bestandteil der zweiten Pipelinestufe. Er enthält die Operanden, die im Programmverlauf zur Berechnung benötigt werden. Außerdem wird er dazu verwendet, die Adressen zur indirekten Adressierung zu speichern. Er enthält insgesamt 16 Einträge mit einer Wortbreite von 32 Bit. Im Fall einer Multiplikation werden die oberen 32 Bit des Ergebnisses der Operation in das erste Speicherwort geschrieben. Deshalb ist in diesem Fall darauf zu achten, dass nicht dieses Register als Speicherort zur Aufnahme der unteren 32 Bit verwendet wird. Ansonsten werden die oberen 32 Bit der Operation verworfen und nur die unteren 32 Bit werden gespeichert. Für die Multiplikation von Faktoren kleiner Breite kann sich diese Eigenart zu Nutze gemacht werden. Beim Verwenden der indirekten Adressierung mit automatischen Inkrement enthält das erste Register des PRF die Adresse, welche inkrementiert wird. Die Speicherelemente des Registersatzes sind durch Flip-Flops implementiert. Als Schnittstelle besitzt das PRF zwei unabhängige Lese- und einen Schreibport sowie den dedizierten Eingangs- und Ausgangsport für das erste Speicherwort im PRF. Die Schreibzugriffe werden über ein separates *we* Signal gesteuert, sowie im Fall von PRF[0] über das Signal *we0*.

4.5.4 Interrupt Einheit

Jeder CPU ist eine Interrupt Einheit zugeordnet, die 16 verschiedene Interrupts auf zwei Ebenen ausführen kann, die individuell konfigurierbar sind. Es wird unterschieden zwischen Interrupts, welche grundsätzlich ausgeführt werden (Level 2) und solche die nur ausgeführt werden, falls es zugelassen ist (Level 1). Um Interrupts mit niedriger Priorität zu unterdrücken, ist ein spezieller Befehl im Instruktionssatz vorgesehen. Ferner können alle Interrupts niedriger Priorität über ein Konfigurationsregister einzeln unterdrückt werden, was für hochprioritäre nicht möglich ist. Diese werden dafür verwendet, den Prozessor in einen definierten Zustand zu bringen, was z.B. nach dem Anschalten der Fall ist. Ein Rücksprung ist für diese Interrupts nicht vorgesehen. Für Interrupts mit niedriger Priorität ist das möglich. Der Instruktionssatz beinhaltet vier dedizierte Befehle, um Interrupts zu behandeln. Das sind:

- INT: Software Interrupt
- IRT: Return from Interrupt
- CLI: Clear Interrupt
- STI: Set Interrupt

Der Befehl *INT* führt einen Software-Interrupt aus, d.h. dem Befehl wird als Argument ein 16 Bit breiter Vektor übergeben, der den Interrupt selektiert, der ausgeführt werden soll.

Der Befehl *IRT* kehrt zum Programmverlauf zurück, aus dem in die Interrupt Prozedur verzweigt wurde. Dabei werden zusätzlich zum Programmzähler die Flags der ALU restauriert. Die Inhalte der privaten und globalen Register bleiben unverändert.

Der Befehl *CLI* wird verwendet, um Interrupts niedriger Priorität zu verbieten. Die anderen Interrupts können auch weiterhin ausgeführt werden.

Der Befehl *STI* setzt die Aktion des Befehls *CLI* wieder zurück, so dass wieder alle Interrupts ausgeführt werden können.

Jede Interrupt Einheit verarbeitet die 16 möglichen Interrupts eines Prozessors. Dabei können die Interrupts von der CPU selbst durch den oben beschriebenen Befehl oder durch externe Peripheriegeräte ausgelöst werden. Jede Interrupt Einheit besitzt eine Schnittstelle zu einer CPU und zum globalen Bus des Prozessors. Die Schnittstelle zum globalen Bus dient der Initialisierung der *Interrupt Vektor Tabelle* (IVT) und der Konfigurationsregister innerhalb der Einheit. Die IVT beinhaltet die absoluten Sprungadressen der sechzehn implementierten Interrupts. Weiterhin sind drei 16 Bit breite Register implementiert, womit die Software- und Hardware-Interrupts bitweise ausmaskiert werden können und ein Register, welches die Ebene des entsprechenden Interrupts (bitweise kodiert) festlegt. Ferner sind Speicherelemente enthalten, welche die Nummer des Interrupts speichern, welcher sich gerade in der Verarbeitung befindet. Außerdem sind zwei Auswahleinheiten implementiert, die eine Auswahl treffen, falls zwei Interrupts gleichzeitig angefordert werden. Die Priorität ist abfallend sortiert. Ein endlicher Automat kontrolliert die Verarbeitung der Interrupts. Er befindet sich im Idle Modus, solange kein Interrupt angefordert wird. Kommt es zum Interrupt, wird die gegenwärtige Nummer des Interrupt gespeichert und ein *NOP* Befehl wird an den Befehlsdecoder übergeben, damit nicht im Fall einer Sprunginstruktion aus der Interrupt Prozedur herrausgesprungen wird. Gleichzeitig wird der Inhalt des Programmzählers und die Flags der ALU in Register gerettet, um nach erfolgter Verarbeitung der Prozedur wieder an diese Stelle zurückzuspringen. Der nachfolgende Zustand wartet darauf, dass ein *IRT* Befehl dekodiert wird, der anzeigt, dass aus der Prozedur zurückgesprungen wird und der Programmverlauf fortgesetzt werden kann. Dann wird die Anforderung (*request*) gelöscht und der Programmzähler und das Flag Register wird restauriert. Der folgende Zustand ist wieder der Ruhe Zustand. Während der Verarbeitung eines Interrupts ist es einem anderen Interrupt gleichen Levels nicht möglich, die laufende Verarbeitung des gegenwärtigen Interrupts zu unterbrechen. Statt dessen werden die anfallenden Anforderungen gespeichert und der Priorität entsprechend abgearbeitet. Interrupts des Levels 2 können laufende Interrupts des Levels 1 zu jeder Zeit der Verarbeitung unterbrechen. Sie dienen der Behandlung von Ausnahmezuständen und kommen unmittelbar zur Ausführung. Abbildung 33 zeigt die Architektur eines Interrupt Controllers.

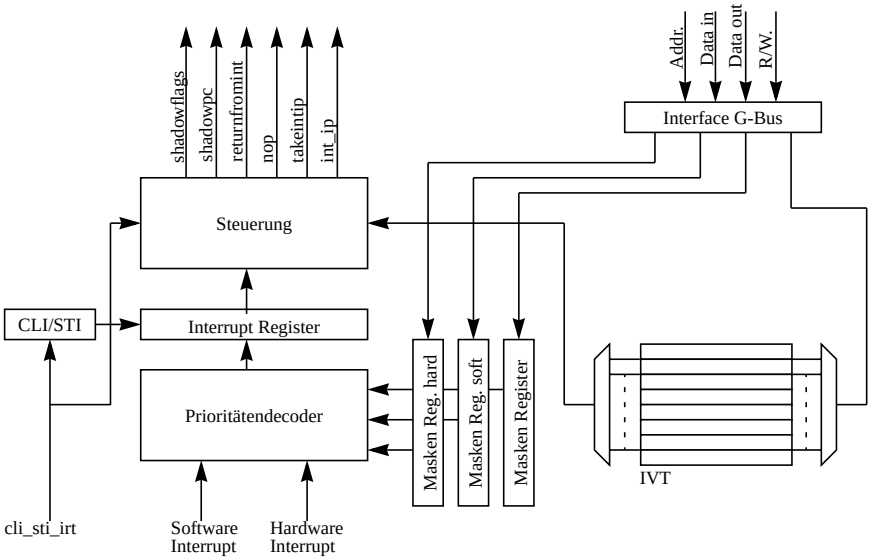


Abb. 33: Architektur eines Interrupt Controllers

Gleichgültig, ob ein Interrupt durch den Programmierer oder hardwareseitig durch ein Peripheriegerät ausgelöst wird, die Reaktion der Einheit ist die gleiche. Die Anforderungen werden intern logisch ODER verknüpft, so dass diesbezüglich keine Unterscheidung getroffen wird.

Im Fall eines Interrupts mit niedriger Priorität werden die Anforderungen in einem Register gespeichert, insofern die Ausführung eines Level 1 Interrupts nicht durch die Instruktion (*CLI*) verboten wurde oder sich ein Level 1 Interrupt bereits in der Verarbeitung befindet. Werden mehrere Interrupts (gleichen Levels) gleichzeitig angefordert, wird derjenige mit der niedrigeren Nummer zuerst ausgeführt. Alle weiteren angeforderten Interrupts werden gespeichert und ausgeführt, sobald der gegenwärtige Interrupt erfolgreich abgearbeitet wurde. Ist die Ausführung von Interrupts mit niedriger Priorität erlaubt und befindet sich nicht ein Interrupt in der Verarbeitung, wird mit der nächsten steigenden Taktflanke die Nummer des angeforderten Interrupts in ein internes Register übernommen. Anschließend wird in den nächsten Zustand gewechselt, in dem Steuersignale an den Datenpfad innerhalb des Prozessors übergeben werden. Diese bewirken, dass der gegenwärtige Wert des Programmzählers in ein internes Register übernommen wird, um den Programmverlauf nach abgeschlossener Verarbeitung wieder rekonstruieren zu können. Analog dazu werden die Flags der ALU in ein internes Register übernommen und der PC wird angewiesen, statt des Folgewertes den Wert der IVT zu übernehmen. Die gegenwärtige Instruktion wird aus der ersten Pipelinestufe entfernt und durch eine *NOP* Instruktion ersetzt, wodurch vermieden wird, dass nicht sofort wieder aus der Interrupt Prozedur herausgesprungen wird, falls sich gerade eine Sprunginstruktion in der Pipeline befindet. Dann wird in den nächsten Zustand übergegangen, in dem der Controller auf die Rücksprunginstruktion (*IRT*) der Interrupt Prozedur wartet. Wird diese detektiert, wird der Datenpfad angewiesen, den alten Wert des Programmzählers sowie die Flags wieder zu übernehmen. Anschließend wechselt der Controller wieder in den Idle Zustand und wartet auf den nächsten Interrupt. In jedem Zustand der Verarbeitung ist es möglich, dass die Verarbeitung eines Interrupts niedriger Priorität durch einen hochprioritären Interrupt unterbrochen wird.

Die Verarbeitung eines hochprioritären Interrupts beginnt mit dem Einsprung in die Interrupt Prozedur. Ein Rücksprung ist nicht möglich, weshalb der gegenwärtige Programmzähler und die Flags nicht gesichert werden. Mit dem ersten Befehl der Interrupt Prozedur werden alle Interrupt Anforderungen gelöscht. Mit dem nächsten Takt können wieder Interrupts angenommen werden.

5 *Anwendung im Übergangsstrahlungsdetektor*

Dieses Kapitel beschreibt die primäre Anwendung, in die der MIMD Prozessor sowie der Preprozessor (Kap. 6) integriert wird. Beide sind Teile einer Datenverarbeitungskette innerhalb eines Gesamtsystems, das aus 1,2 Millionen analogen Datenkanälen besteht und nach 6 µs Verarbeitungszeit eine Triggerentscheidung trifft. Eingangs wird auf das Experiment eingegangen, für das die Module entwickelt wurden. Anschließend wird die Trigger Elektronik vorgestellt, um dann die Arbeitsweise und die Integration darzustellen.

5.1 Das ALICE Experiment

Der Urknalltheorie zufolge entstand unser heutiges Universum vor ungefähr 15 Millarden Jahren aus einer einzigen Singularität unendlich hoher Energiedichte, die sich explosionsartig ausdehnte. Dabei durchlief das Universum verschiedene Phasenübergänge. Anfangs befanden sich alle Teilchen, wie Quarks, Leptonen, Eichbosonen sowie ihre Antiteilchen im thermodynamischen Gleichgewicht. Nach 10^{-35} Sekunden fand die Entkopplung von starker und elektroschwacher Kraft statt. Nach 10^{-32} Sekunden während eines weiteren Phasenübergangs “friert” die starke Wechselwirkung aus und ein kleiner Überschuss an Materie gegenüber Antimaterie wird erzeugt. Dieser Überschuss (ungefähr ein Billionstel) ist der entscheidende Teil, womit sich die gegenwärtige Dominanz an Materie erklären läßt. Die Temperatur in dieser Phase ist so hoch, dass sich die Quarks noch nicht zu Neutronen oder Protonen verbinden können, was als Quark Gluonen Plasmas (QGP) bezeichnet wird. Diese frühe Phase des Universums soll innerhalb des ALICE (**A** **L**arge **H**adron **C**ollider **E**xperiment) Experiments erforscht werden. Insbesondere soll der Phasenübergang zum QGP erforscht werden. Dafür werden innerhalb dieses Schwerionenexperiments Blei-Ionen mit Schwerpunktenergien um 1200 TeV zur Kollision gebracht, so dass die Nukleonen in ihre Bestandteile Quarks und Gluonen aufgebrochen werden. Die experimentelle Untersuchung ist ein wichtiger Test des Aspektes der Theorie der starken Wechselwirkung. Wie im frühen Stadium des Universums kühlt auch hier das Plasma aus und es bilden sich Hadronen, die in einem Detektor nachgewiesen werden können, so dass Rückschlüsse auf die Zustände während der Kollision möglich sind. Das QGP soll durch die Unterdrückung der Vektormesonen J/Ψ und Y beim Phasenübergang zum QGP nachgewiesen werden. Außerdem kann bei Kern-Kern Reaktionen eine Anreicherung mit Teilchen gemessen werden, die Strange Quarks enthalten. Da nach der Bildung von Hadronen keine Strange Quarks mehr erzeugt werden können, müssen diese vorher in einer Phase wechselwirkender Quark-Gluonen-Materie entstanden sein. Ein Anstieg in der Produktion kann somit als Indikator für die Existenz eines QGP gewertet werden.

Bei der zentralen Kollision von Kernen mit der beschriebenen Schwerpunktenergie ist mit bis zu 8000 geladenen Teilchen pro Rapiditätsintervall zu rechnen [1], aus denen mit Hilfe verschiedener Detektoren auf den Phasenübergang zum QGP geschlossen werden soll. Die besondere

Herausforderung des ALICE Experiments liegt in der großen Anzahl an geladenen Teilchen, die identifiziert werden müssen und in der kurzen Verarbeitungszeit die den einzelnen Detektoren dafür zur Verfügung steht.

5.2 Der ALICE Detektor

Der ALICE Detektor [1], einer der vier Detektoren am Large Hadron Collider (LHC), besteht aus insgesamt sechs zentralen Detektoren (ITS, TPC, TRD, TOF, PHOS, und HMPID) und einigen Vorwärtsdetektoren, die der Teilchenidentifikation, der Spurverfolgung (engl. *Tracking*) dienen oder als Auslösedetektor (engl. *Trigger*) verwendet werden. Der wichtigste Detektor ist dabei die Zeit-Projektionskammer (engl. *Time Projection Chamber - TPC*), die zur Spurrekonstruktion und Teilchenidentifikation verwendet wird, sowie der Übergangsstrahlungsdetektor (engl. *Transition Radiation Detektor - TRD*), der primär als Trigger der TPC arbeitet, aber auch zur Spurrekonstruktion verwendet wird. Diese beiden Detektoren werden im Folgenden genauer betrachtet. Die Funktionsweise der weiteren Detektoren kann in [1] nachgelesen werden. Abbildung 34 [2] zeigt den ALICE Detektor in einem Überblick:

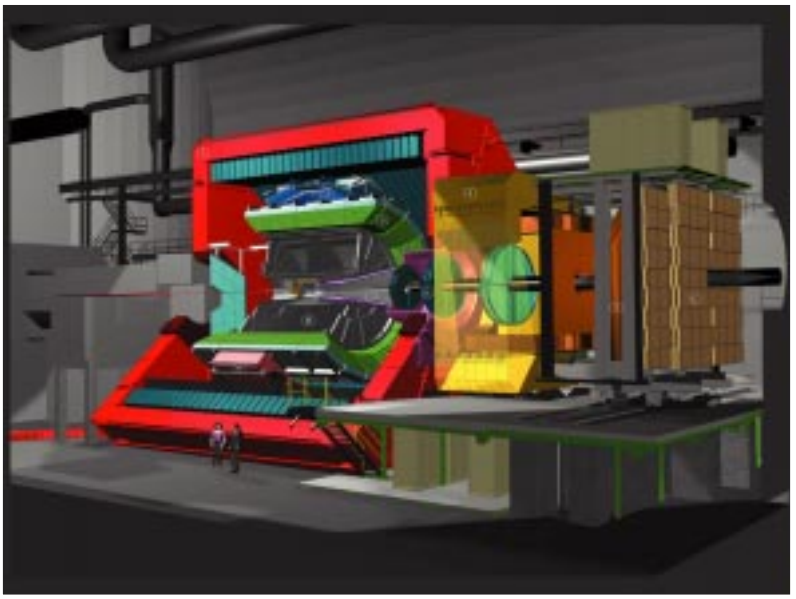


Abb. 34: Der ALICE Detektor

Der grau dargestellte Zylinder stellt die TPC dar, die radial von dem grün dargestellten TRD umgeben ist. Rot dargestellt ist der Magnet.

5.3 Die Zeit-Projektionskammer

Die TPC ist der Hauptdetektor des Experiments und liefert im Mittel die meisten Daten. Sie besteht aus zwei Zylindern mit einem Durchmesser von 1140 bzw. 5560 mm und einer Länge von 5100 mm. An beiden Enden sind die Zylinder durch Endkappen mit Proportionalitätskammern mit Kathodenpads ausgerüstet, die aus 18 Sektoren bestehen. Parallel zu den Endkappen befindet sich in der Mitte des Driftvolumens die zentrale Hochspannungselektrode. Zwischen den Endkappen und der Elektrode besteht ein elektrisches Feld, in dem die beim Durchgang der geladenen Teilchen durch Ionisation entstandenen Elektronen in ungefähr 80 μs zu den Endkappen driften. Dort werden die von den Elektronen erzeugten Signale detektiert, und mit Hilfe der Auslese-Elektronik der TPC verarbeitet. Ingesamt verarbeitet die TPC etwa 570000 analoge Datenkanäle, wobei die TPC im Schwerionen Modus, also wenn Blei Ionen zur Kollision gebracht werden, mit einer Frequenz von maximal 100 Hz betrieben werden kann [1].

5.4 Der Übergangsstrahlungsdetektor

Übergangsstrahlung wird emittiert, wenn ein geladenes Teilchen zwei Medien unterschiedlicher Dielektrizitätskonstante, also z.B. eine Anordnung aus Folien und Luftspalten passiert [24]. Emittiert wird die Strahlung an den Grenzflächen zwischen den beiden Medien. Bildlich gesprochen entsteht die Übergangsstrahlung, indem das geladene Teilchen mit der Bildladung im Medium einen Dipol bildet, dessen Feldstärke bei Annäherung des Teilchens an das Medium abnimmt und beim Eintritt vollständig verschwindet, wobei die variierende Dipolfeldstärke die Strahlung erzeugt, die in einem Kegel mit dem Öffnungswinkel $1/\gamma$ konzentriert ist. Die Intensität steigt dabei mit dem Lorentz-Faktor γ. Für experimentelle Anordnungen mit Folienstapeln ergeben sich konstruktive Interferenzeffekte, die ein Schwellenverhalten der Übergangsstrahlung bei bestimmten Werten von γ verursachen und durch einen Detektor registriert werden können. Insbesondere können dadurch Teilchen verschiedener Massen, aber gleichen Impulses (z.B. Elektronen und Pionen) aufgrund ihres unterschiedlichen γ identifiziert werden. Dieser Effekt wird auch im ALICE Experiment ausgenutzt. Als Radiator werden üblicherweise Folien aus Materialien niedriger Ordnungszahl verwendet, da die Absorption von Übergangstrahlung stark mit steigender Ordnungszahl zunimmt. Für den sog. Radiator im Alice Experiment wird eine Kombination aus Fiberglas-Platten und ausgehärteten Rohacell Schaum (HF71) verwendet, was einen Kompromiss aus mechanischer Festigkeit, chemischer Qualität und physikalischen Eigenschaften darstellt.

Hinter dem Radiator befindet sich eine Vieldraht-Proportionalitätskammer, welche zur Ortsmessung verwendet wird. Dabei macht man sich zu Nutze, dass ein ionisierendes Teilchen längs seiner Spur im Gas der Driftkammer Elektron-Ionen Paare erzeugt. Die erzeugten Elektronen driften entlang des elektrischen Feldes zum Anodendraht und werden dort durch das Feld in der Nähe des Anodendrahtes so stark beschleunigt, dass ein Sekundärionisationsprozess mit einer lawinenartigen Ladungsvermehrung auftritt (Gasverstärkung). Die zeitliche Differenz zwischen dem Teilchendurchgang und der Anstiegsflanke des Anodenimpulses hängt vom Abstand der Primärionisation und dem Ort des Anodendrahtes zusammen, also dem Zeitpunkt der Primärionisation t_0, bis zum Eintritt in die Region hoher Feldstärke und anschließender Lawinenbildung zur Zeit t_1. Die Ortskoordinate kann bei annähernd konstanter Driftgeschwindigkeit v durch die Beziehung $pos=v(t_1-t_0)$ bestimmt werden. Die konstante

Driftgeschwindigkeit wird erreicht, indem den Anodendrähten in bestimmten Abständen Kathodendrähte gegenübergestellt werden. Sobald die Elektronen, die sich im Driftvolumen gebildet haben, durch die Ebene, die durch die Kathodendrähte aufgespannt wird, bewegen, beginnt der Lawineneffekt, d.h. die Elektronen gewinnen im elektrischen Feld an Energie und ionisieren weitere Atome usw. Die Elektronen driften zum Draht, die Ionen entfernen sich radial vom Draht. Die in den Pads erzeugte Influenzladung wird ausgelesen. Das Prinzip ist in Abbildung 35 [1] dargestellt:

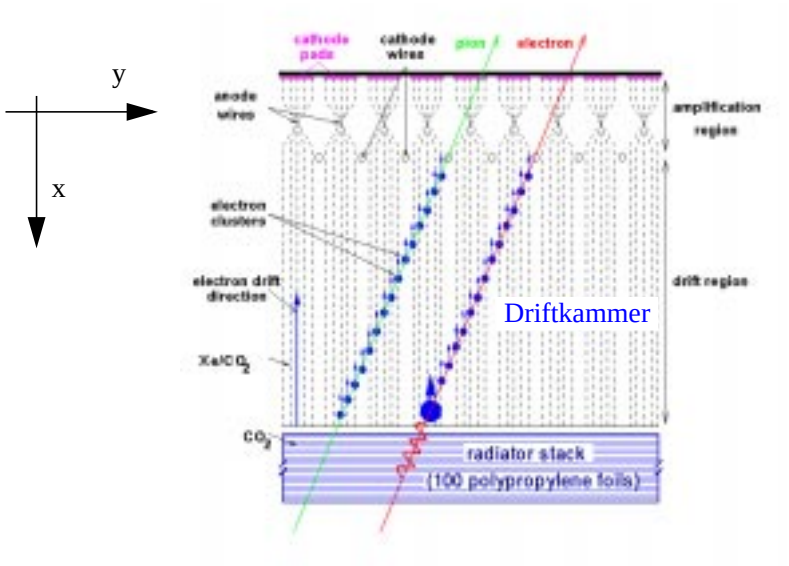


Abb. 35: Driftkammer mit Elektron und Pion

Das Bild zeigt die Driftkammer mit den Pads, dem Radiator und den Anoden- und Kathodendrähten. Die gestrichelten Linien stellen das elektrischen Feld dar, entlang dessen die Elektronen zu den Anodendrähten driften.

Der Übergangsstrahlungsdetektor wird primär zur Elektronen-Identifikation bzw. Elektronen-Pionen-Separation verwendet. Dabei haben die zu unterscheidenden geladenen Teilchen einen Transversalimpuls jenseits von 3 GeV/c. Zusammen mit den Daten des ITS- und des TPC-Detektors erhält man eine gute Elektronen-Identifikation. Der TRD dient der TPC dabei als Trigger, da die geringe Auslesefrequenz der TPC verbunden mit der Produktionsrate der Vektormesonen J/Ψ und Υ von 5% bei allen Ereignissen nicht ausreicht, um genug interessante Ereignisse mit der TPC zu detektieren. Durch Triggern auf Ereignisse, welche hochenergetische Elektron-Positron-Paare mit Tranversalimpulsen > 3 GeV/c enthalten, welche aus der Zerfallskette der Υ stammen, kann die Effizienz der TPC um eine Größenordnung gesteigert werden.

Der zylinderförmige Detektor besteht aus 5 Elementen in z-Richtung und 18 Elementen in ϕ–Richtung. Jedes dieser Elemente (Chamber) hat zwischen 12 und 16 sog. Pad Rows, die wiederum aus 144 Pads bestehen. Diese werden auf acht Chips aufgeteilt, die je 18 Kanäle verarbeiten. Die Aufteilung ist in Abbildung 36 [2] dargestellt. Die Abbildung zeigt ferner die Flugbahn von zwei geladenen Teilchen. Das rot dargestellte Teilchen hat einen schwachen Transversalimpuls, was im Magnetfeld des Detektors, in eine gekrümmte Spur resultiert. Das blau dargestellte Teilchen hat einen hohen Impuls und wird daher nur schwach gekrümmt. Der TRD hat die Aufgabe, diese

Teilchen zu finden und durch einen linearen Fit die Flugbahn zu beschreiben. Von der Elektronik werden nur die Teilchen detektiert, die eine steife Flugbahn beschreiben und maximal die Breite von zwei Kathodenpads (in ϕ-Richtung) überstreichen [2].

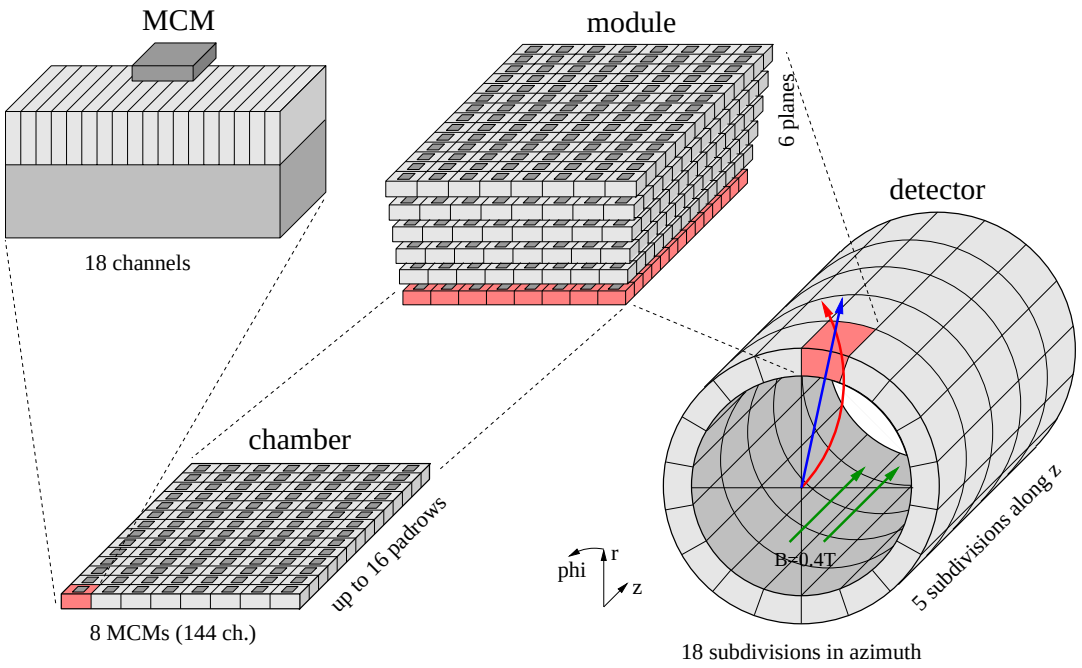


Abb. 36: Aufteilung des TR Detektors

Die TRD-Elektronik hat die Aufgabe, die Daten der 1,2 Millionen Kanäle des TR-Detektors zu verarbeiten und eine Triggerentscheidung zu treffen, ob innerhalb eines Ereignisses ein Elektron/ Positron-Paar mit hohem Transversalimpuls (> 3 GeV/c) den Detektor durchflogen hat. Diese Teilchen mit hohem Transversalimpuls korrespondieren mit einer „steifen“ Spur (Tracklet) innerhalb der oben beschriebenen Driftkammer (Tracklet). Die Aufgabe der Elektronik besteht darin, diese steifen Spuren zu selektieren und mittels verschiedener Verfahren zu qualifizieren. Die Diskretisierung zwischen Pionen und Elektronen wird mit Hilfe der Übergangsstrahlungssignatur vorgenommen. Die Spurverfolgung erfolgt mit Hilfe eines linearen Fits, der an die Datenpunkte der Driftkammern eine Gerade legt, die als Tracklet bezeichnet wird.

Der Detektor ist aus insgesamt sechs Lagen von Driftkammern aufgebaut, so dass die gefundenen Tracklets mittels der *Global Tracking Unit* (GTU) zu einer Spur (engl. *track*) zusammengesetzt werden können. Erst wenn die einzelnen Tracklets bestimmten Qualitätsmaßen entsprechen und eine Spur sich aus mindestens vier Tracklets zusammensetzt, wird eine positive Triggerentscheidung getroffen. Zusätzlich soll die Elektronik die Übergangsstrahlung detektieren. Diese resultiert in einem hohen Amplitudenwert, über mehrere Quantisierungspunkte, am Ende der Driftzeit. Nur steife Spuren, die zusätzlich eine Übergangsstrahlungssignatur tragen, werden von der Trigger-Elektronik akzeptiert. Dadurch können Ereignisse die von Pionen herrühren unterdrückt werden. Die Triggerentscheidung muss so schnell wie möglich erfolgen, da ansonsten die Daten der TPC verloren gehen. Für den TRD ergibt sich, dass die Entscheidung nach 6 µs vorliegen muss.

5.5 Die Auslesekette des TRD

Die Abbildungen 37 und 38 zeigen die Verarbeitungskette des TRD und den zugehörigen Zeitverlauf. Sie beginnt mit dem Drift der Elektronen im Volumen der Driftkammern, die ihrerseits die Elektronen-Cluster erzeugen, welche an den Kathodenpads der Driftkammern detektiert werden. Daran angeschlossen sind 10 Bit Analog Digital Wandler (ADC), die ihrerseits den Preprozessor mit Daten versorgen und zusammen mit den Event Buffern und dem MIMD Prozessor in einem Chip integriert sind. Der Preprozessor berechnet mit den digitalisierten Werten die Position der Elektronen-Cluster und führt eine Vorverarbeitung der Werte für die lineare Regression durch. Die Position und die daraus abgeleiteten Parameter werden während der Driftzeit in dem Fit Register File akkumuliert und am Ende dem MIMD Prozessor zur Verfügung gestellt. Durch das Berechnen der Fit Parameter während der Driftzeit der Elektronen, können 2 µs sinnvoll genutzt werden. Der MIMD Prozessor führt die Regression durch und liefert die berechneten Tracklets an die GTU, welche eine Triggerentscheidung liefert, ob ein interessantes Ereignis vorliegt.

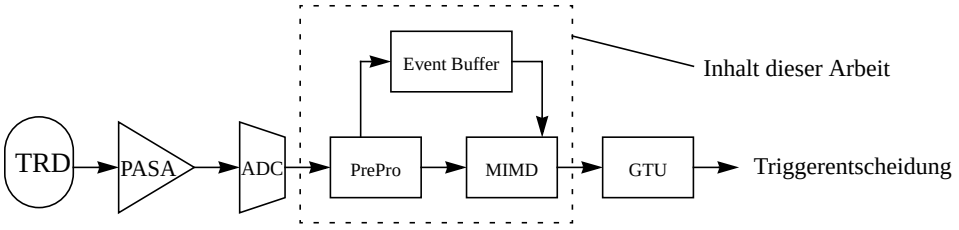


Abb. 37: Verarbeitungskette des TRD

Die Abbildung zeigt die einzelnen Verarbeitungsstufen im TRD, die sequentiell angeordnet sind. In den Event Buffern werden die Eingangsdaten gespeichert, um ein Ereignis vollständig rekonstruieren zu können. Der Preprozessor arbeitet mit der Frequenz der ADC's. Der MIMD Prozessor arbeitet mit 120 MHz und verarbeitet die Ergebnisse des Preprozessors durch vier CPU's. Die Resultate (Tracklets) werden über ein Netzwerk an die 'Global Tracking Unit' übergeben.

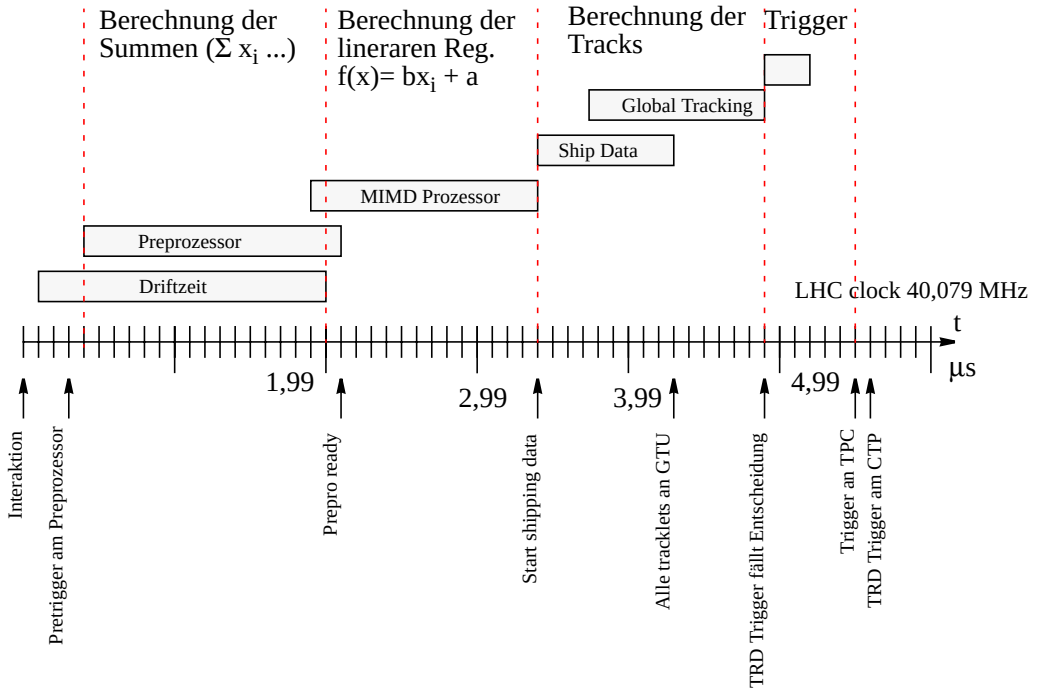


Abb. 38: Zeitlicher Verlauf der Verarbeitung im TRD

Dem Preprozessor stehen insgesamt 2 µs für die Verarbeitung der Eingangsdaten zur Verfügung. Da das Startsignal (Pretrigger) erst eintrifft, wenn bereits die Driftkammern Daten liefern, wird die Historie der Eingangsdaten zwischengespeichert, so dass beim Erreichen des Pretriggers alle Daten verarbeitet werden können. Vorher befindet sich das System im Ruhezustand (engl. *idle*). Sobald der Preprozessor die Verarbeitung abgeschlossen hat, wird der Preprozessor deaktiviert und der MIMD Prozessor aktiviert. Diesem stehen zum Berechnen der Tracklets insgesamt 1,8 µs zur Verfügung, was bei einer Taktfrequenz von 120 MHz in 216 Taktzyklen resultiert. Darin enthalten ist die Übergabe der Tracklet-Parameter an die Netzwerk Schnittstelle, was seinerseits die Daten an die GTU verschickt. Dafür sind 0,9 µs vorgesehen. Dann folgt die Triggerentscheidung, die an die GTU versendet wird.

5.6 Elektron Pion Separation und Spurrekonstruktion

Die Aufgabe des TRD ist, Elektronen von Pionen mit großem Transversalimpuls zu separieren und die Spuren zu rekonstruieren. Der Radiator emittiert Photonen, die am Ende der Driftzeit einen grossen Amplitudenwert erzeugen (Übergangsstrahlungssignatur). Die Driftkammern erzeugen die Datenpunkte zur Rekonstruktion der Flugbahn der geladenen Teilchen. Dabei macht man sich die Eigenschaft zu Nutze, dass hochenergetische geladene Teilchen eine steife Spur im Magnetfeld des Detektors beschreiben. Die Elektronik des Detektors berücksichtigt deshalb nur Teilchen, die innerhalb einer Detektor-Lage maximal die Breite von zwei Kathodenpads überstreicht. Für diese steifen Tracklets wird während der Driftzeit die Position jedes Ladungs-

Clusters berechnet, so dass unter Berücksichtigung des Zeitpunktes eine lineare Regression durchgeführt werden kann. Der folgende Zusammenhang ist dabei implementiert [6], [8]:

$$y_i = bx_i + a \qquad (1)$$

$$b = \frac{N \cdot \sum_{i=0}^{N-1} x_i \cdot y_i - \left(\sum_{i=0}^{N-1} x_i \cdot \sum_{i=0}^{N-1} y_i \right)}{N \cdot \sum_{i=0}^{N-1} x_i^2 - \left(\sum_{i=0}^{N-1} x_i \right)^2} \qquad (2)$$

$$a = \frac{\sum_{i=0}^{N-1} x_i^2 \cdot \sum_{i=0}^{N-1} y_i - \left(\sum_{i=0}^{N-1} x_i \cdot y_i \cdot \sum_{i=0}^{N-1} x_i \right)}{N \cdot \sum_{i=0}^{N-1} x_i^2 - -\left(\sum_{i=0}^{N-1} x_i \right)^2} \qquad (3)$$

Abb. 39: Lineare Regression

In den obigen Formeln gibt die Variable x_i die Driftdistanz innerhalb der Driftkammer an und die Variable y_i die Position eines Ladungsclusters in der Padebene, was bezogen auf Zylinderkoordinaten einer Position in ϕ-Richtung entspricht. Der Wert x_i entspricht einem Ort in radialer Richtung. Die Position in y-Richtung wird über das Verhältnis der Ladung der drei beteiligten Kanäle berechnet. Der Preprozessor berechnet die in den Gleichungen dargestellten Summen während der Driftzeit, so dass anschließend nur noch die übrigen arithmetischen Operationen vom MIMD Prozessor ausgeführt werden müssen, um den Achsenabschnitt und die Steigung der Regressionsgeraden zu berechnen. Dadurch kann die lineare Regression in der kurzen Zeit und der niedrigen Frequenz durchgeführt werden. Der Preprozessor berechnet die dargestellten Summen und speichert die Werte im Fit Register File. Der MIMD Prozessor führt die oben dargestellte Regression für die berechneten Werte des Preprozessors aus. Dabei berücksichtigt er, dass benachbarte Tracklets von einem Teilchen stammen können und führt diese zusammen. Ferner wird die Amplitudensumme ausgewertet, die zur Identifikation einer Übergangsstrahlungssignatur verwendet wird. Durch die Flexibilität des Prozessors können verschiedene Algorithmen ausgeführt werden.

Die Elektronen-Pionen Diskretisierung findet über die Integration der eintreffenden Amplitudenwerte statt. Im Mittel korrespondieren die Ladungs-Cluster innerhalb der Driftkammer mit einem ADC Wert, der im Bereich 40 (von 1024 möglichen) liegt. Für Elektronen, die Übergangsstrahlung emittierten, liegt der ADC Wert der Übergangsstrahlung bei > 100. Der Sachverhalt ist in Abbildung 40 [2] dargestellt.

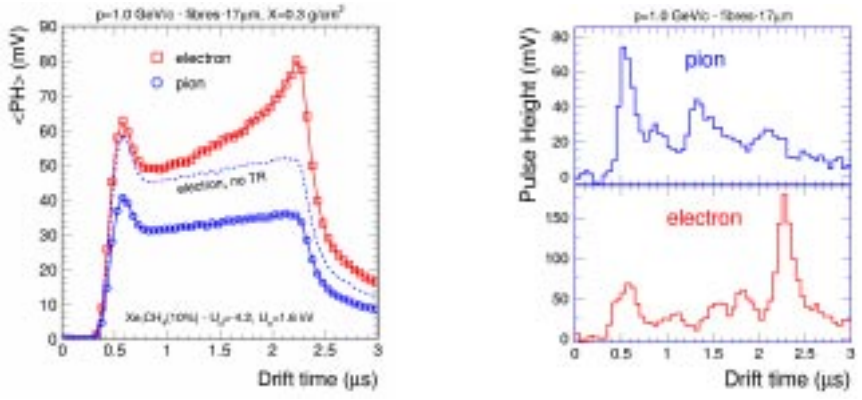


Abb. 40: Amplitudenverlauf für geladene Teilchen mit und ohne Übergangsstrahlung

Dementsprechend wird vom Preprozessor (Kap. 6) die Amplitudensumme der drei beteiligten Pads im letzten Drittel der Driftzeit aufsummiert und mit den Parametern der linearen Regression an den MIMD Prozessor übergeben. Die Auswertung erfolgt innerhalb der GTU, die ihrerseits eine Entscheidung darüber trifft, ob das Tracklet verworfen oder verwendet wird.

6 *Der Preprozessor*

Das nachfolgende Kapitel beschreibt die Funktion und die Architektur des Preprozessors, welcher für die Datenakquisition zuständig ist und die Datensätze produziert, mit denen der MIMD Prozessor anschließend arbeitet. Eine detailierte Beschreibung des Preprozessors ist in [20] dargestellt.

Die Funktion des Preprozessors liegt in der Berechnung der Summen der in Kapitel 5 dargestellten Formeln (2) und (3). Die Position (y_i) des Ladungsclusters wird aus einem Verhältnis der drei beteiligten Amplituden berechnet. Die Variable x_i stellt den Zeitpunkt, gemessen ab dem Eintreffen des Pretriggers bis zum Ende der Driftzeit, dar. Die weiteren Werte, die für die lineare Regression notwendig sind, werden aus diesen beiden Werten abgeleitet. Ferner werden die Amplitudenwerte der gefilterten Rohdaten akkumuliert, womit auf das Auftreten von Übergangsstrahlung geschlossen werden kann. Die Resultate werden im Fit Register File akkumuliert. Vor der Berechnung der Position, werden die eintreffenden Amplitudenwerte der ADC's durch einen digitalen Filter vorverarbeitet. Zur Rekonstruktion des Ereignisses werden zusätzlich alle Rohdaten in sog. *Event Buffern* zwischengespeichert. Diese können über den globalen Bus vom MIMD Prozessor ausgelesen werden. Der Preprozessor arbeitet mit der Abtastfrequenz der ADC's, wodurch ein geringer Leistungsverbrauch gewährleistet werden kann und das digitale Rauschen auf ein Minimum reduziert wird.

6.1 Überblick

Das nachfolgende Kapitel zeigt die Architektur des Preprozessors mit dem fünfstufigen digitalen Filter. Die Beschreibung der einzelnen Verarbeitungseinheiten orientiert sich an dem Prototyp Trap 1. Der Preprozessor besteht aus einem Frontend, das die Eingangswerte im Zeitmultiplexverfahren auf die Berechnungskanäle verteilt. Drei Kanäle werden mit digitalen Werten beschaltet, die aus ADC's stammen, die in den Prototypen integriert sind. Die weiteren Blöcke sind die Event Buffer, welche die eingehenden Rohdaten speichern und über den globalen Bus des MIMD Prozessors ausgelesen werden können. Die Konfigurationseinheit stellt den integrierten Blöcken verschiedene Parameter zur Verfügung, die für die Berechnung der y-Position benötigt werden. Die Einheit zur Berechnung der Fit-Parameter führt die arithmetische Verknüpfung der gefilterten Eingangswerte durch und akkumuliert diese im Fit Register File. Diese ist gleichzeitig die Schnittstelle zwischen Preprozessor und MIMD Prozessor.

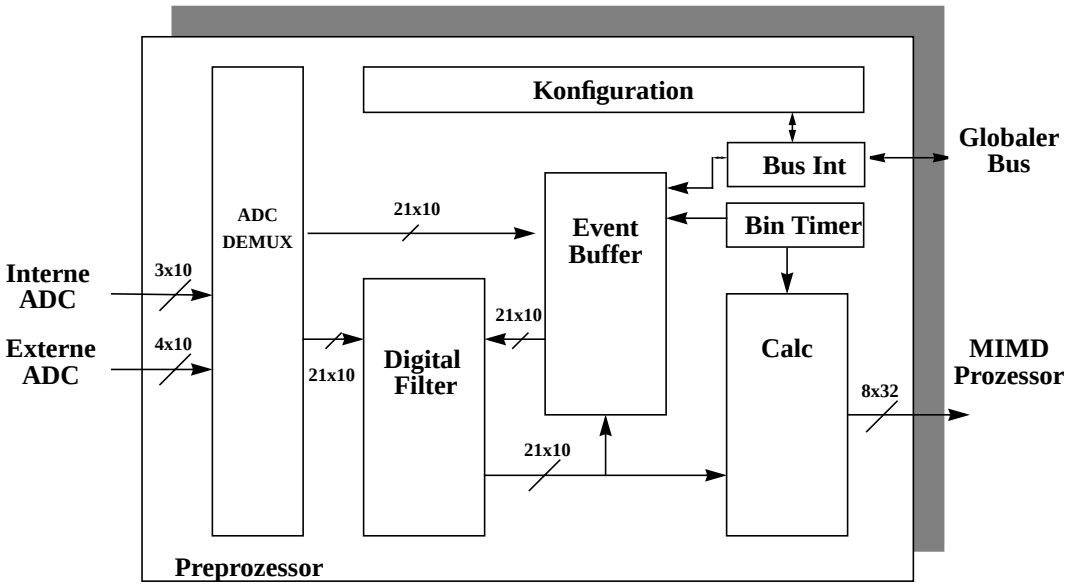


Abb. 41: Architektur des Preprozessors

6.2 Frontend mit digitalem Filter

Das Frontend des Preprozessors besteht aus vier Demultiplexern, welche von jeweils vier 10 Bit breiten Eingängen mit Daten von externen ADC's [62] beschaltet werden. Diese schalten die Daten im Zeitmultiplexverfahren auf 16 Datenkanäle des Preprozessors. Dabei werden die äußeren rechten und linken acht Kanäle von außen über die Demultiplexer versorgt. Die mittleren Kanäle verwenden die Daten der integrierten ADC's. Zwei der eingekoppelten Daten der internen ADC's werden für jeweils zwei Kanäle verwendet. Daran angeschlossen sind 21 FIFO's konfigurierbarer Tiefe (zwischen 0 und 6), womit die Historie der eintreffenden Daten gespeichert werden kann. Diese sind mit den Eingängen der digitalen Filtereinheit verbunden. Danach folgt eine fünfstufige Filtereinheit. Diese beginnt zunächst mit einem Nichtlinearitätsfilter, der den Fehler der ladungsempfindlichen Vorverstärker eliminiert. Korrigiert werden die Werte mit Hilfe einer Look Up Table (LUT), welche die Differenzen der Übertragungsfunktion eines idealen Vorverstärkers und der tatsächlichen Realisierung (PASA), in 64x5 Bit Werten speichert. Diese Werte stammen aus Simulationen und Messungen [1]. Damit werden 16 Eingangswerte auf einen Wert der LUT abgebildet, da davon ausgegangen werden kann, dass die Änderung des Eingangswertes allmählich verlaufen wird. Die Korrektur erfolgt auf die unteren 4 Bits sowie zwei Nachkommastellen. Nach erfolgter Korrektur beträgt die Nichtlinearität noch 0,7 LSB's. Daran angeschlossen ist ein Filter, der den Pedestal der ADC's eliminiert. Da aufgrund der auftretenden Prozessvariationen dieser Offset unterschiedlich ausfallen kann, ist für jeden Datenkanal eine individuelle Korrektureinheit implementiert. Diese bestimmt im Idle Modus des Preprozessors, also während der Zeit in der keine Eingangsdaten eintreffen, das durchschnittliche Pedestal, bevor

dieser individuelle Mittelwert vom Eingangswert abgezogen wird. Vorher wird der Nullpunkt der Eingangswerte durch Addition einer Konstanten angehoben, um Überläufe zu vermeiden.

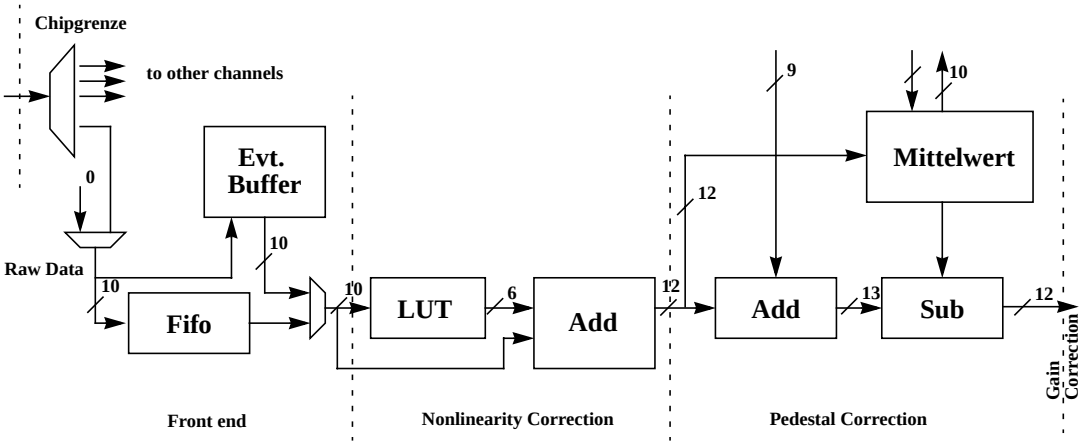


Abb. 42: Architektur des Frontends für einen Datenkanal

Für die Implementierung werden Standardelemente verwendet. Es erfolgt keine Speicherung in Registern.

Dann folgt eine Korrektur des Verstärkungsfaktors (engl. *gain*), d.h es werden die Unterschiede der Kanäle untereinander ausgeglichen. Erreicht wird dies über eine Multiplikation mit einem Wert Nahe 1. Die Korrektur erfolgt mit einem maximalen Wert von +/- 1/16, wobei der Korrekturfaktor über ein Verhältnis der eintretenden Werte errechnet wird. Der Korrekturfaktor wird solange verändert, bis der erwünschte Wert eingestellt ist. Berechnet wird der Faktor durch eine der CPU's in einem speziellen Testmodus. Die Ausgangswerte des zweiten Filters können hierfür über den globalen Bus ausgelesen werden. Der Korrekturfaktor wird für alle Kanäle individuell berechnet. Der Korrekturfaktor verwendet eine Vorkommastelle und 11 Nachkommastellen.

Angeschlossen an den Filter zur Korrektur der Gainverschiebung ist der sogenannte „Tail Cancellation Filter“, der als IIR-Filter implementiert ist. Mit ihm wird die Signalverfälschung ausgeglichen, welche durch den langsameren Drift der Ionen gegenüber den Elektronen innerhalb der Verstärkungsregion der Driftkammern hervorgerufen wird [5], [24]. Die Driftkammern liefern die analogen Eingangswerte der ladungsempfindlichen Vorverstärker, die wiederum die ADC's mit Daten versorgen. Die Impulsantwort der Driftkammer, welche die analogen Eingangswerte des Gesamtsystems bestimmt, kann durch zwei Exponentialfunktionen approximiert werden, die durch Inversion der Übertragungsfunktion zur Filterfunktion führt. Abbildung 43 zeigt den Verlauf der Impulsantwort und das Model zur Approximation durch zwei Exponentialfunktionen. Außerdem ist die Umkehr des Modells dargestellt, das für jeden Kanal implementiert wurde. Die Filterkoeffizienten werden durch ein numerisches Verfahren festgelegt [20] und können konfiguriert werden.

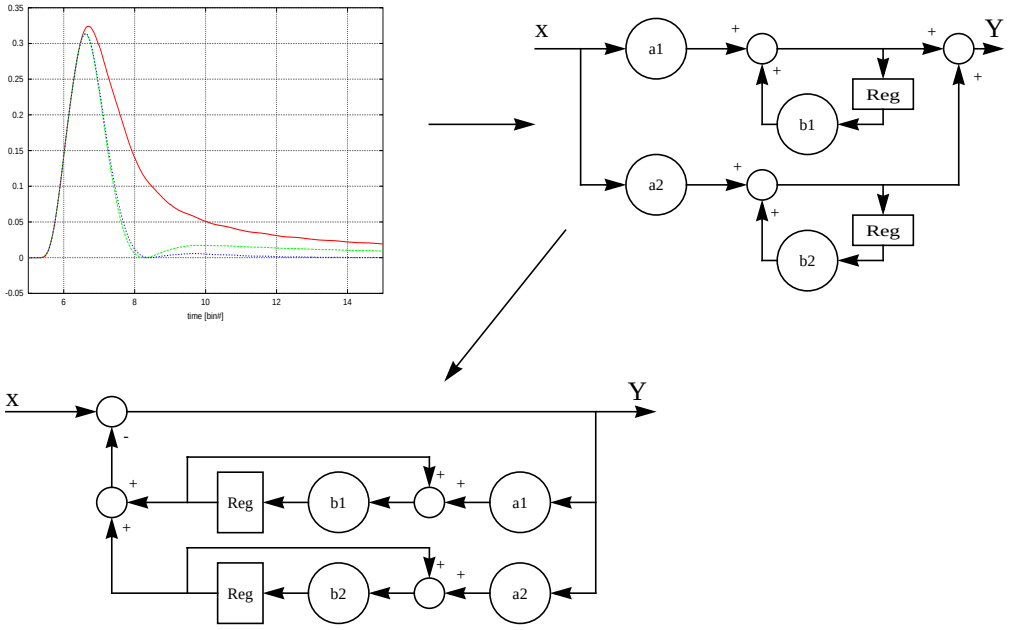


Abb. 43: Tail Cancellation Filter

Als letzte Filterstufe folgt ein sogenannter Crosstalk Filter (als FIR-Filter implementiert), der das Übersprechen benachbarter Kanäle ausgleicht. Er ist als 5x3 Filter implementiert. Alle Werte werden additiv verknüpft und liefern einen 12 Bit breiten Output, der schließlich im Preprozessor weiterverarbeitet wird. Die fünf Filterkoeffizienten sind für alle 21 Kanäle gleich und werden über den globalen Bus konfiguriert.

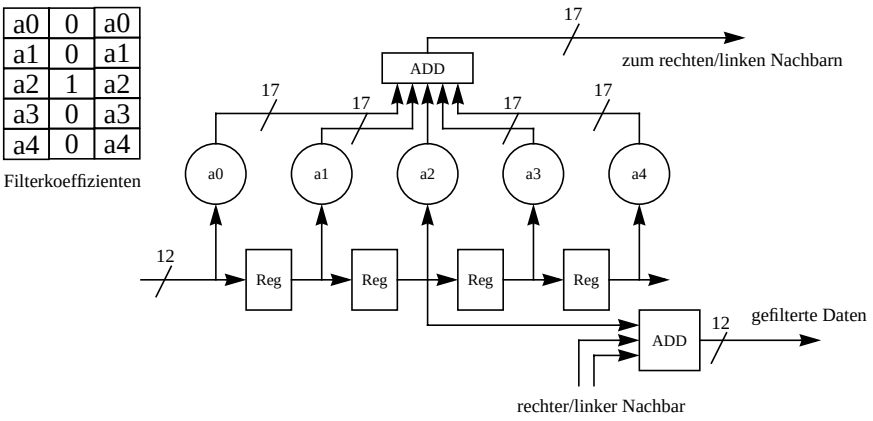


Abb. 44: Architektur des Crosstalk Filters

Nachdem die Eingangsdaten durch den fünfstufigen Filter verarbeitet wurden, werden die Daten an eine Selektionseinheit übergeben, welche die maximal vier höchsten Eingangswerte auswählt und in die vier Verarbeitungskanäle schiebt.

6.3 Auswahl der Datensätze

Zu jedem Zeitpunkt während der Datenakquisition werden für jeden Kanal genau dann die Eingangswerte verwendet, um die Parameter des linearen Fits zu bestimmen, falls die folgenden Bedingungen erfüllt sind:

- Die Summe der Eingangsdaten einer Gruppe von drei nebeneinander liegenden Kanälen muss größer sein als eine einstellbare Schwelle.
- Der Eingangswert des linken Nachbarn muss kleiner oder gleich sein als der betrachtete Kanal.
- Der Eingangswert des rechten Nachbarn muss kleiner sein als der betrachtete Kanal.

Von den maximal zehn Kanälen, die dieses Kriterium erfüllen können, werden vier ausgewählt, von denen die Position und die abgeleiteten Parameter berechnet werden. Die vier ausgewählten Kanäle besitzen die größte Amplitudensumme der drei beteiligten Kanäle. Von diesen Werten werden die Parameter der linearen Regression berechnet (vgl. Kap. 5.6, Abb. 33) und in einem Registersatz akkumuliert, um die Summen nach erfolgter Berechnung dem MIMD Prozessor zur Verfügung zu stellen. Zur späteren Rekonstruktion des Experiments werden alle Eingangsdaten in Speicherelementen (*Event Buffer*) zwischengespeichert. Des Weiteren ist es möglich in den Event Buffern Testdaten abzulegen, um das korrekte Verhalten des Preprozessors durch einen exemplarischen Datensatz zu überprüfen.

Für jeden ADC Takt werden vier der 19 Datensätze ausgewählt - die beiden äußeren Kanäle bleiben unberücksichtigt und werden nur zur Positionsbestimmung verwendet - und in die Berechnungspipelines geschoben. Dabei werden diejenigen ausgewählt, welche die größten ADC Summen besitzen und die oben beschriebenen sog. „Hitbedingungen“ erfüllen. Diese werden durch Komparatoren implementiert, deren Ergebnisse logisch UND verknüpft werden. Daraus resultiert ein 19 Bit breiter Vektor, der anzeigt, welche Kanäle die Eingangsbedingungen erfüllen. Dieser Vektor wird der Auswahllogik übergeben, die wiederum aus den vorselektierten Werten die besten vier auswählt. Das Verfahren eliminiert sukzessive die Kanäle mit den niedrigsten Amplitudenwerten, bis maximal vier übrig geblieben sind. Über Multiplexer werden die ausgewählten Kanäle an die Verarbeitungskanäle übergeben. Es erfolgt keine Speicherung in Register. Die Auswahllogik ist rein kombinatorisch.

6.4 Berechnung der Position und abgeleiteter Parameter

Die Position, die im Preprozessor berechnet wird, ist der zentrale Wert aus dem alle weiteren Parameter abgeleitet werden. Diese werden verwendet, um die lineare Regression auszuführen. Die Position wird gebildet, indem die drei involvierten Datenpunkte, ähnlich der Schwerpunktsbildung, miteinander verknüpft werden und nach einem Korrekturschritt die Position repräsentiert. Es stellt somit die Inversion der sog. *Pad Responce Function* (PRF) dar [2]. Das Prinzip der Positionsbestimmung beruht darauf, dass sich durch die Verteilung der Influenzladung auf drei Pads, die Position des Zentrums der Ladung einfach berechnen läßt. Die

Ladungsverteilung auf den drei beteiligten Pads hat eine annähernd gausförmige Form, so dass die tatsächliche Position durch den Zusammenhang $f=A_l/A_c - A_r/A_c$ berechnet werden kann. Die Werte A_l und A_r sind die gefilterten Amplitudenwerte der benachbarten Kanäle. Der Wert A_c ist der Amplitudenwert des zentralen Pads. Anschließend wird mit einer 128x5 Bit Look Up Table (LUT) die Nichtlinearität ausgeglichen. Die Positionsbestimmung liefert einen 8 Bit breiten Wert (VZ + 7 Bit Position), was mit einem Diskretisierungsabstand von ca. 30 µm korrespondiert. Außerdem wird ein Qualitätsmaß gebildet, das angibt, ob die Werte zu verwenden sind. Erfüllt das Qualitätsmaß die Bedingung: $A_l/A_c * A_r/A_c < TH$ wird die Position zur weiteren Verarbeitung verwendet, andernfalls wird der errechnete Wert verworfen. Der Wert TH stellt eine konfigurierbare Schwelle dar.

Mit der errechneten Position werden die restlichen Werte für die lineare Regression berechnet. Dabei repräsentiert die berechnete Position den Wert in y-Richtung und die Ziffer des Diskretisierungswertes die x-Koordinate. Der Wert x_i wird mit einem Zähler erzeugt, der startet, sobald die Verarbeitung des Preprozessors beginnt. Die weiteren Werte sind x_i*y_i, x_i^2 und y_i^2. Die Produkte werden mit Standardelementen der Synopsys Bibliothek [53] implementiert. Alle errechneten Parameter, die im nachfolgenden Fit Register akkumuliert werden sind in Tabelle 9 dargestellt:

Parameter	Breite	Erklärung
Σx_i	9 Bit	Timebin
Σx_i^2	14 Bit	Quadrat Timebin
Σy_i	14 Bit	Position
$\Sigma x_i y_i$	17 Bit	Produkt aus Timebin und Position
Σy_i^2	21 Bit	Position zum Quadrat
Σ HC	5 Bit	Anzahl der Hit in einem Kanal
Σ Q	17 Bit	Summe der ADC Werte, die Hit Bed. erfüllen
n	5 Bit	Kanalnummer

Tab. 9: Parameter eines Wortes im Fit Register

Die in der obigen Tabelle dargestellten Werte werden im FIT Register akkumuliert, um nach erfolgter Akquisition die Daten zur Berechnung des Geradenfits dem MIMD Prozessor zur Verfügung zu stellen. Abbildung 45 zeigt die zweite Stufe in einem Blockdiagramm.

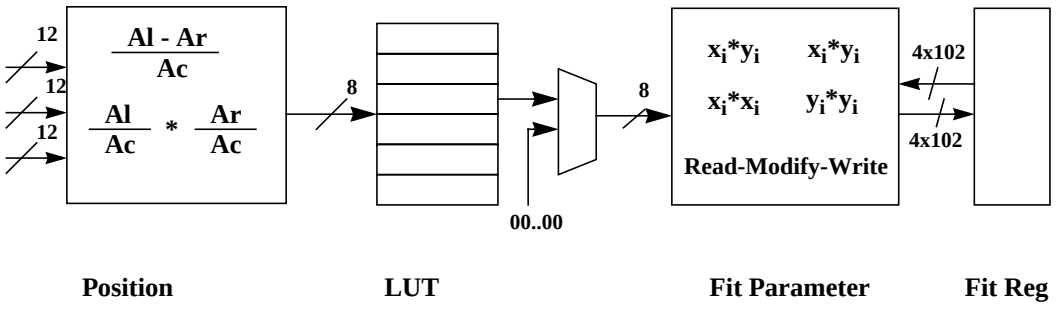


Abb. 45: Blockdiagramm der zweiten Stufe des Preprozessors

Die Blöcke zeigen Einheiten zur Berechnung der Position mit anschließender Korrektur durch eine LUT. Angeschlossen ist die Einheit zur Berechnung der Fit Parameter, die im Fit Register File akkumuliert werden

6.5 Fit Register File

Im Fit Register File werden die Werte des Preprozessors akkumuliert. Für jeden Eingangskanal gibt es eine Speicherzeile. Dabei korrespondiert die Speicherzeile mit der Kanalnummer des Eingangs. Der Summenspeicher verfügt über vier Lese- und vier Schreibports. Seitens des MIMD Prozessors verfügt der Speicher über acht Leseports, was durch eine Zwischenspeicherung von Speicherzeilen in Latches erfolgt. Dadurch kann jede CPU innerhalb eines Taktes zwei Werte des FIT Registers miteinander assoziieren. Intern ist der Speicher aus 19x102 Bit aufgebaut, der durch Flip-Flops implementiert ist. Während des Betriebes des Preprozessors werden mit jedem Takt vier Zeilen ausgewählt, dessen Daten ausgelesen werden, akkumuliert mit den gerade berechneten Werten und auf die nächste steigende Taktflanke zurückgeschrieben werden. Sobald die Verarbeitung durch den Preprozessor abgeschlossen ist, werden die maximal vier besten Datensätze durch die sog. *Tracklet Candidate Select Logik* (TCSL) ausgewählt, so dass der MIMD Prozessor aus diesen Werten selektieren kann. Abbildung 46 zeigt den Aufbau des Summenspeichers.

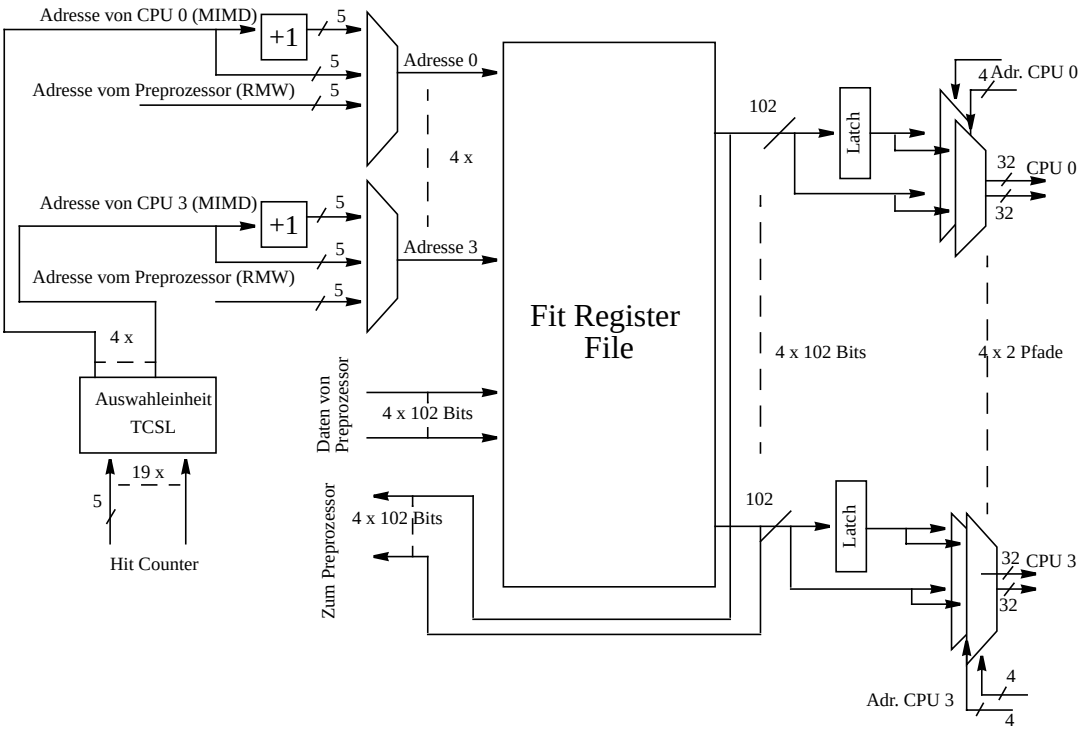


Abb. 46: Architektur des Fit Register

Intern verfügt der Speicherbaustein über vier Lese- und vier Schreibports. Um dennoch nach außen für den MIMD Prozessor acht Leseports zu haben, ist die oben gezeigte Architektur implementiert. Nach erfolgter Akquisition der Daten liefert die TCSL einen 19 Bit Vektor, der die ausgewählten Kanäle bitweise anzeigt. Eine Steuereinheit schaltet die vier ausgewählten Speicherzeilen auf die Ausgänge, so dass an beiden Ports die Werte anliegen. Dann wird das Latch geschaltet, so dass die Werte für diesen Port fest stehen. Anschießend wird der Adresszähler inkrementiert, wodurch der Folgekanal an dem zweiten Ausgang anliegt. Die vier CPU's können nun auf die zwei Datensätze zugreifen.

Da der Preprozessor vier Berechnungs-Pipelines enthält, kann jede CPU dann in den sechzehn Werten des ausgewählten- und des nächsten Kanals adressieren. Abbildung 47 zeigt die Schnittstelle mit dem MIMD Prozessor und dem Preprozessor:

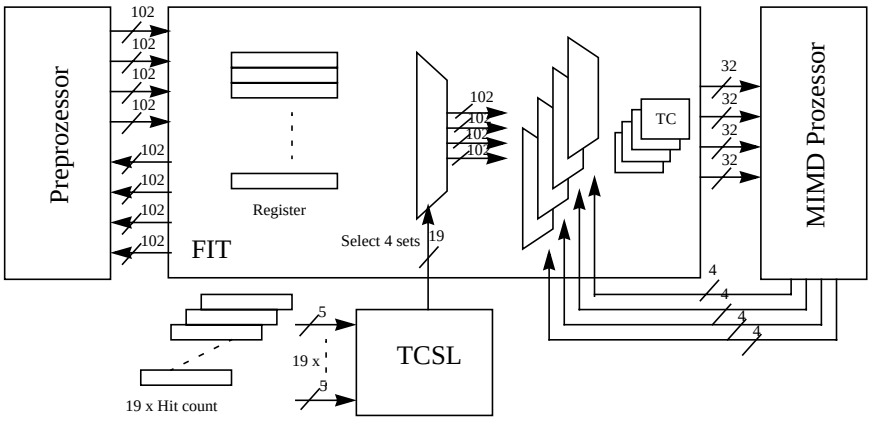


Abb. 47: Schnittstelle zwischen Preprozessor und MIMD Prozessor

Von Seiten des MIMD Prozessors wird die Schnittstelle wie ein Registersatz adressiert. Jeder CPU wird von der Auswahleinheit des Fit Registers ein Registersatz, bestehend aus zwei Speicherzeilen, übergeben. Jede CPU kann dementsprechend aus 16 Wörtern selektieren. Zur weiteren Verarbeitung werden die Werte vorzeichenerweitert.

6.6 Die Auswahleinheit

Die Auswahleinheit wählt vier der neunzehn Datensätze des Fit Registers aus und verteilt sie als fest adressierbare Datensätze auf die vier CPU's des MIMD Prozessors. Als Auswahlwert wird die Summe der *Hit Counter*[1] von je zwei benachbarten Kanälen verwendet. Selektiert werden die Datensätze von den vier Kanälen, welche während der Driftzeit am häufigsten die beschriebenen Hit-Bedingungen erfüllten und somit den größten Wert haben. Da die Regression auch für Spuren ausgeführt werden soll, die sich über zwei Kanäle erstrecken, besteht ein ausgewählter Datensatz aus dem Kanal und seinem rechten Nachbarn. Damit die beiden richtig beteiligten Kanäle ausgewählt werden, muss zusätzlich der linke Nachbar des ausgewählten Kanals einen Wert des Hit Counters besitzen, der über einer einstellbaren Schwelle liegt ($HC > th_{leftneighbor}$). Dieser Schwellenwert ist konfigurierbar und befindet sich im Adressbereich des globalen Busses. Das Ergebnis der Auswahl ist ein 19 Bit breiter Vektor, der angibt, welche Datensätze ausgewählt werden (bitweise kodiert), um weiter durch den MIMD Prozessor verarbeitet zu werden. Die Steuereinheit des Preprozessors schaltet daraufhin diese vier Datensätze auf den Ausgang des Fit

1. Bei dem Wert „Hit Counter“ handelt es sich um einen akkumulierten Wert, der bei der Verarbeitung im Preprozessor auftritt. Er repräsentiert die Anzahl, wie oft die Auswahllogik diesen Kanal für eine Berechnungspipeline ausgewählt hat.

Registers, aus denen der MIMD Prozessor seine Daten, wie aus einem Registersatz, auswählt. Werden weniger als vier Datensätze ausgewählt, wird anstatt der Kanalnummer, die Bestandteil des Datensatzes ist, ein Vektor der nur aus Einsen besteht an den MIMD Prozessor übergeben.

Die Eingangsstufe der Auswahllogik besteht aus 18 Addierern, welche die Summen von jeweils zwei Kanälen bilden und 18 Komparatoren, die entscheiden, ob der Wert des linken Nachbarn größer ist, als die beschriebene Schwelle $th_{leftneighbor}$. Ferner sind 18 Multiplexer integriert, die entweder die Summe oder einen Nullvektor, insofern die Bedingung nicht erfüllt ist, an die Auswahllogik übergeben. Daran angeschlossen sind Komparatoren, die überprüfen, ob der Wert der eintreffenden Daten eine Mindestschwelle übersteigen. Dies wird parallel für alle Schwellenwerte des Spektrums [th_{min}, th_{max}] überprüft. Die Ergebnisse werden verwendet, um ausgehend von th_{max} genau diejenigen Kanäle auszuwählen, in denen maximal vier Werte über dem gerade getesteten Wert liegen. Durch weitere Komparatoren wird für alle Schwellenwerte überprüft, ob ein, zwei, drei, vier oder mehr als vier Kanäle die Schwelle übertrafen. Dann wird mit Hilfe von Prioritätsencodern die Schwelle ausgewählt, bei der vier Kanäle die Schwelle überschreiten. Kann das für keine Schwelle erreicht werden, wird eine Schwelle ausgewählt, bei der mindestens drei Kandidaten die Schwelle übersteigen, bzw. zwei oder ein Kandidat. Letztendlich werden die vier höchsten Werte selektiert. Dabei werden die beschriebenen Operationen innerhalb eines Taktzyklusses ausgeführt. Das Ergebnis ist das bitweise in einen 19 Bit breiten Vektor kodierte Ergebnis, das an den Preprozessor übergeben wird und angibt, welche vier Datensätze dem MIMD Prozessor statisch für die Zeit der Verarbeitung zur Verfügung gestellt werden. Dieser adressiert dann innerhalb dieser ausgewählten Datensätze.

7 *Entwicklungsumgebung des MIMD Prozessors*

Dieses Kapitel beschreibt die Entwicklungsumgebung des MIMD Prozessors. Sie beschränkt sich auf einen Assembler zum Übersetzen des Assembler Codes und einen Simulator, der das Verhalten des MIMD Prozessors auf Register Transfer Ebene emuliert [31].

7.1 Der Assembler

Der entwickelte Assembler übersetzt Programme aus Assembler Notation in Maschinenbefehle, die der Prozessor verstehen kann. Es kann einerseits als Input für den Simulator verwendet werden oder direkt zum Download des Programmes in den Instruktionsspeicher des Prozessors.

Die Übersetzung findet dabei in zwei Schritten statt.

- Dem Laden des Instruktionssatzes aus einer Textdatei (Command-Datei).
- Dem Übersetzen des Quellcodes in Maschinencode.

Der Instruktionssatz des Chips ist nicht in einem Header-File abgelegt, sondern wird als Command-Datei zur Laufzeit eingelesen. Bei dieser Datei handelt es sich um eine durch Tabulatoren getrennte Tabelle, Leerzeichen werden ignoriert. Abgesehen von Kommentaren und Leerzeilen wird ein Befehl je Zeile eingelesen. In den Spalten muss in dieser Reihenfolge folgendes stehen:

- Nr: Die Nummer des Befehls. Sie wird ignoriert, muss jedoch in der Datei stehen. Sie besteht aus einer oder mehren Ziffern.
- Name: Der Name des Befehls, wie er im Quellcode stehen soll. Der Assembler ignoriert Groß- und Kleinschreibung.
- Format: Das Format des Maschinencodes nach dem Opcode. Es besteht aus drei Buchstaben, die angeben, was im Maschinencode an dieser Stelle stehen soll. Mögliche Werte sind zum Beispiel ’R’ für Register, ”” für nichts oder ’8’ für eine 8 Bit breite Zahl. Diese Werte werden ODER verknüpft.
- Opcode: Der 7-Bit Opcode des Befehls in binärer Schreibweise inklusive des Write-Back-Bits.
- Sprungcode: Der Sprungcode des Befehls. Es handelt sich um eine Konstante, die angibt, nach welchem Flag gesprungen wird. Werden diese Daten nicht benötigt, so läßt sich dies durch ’-’ ausdrücken.
- OP1, OP2, OP3: Die erlaubten Ziele für den jeweiligen Operanden. Mögliche Typen sind “PRF”, “GRF”, “FIT”, “CON” und “DIR”, wobei die letzten beiden für das Konstanten Registerfile bzw. die direkte Angabe einer Konstanten im Maschinencode stehen. Die Auswertung erfolgt in der Form, dass nur nach dem Auftreten dieser Zeichenfolgen im Substring gesucht wird.

Abbildung 48 zeigt, wie das entsprechende File aussehen soll:

```
#  Mnemonic  Format   Opcode+WB   Branch   OP1       OP2               OP3
0  NOP       ___      0000000      -        -         -
1  ADD       RRR      1000001      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
2  ADC       RRR      1000011      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
3  SUB       RRR      1000101      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
4  SBC       RRR      1000111      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
5  MUL       RRR      1001001      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
6  MUS       RRR      1001011      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
7  DIV       RR_      1001100      -       PRF,FIT   PRF,GRF,FIT,CON   -
8  DIE       __R      1001111      -       -         -                 PRF,GRF
9  AND       RRR      1010011      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
10 ATT       RR_      1010010      -       PRF,FIT   PRF,GRF,FIT,CON   -
11 ORR       RRR      1010101      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
12 COM       _RR      1011111      -       -         PRF,GRF,FIT,CON   PRF,GRF
13 NEG       _RR      1010111      -       -         PRF,GRF,FIT,CON   PRF,GRF
14 EOR       RRR      1010001      -       PRF,FIT   PRF,GRF,FIT,CON   PRF,GRF
15 SHA       4RR      1011011      -       DIR       PRF,GRF,FIT,CON   PRF,GRF
16 SHT       4RR      1011001      -       DIR       PRF,GRF,FIT,CON   PRF,GRF
17 ROR       _RR      1011101      -       -         PRF,GRF,FIT,CON   PRF,GRF
```

Abb. 48: Beispiel eines Instruktionssatz File

Beim Einlesen der Command-Datei werden die Daten in einer einfach verketteten Liste zur weiteren Verwendung gespeichert. Zeilen die mit “#” beginnen sind Kommentare und werden ignoriert. Enthält die Datei fehlerhafte Zeilen, werden diese angemerkt und ignoriert. Derartige Fehler treten z.B. auch dann auf, wenn falsche Format-Typen verwendet werden oder ein 8-Bit Opcode angegeben wird. Es findet, bis auf die Kontrolle auf erlaubte Eingangswerte, wobei unbekannte Ziele der Operanden ignoriert werden, keine Überprüfung der Command-Datei statt.

Die Assemblierung der Quelldatei erfolgt in zwei Durchläufen, wobei der zweite nur dann durchgeführt wird, wenn keine Fehler gefunden wurden. Hierbei können zwei Typen von Fehlern auftreten: Leichte Fehler, die beim Programmieren auftreten, wie z.B. falsche Syntax etc. und schwere Fehler, wie sie in einer fehlerhaften COMMAND-Datei passieren. Bei schweren Fehlern wird der Assembler mit einem Fehler-Code größer als 1 sofort beendet. Bei leichten Fehlern im Quelltext wird die Ausgabedatei gelöscht und das Programm mit dem Fehlercode 1 beendet. Eine Zeile im Quellcode hat folgende Form:

’’[Label:][Befehl [Operand1[,Operand2[,Operand3]]]][; Kommentar]’’

also die gleiche Form, die auch von anderen Assemblern wie z.B. TASM für 80x86 Systeme verwendet wird, mit der Einschränkung, dass hier nur ein Befehl pro Zeile zulässig ist. Im ersten Durchlauf werden die Labels abgearbeitet und ihre Adressen gespeichert, sowie auf doppelte Labels und korrekte Syntax geprüft. Im zweiten Durchlauf erfolgt die Assemblierung. Hierzu wird erst der Befehl aus der Zeile extrahiert und dann mit Hilfe regulärer Ausdrücke weiter in seine Teile wie Namen und Operanden geteilt. Anschließend werden die drei verschiedenen Positionen

nach dem Opcode bearbeitet, deren Belegung in der Command-Datei unter ’’Format’’ angegeben sind. Die nachfolgende Tabelle zeigt die erlaubten Format Spezifizierer.

Parameter	Breite	Erklärung
R	-	Register (PRF, GRF, CON, FIT)
-	0	kein Wert
4	4 Bit	-
8	8 Bit	-
6	16 Bit	-
A	10 Bit	-
B	11 Bit	-
C	12 Bit	-
X	13 Bit	-

Tab. 10: Verwendete Format Spezifizierer

7.2 Der Simulator

Das Programm teilt sich in drei Teile:

- Den Simulator.
- Die davon unabhängige Grafische Benutzer Schnittstelle (GUI).
- Das Hauptprogramm, in dem das Assemblerprogramm mit den Daten geladen wird.

Der Simulator arbeitet in drei Teilen: Aktionen, die innerhalb eines Taktes passieren, wie z.B. das Ausführen der Instruktionen, Aktionen die auf eine Flanke geschehen, wie z.B. das Übernehmen von Registerinhalten und Aktionen die zwischen dem ersten und zweiten Teil passieren, wie das Lesen aus dem RAM und das Ändern der Synchronisationsflags. Das Instruktionswort, welches der Simulator bearbeitet, hat nicht 24 Bit, sondern 32 Bit, wobei die oberen 8-Bit als 0 angenommen werden. Die internen Aufrufe erfolgen über Listen, in die sich die Klassen bei ihrer Erzeugung eintragen. Jede eingetragene Klasse wird dann in der entsprechenden Taktphase aufgerufen. Um ein quasiparalleles Arbeiten der verschiedenen Elemente zu erreichen, arbeiten die Ableitungen des Typs class cycle nur auf gelatchten Variablen, also Variablen, deren Änderung erst beim Aufruf einer Flanke erfolgt. Einzige Ausnahme ist hier der Speicher, der anders gehandhabt wird. Bei dieser Vorgehensweise werden immer alle Register zur Flanke betrachtet um beispielsweise auf Änderungen der Registerinhalte und deren Zustand zu achten. Auch die Pipeline ist in zwei Teile unterteilt und wird für jede CPU einzeln aufgerufen.

Das GUI ist mit Hilfe des GTK-Toolkits [56] geschrieben. Wie in Abbildung 49 dargestellt, besteht das GUI aus 8 Fenstern (hier nur 7 sichtbar): Einem Kommando-Fenster, 4 CPU-Fenstern, einem Fenster für die Daten im GRF sowie einem Fenster für den Speicher, sowie einem ein Bus-Modul (hier nicht zu sehen). Die Hintergrundfarben der Speicher- und Registerzellen haben folgende Bedeutung:

- Pink: Der Wert dieses Registers ist möglicherweise ungültig
- Rot: Der Schreibzugriff im letzten Takt kam aus mehreren Quellen gleichzeitig

- Grün: Der Wert wurde im letzten Takt von einer Quelle geschrieben
- Weiss: Der Wert wurde vor einiger Zeit von einer Quelle aus geschrieben

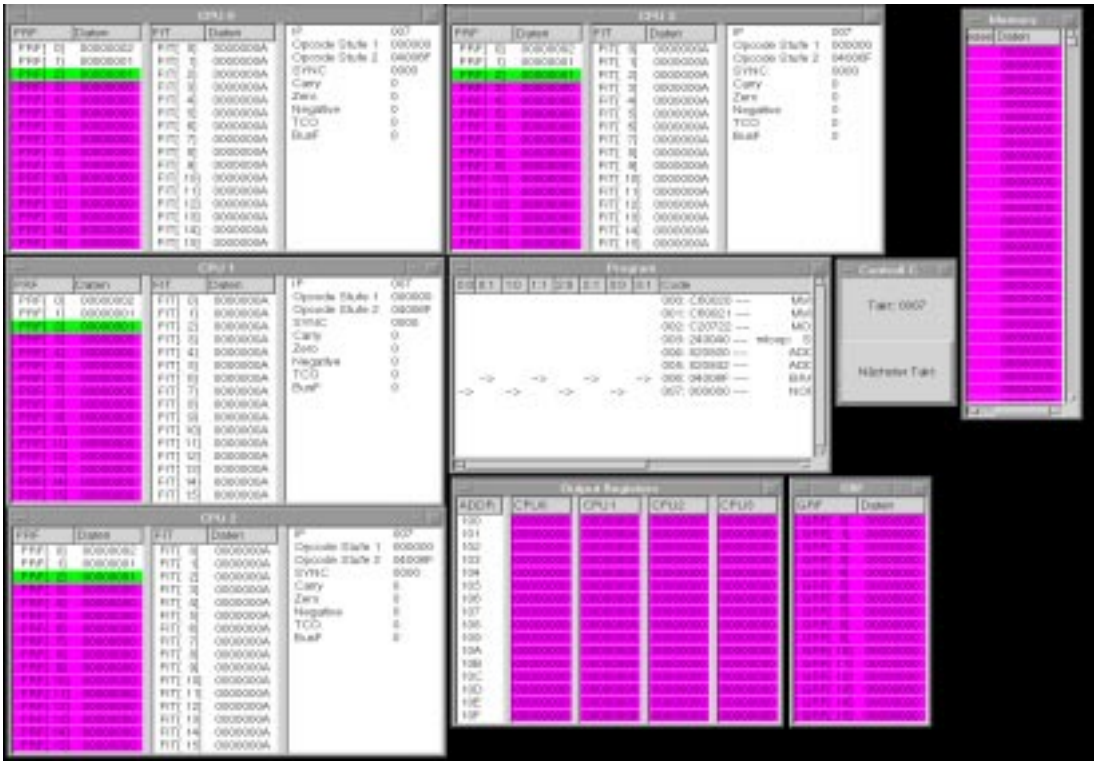


Abb. 49: GUI des Simulator

Die Farben werden nur für die Daten angezeigt und nicht für die Flags, auch wenn es im Fall des Synchronisationsregisters zum Schreiben aus verschiedenen Quellen kommen kann. Dieser Fall muß eindeutig gehandhabt werden und stellt keinen Fehler dar. Das Schließen der Fenster erfolgt über das Schließen des Kommando-Fensters.

8 *Test des MIMD Prozessors*

Dieses Kapitel beschreibt den Test des MIMD Prozessors. Er wurde auf Register Transfer Ebene (RTL), auf Gatterebene mit ModelSim [59] und als Implementierung auf einem FPGA getestet.

Außerdem wird in diesem Kapitel eine Möglichkeit zum Testen der Speichermodule sowie ein Test mit Hilfe des implementierten „Boundary Scan“ Pfades vorgestellt.

8.1 Testprogramm

Für die Simulation wurden verschiedene Testprogramme erstellt. Sie beinhalten alle implementierten Instruktionen mit unterschiedlichen Verarbeitungsdaten. Dabei wurden die Befehle aller Befehlsgruppen (siehe Kap. 4.2) in verschiedenen sinnvollen und sinnlosen Kombinationen ausgeführt. Die Programme wurden aus der mnemonischen Darstellung mit Hilfe des Assemblers in die Binärdarstellung übersetzt. Anschließend wurde das Verhalten des Programms mit Hilfe des Simulators überprüft. Die generierten Befehlsfolgen wurden im Instruktionsspeicher abgelegt. Beim RTL-Test und beim Test auf Gatterebene wurden die Daten in Speichermodulen abgelegt, die das Verhalten der Speicherelemente emulieren. Beim Test mit einem FPGA wurden für den Instruktionsspeicher interne Module verwendet, die mit der mitgelieferten Software initialisiert werden können. In der nachfolgenden Abbildung ist ein Ausschnitt aus einem Testprogramm dargestellt:

```
AND PRF[5],PRF[6],GRF[15];  0x0000 && 0xffff=0x0000
ORR PRF[5],PRF[6],GRF[14];  0x0000 && 0xffff=0xffff
COM PRF[6],GRF[13];         !0xffff=0x0000
NEG PRF[6],GRF[12];         !0xffff=0x0001
EOR PRF[6],PRF[5],GRF[11];  0x0000^0xffff=0xffff
SHA 2,PRF[5],GRF[10];       0xffff sha 2=0xffff
SHT 2,PRF[5],GRF[ 9];       0xffff <<2=0xfffc
SHT -2,PRF[5],GRF[ 9];      0xffff <<2=0x3fff
ROR PRF[2],GRF[ 0];         0x0002 >>2=0x0001
```

Abb. 50: Ausschnitt aus dem Testprogramm

8.2 Simulation des Hardwareentwurfs

Für den RTL-Test des MIMD Prozessors wurden wesentliche Einheiten des Prozessors einzeln gestestet. Dafür wurden individuelle Testbenches entworfen und die Module mit sinnvollen Eingangsstimuli beschaltet. Die Überprüfung der Ergebnisse wurde teilweise mit Hilfe des

Waveform Viewers vorgenommen oder die Antworten des Designs wurden automatisch durch die Testbench überprüft.

Zu Beginn der RTL-Simulation wurden die Instruktionsspeicher initialisiert. Startwerte der Interrupt Controller und der Konstanten-Speicher werden mit dem Prozessor über den globalen Bus in die Einheiten geschrieben. Der Instruktionsspeicher wird durch ein Verhaltensmodell ersetzt. Nach Resetfreigabe beginnt die Initialisierungsprozedur. Hier werden die Module des globalen Busses beschrieben. Über die Konfigurationseinheit werden danach der Instruktionsspeicher und der Datenspeicher beschrieben. Dann wird dem Prozessor durch einen Interrupt die Startadresse mitgeteilt, der daraufhin das Programm ausführt. Die Kontrolle findet im Timingdiagramm des Waveform Viewers statt oder kann durch das Endergebnis eines Test-Algorithmusses überprüft werden. Anschließend wurden die Hierarchieebenen unterhalb des Toplevels auf ihre korrekte Funktionalität überprüft.

Für die Simulation auf Gatterebene ist der Entwurf in eine Netzliste übersetzt und in VHDL-Code zurückgeschrieben worden. Die Testbenches aus dem vorhergehenden Lauf wurden wiederverwendet. Die Kontrolle der Ausgangssignale der Module erfolgte auf dem jeweiligen Toplevel. Für diesen Testdurchlauf fand das bisherige Testprogramm erneut Verwendung. Die Antwortzeiten der angehängten Speichermodule sowie der Systemtakt der Testumgebung wurde dabei den Syntheseergebnissen angeglichen. Eingestellt wurde ein Systemtakt von 120 MHz, was der angestrebten Zielfrequenz für die ausgewählte Technologie entspricht. Die Kontrolle fand erneut im Waveform Viewer statt. Diesmal ausschließlich im Toplevel. Die Simulation lieferte erwartungsgemäß die gleichen Ergebnisse. Die bisherigen Simulationsergebnisse konnten somit bestätigt werden.

8.3 Test mit einem FPGA

Der Test auf einem FPGA wurde mit Hilfe eines Altera FPGA’s des Typs Apex 20K1000E durchgeführt. Die Synthese wurde mit dem FPGA Compiler II der Firma Synopsys durchgeführt, das Abbilden auf das FPGA mit dem Programm Quartus von Altera.

Der Testaufbau besteht aus einer CPU, einem internen Speichermodul mit getrenntem Lese- und Schreibport (256x32 Bit), der die Funktion des Datenspeichers emuliert, einem internen Speichermodul mit zwei getrennten Ports (16x32 Bit), der für das Globale Register File verwendet wird, sowie einem ROM, worin das Programm gespeichert ist. Die Konstanten (CONST) werden fest verdrahtet implementiert. Zudem wird ein VGA Modul verwendet, mit dessen Hilfe die Aktionen des Prozessors visualisiert werden können. Dafür muss per Instruktion in den Videospeicher geschrieben werden, der den Wert direkt auf den Bildschirm darstellt.

Für den Test wurde eine Taktfrequenz von 12,5 MHz gewählt. Zu Beginn wird der Prozessor initialisiert, d.h der Instruktionsspeicher wird mit dem Programm beschrieben. Anschließend wird die Aktivität an den Prozessor übergeben, der selbständig alle weiteren Aktionen ausführt. Abbildung 51 zeigt die verwendeten Module:

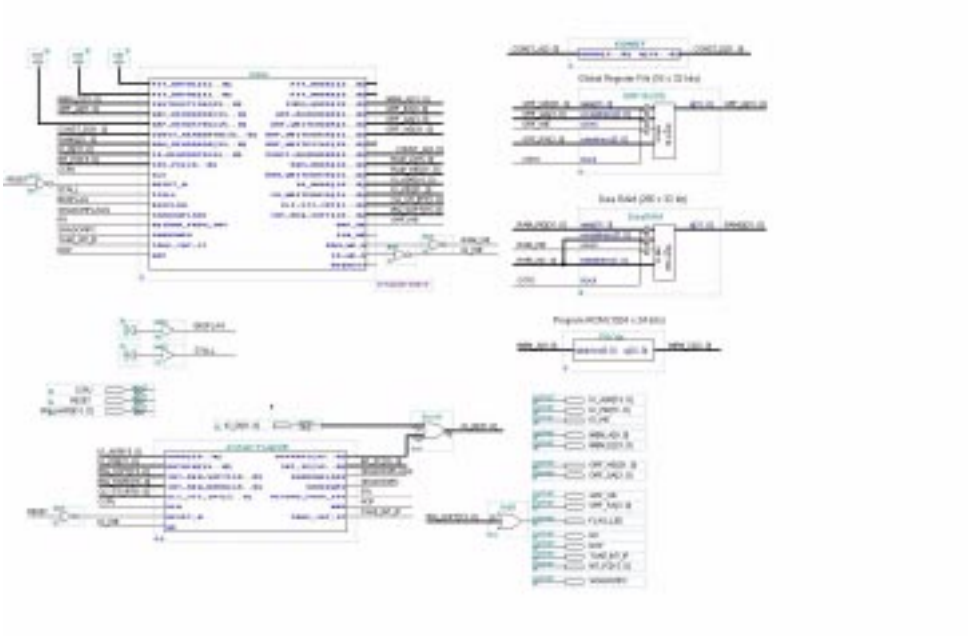


Abb. 51: Testaufbau einer CPU

Als Testprogramme wurden verschiedene Programme verwendet, sowie ein Programm, womit auf dem Prozessor das Spiel “Pac Man” gespielt werden kann. Der Test lief mehrere Tage ohne Fehler. Damit konnte die Funktionalität der VHDL Beschreibung nachgewiesen werden. Abbildung 52 zeigt ein Foto der Bildschirmausgabe sowie das FPGA Testboard.

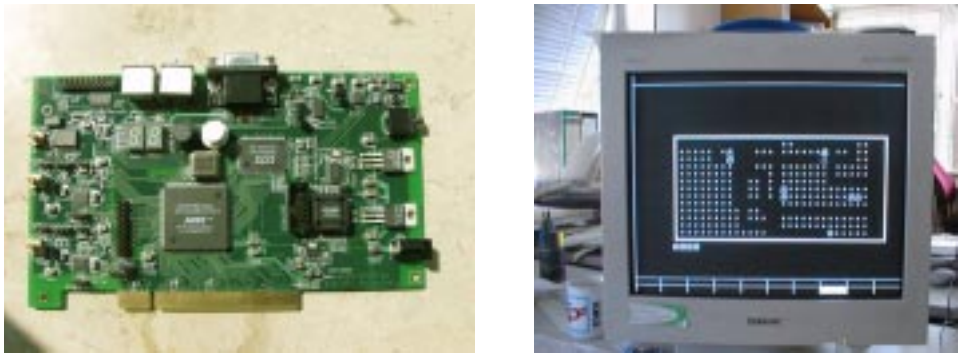


Abb. 52: ACEX-FPGA Testboard und Bildschirmausgabe von “Pac Man”

Der MIMD Prozessor wurde ebenfalls auf Register-Transfer- und Gatter-Ebene getestet [18]. Außerdem konnte der Gesamtentwurf auf ein FPGA des Typs Altera APEX 20kE1000 getestet werden. Dabei wurden die Speichermodule wieder durch interne Module ersetzt, die das Verhalten der QPM emulieren. Insgesamt konnte die volle Funktionalität aller vier CPU’s des MIMD Prozessors gemeinsam getestet werden. Insbesondere wurden die Datenpfade zum Beschreiben und zum Auslesen des Instruktions- und Datenspeichers sowie des globalen Busses durch die Konfigurationseinheit getestet.

8.4 Test eingebetteter Speichermodule

Moderne Mikroprozessorsysteme beinhalten in den meisten Fällen große Speicherbereiche für Instruktionen und Daten. Aufgrund der hohen Packungsdichte besteht hier die größte Wahrscheinlichkeit von Fabrikationsfehlern innerhalb des Chips. Demnach beeinflussen die internen Speicherzellen stark die Ausbeute (engl. *Yield*) der Herstellung. Deshalb werden zumeist redundante Speicherreihen und/oder Spalten eingebaut. Der Test wird meistens durch Selbsttestmodule realisiert, welche die korrekte Funktionalität feststellen und ggf. den Fehlerort lokalisieren. Der am meisten verbreitetste Algorithmus wird als MARCH Algorithmus bezeichnet, der alle Stuck-at Fehler auf Zellebene, Fehler durch Kopplung und Fehler des Adressdecoders detektieren kann. Der Algorithmus kann in Pseudocode folgendermaßen beschrieben werden [11].

```
for (address=0 to address=last) {write data}
for addressSwep = forward to AddressSwep = reverse {
     for (data = 0 to data = 1) {
        for (address = first to address = last)
          {Read data; write ~ data;} // Read and compare. Write comple.
     }
}
```

Im Wesentlichen laufen alle Testvarianten darauf hinaus, den Speicher mit Nullen und Einsen zu beschreiben, um dann den Wert auszulesen und ihn mit dem Erwartungswert zu vergleichen.

Für den Test der Speichermodule innerhalb des MIMD Prozessors kann diese Prozedur durch eine CPU bzw. durch die Konfigurationseinheit durchgeführt werden. Der Instruktionsspeicher wird durch die oben beschriebene Konfigurationseinheit mit Nullen bzw. Einsen initialisiert. Dann werden die Speicherstellen wieder zurückgelesen und verglichen. Der Datenspeicher kann analog durch eine der vier CPU's getestet werden. Werden fehlerhafte Speicherstellen gefunden, wird dies dem Benutzer durch die Konfigurationseinheit mitgeteilt werden. Die entsprechenden Speicherstellen werden für den weiteren Programmverlauf aus dem Speicher ausgeblendet. Einzelne Bitfehler werden von der Power-Management-Einheit gezählt. Sie werden durch das Hamming-Verfahren korrigiert und müssen nicht unbedingt von der Verwendung ausgeschlossen werden.

8.5 Test mit Boundary Scan

Der Test nach dem IEEE 1149.1 Standard ist integraler Bestandteil des Tests auf Boardlevel Niveau. Ziel des Boundary Scan Designs ist der Test der Verbindungen auf Board Level Niveau sowie der Initialisierung von Verbindungsknoten mit Testwerten. Somit stellt der Boundary Scan Test die Weiterentwicklung des sog. "In Circuit Test" dar, bei dem die IC's mit Hilfe von Nadelkarten getestet wurden. Beim BSD Test werden statt dessen die äusseren Nadeln durch sog. "Silicon Nails" ersetzt. Abbildung 53 [35] zeigt schematisch die Integration des Boundary Scan Pfades mit der zusätzlichen Steuerlogik und das Zustandsdiagramm des TAP Controllers.

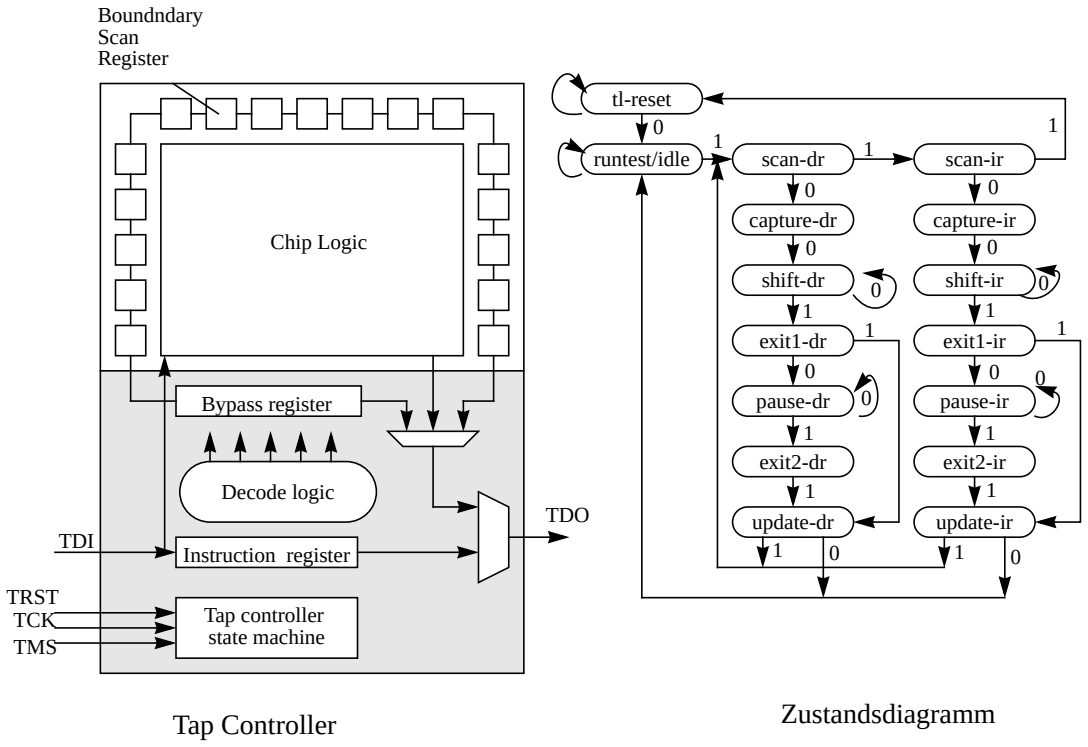


Abb. 53: Boundary Scan Logik mit Kontrolleinheit nach IEEE 1149.1

Die Abbildung zeigt die Registerzellen, die um die primären Eingangs- und Ausgangspins des Chips gelegt sind und den „Test Access Port“ (TAP), der den Ablauf des Tests steuert. Das dargestellte Instruktionsregister (IR) beinhaltet die Test-Instruktion und selektiert die Datenregister für den seriellen Zugriff und kontrolliert die Testfunktionen. Die Adressierung der Datenregister erfolgt also durch die vorher in das Instruktionsregister eingeschobene Instruktion. Das selektierte Datenregister wird anschließend als Schiebepfad zwischen die TDI- und TDO-Pins geschaltet. Das Bypass Register ist ein 1 Bit Schieberegister, welches einen Schiebepfad durch den TAP zur Verfügung stellt. Falls mehrere BSD-Pfade getestet werden sollen, kann dadurch der Pfad zwischen TDI und TDO auf die Länge 1 verkürzt werden. Der TAP Controller ist ein Zustandsautomat, welcher den Zugriff auf das Instruktionsregister und die Datenregister kontrolliert und somit die Testfunktionen steuert. Er kann 16 verschiedene Zustände annehmen, die den unterschiedlichen Betriebsarten entsprechen. Der Übergang von einem Zustand in den nächsten erfolgt immer mit der steigenden Flanke von TCK, wobei der jeweils gültige Pegel an TMS einen der beiden möglichen Folgezustände auswählt. Die gesamte Boundary Scan Struktur ist gemäß dem IEEE-Standard als Synchronschaltung implementiert. Nach “Power On” oder “Reset” wird der TAP Controller in den Reset-Zustand gebracht, was eine Deaktivierung der Boundary Scan Struktur zur Folge hat.

Die für den Testbetrieb wichtigsten Zustände des TAP Controllers sind das Ein- bzw. synchrone Ausschieben der Testdaten (*shift-dr*), das Übernehmen von Daten in die Schiebekette (*capture-dr*), das Übergeben der Testdaten an die Ausgangs-Latches der Testzellen (*update-dr*) sowie die

analogen Zustände für das Instruktionsregister (*shift-ir, capture-ir, update-ir*). Zusammenfassend kann die Architektur folgendermaßen beschrieben werden:

- Parallele Übernahme von Testvektoren in die Boundary Scan Zellen (sample).
- Serielles Einschieben und simultanes Herausschieben übernommener Testvektoren (shift).
- Paralleles Anlegen von eingeschobenen Testvektoren an zu prüfende Schaltungsteile (update).
- Test/Stimulierung der inneren Schaltung (Internal Test).
- Test/Stimulierung der äußeren Signale, die am DUT anliegen (External Test).

Boundary Scan-Zellen sind in diversen Ausführungen möglich, je nachdem, ob lediglich getestet (Capture-Zellen) oder auch stimuliert werden soll. Eine universelle Zellarchitektur ist in Abbildung 54 [35] dargestellt. Das Masterlatch wird durch das Broadcast-Signal CLOCK DR getaktet und nimmt entweder Daten von der vorhergehenden Scanzelle auf (Schieberegisterbetrieb) oder tastet den Eingang ab. Das Shadowlatch übernimmt die Daten vom Masterlatch, wenn diese nach dem Schieben stabil anliegen. Der Pegel des MODE-Signals bestimmt dann, ob diese an den Zellausgang transferiert werden (Testmode) oder nicht (Betriebsmode). Universalzellen sind sowohl an einem Eingangspin, als auch einem Ausgangspin einsetzbar.

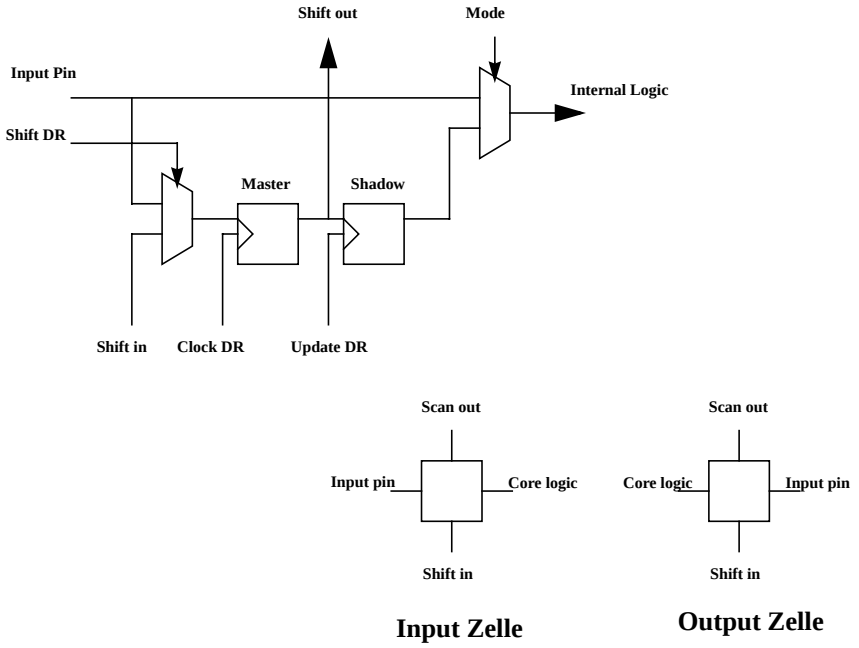


Abb. 54: Universelle Boundary-Scan Zelle

Steuerleitungen wie MODE, SHIFT DR, UPDATE DR und CLOCK DR liegen gleichzeitig als Broadcast-Signale an allen Scanzellen an. Daher sind auch immer alle Zellen im gleichen Funktionszustand, d.h. alle Zellen, die in der Lage sind, Eingangsdaten zu sampeln, tun dies

simultan mit der steigenden Flanke von CLOCK DR im TAP-Zustand *capture-dr*. Analog dazu erfolgt die Ausgabe eingescannter Testdaten an allen dazu fähigen Scanzellen mit der fallenden Flanke von UPDATE DR im TAP-Zustand *update-dr*. Gesteuert vom Signal MODE, sind alle Scanzellen entweder im Betriebsmode oder im Testmode. Dabei entscheidet immer die aktuell eingescannte Instruktion darüber, welcher Mode vorliegt. Die hier dargestellte universelle Zelle unterstützt die Modi *capture* und *update*.

Zusätzlich zu der bisher erläuterten Funktionalität bietet der Standard einige weitere Instruktionen, die für unterschiedliche Zwecke verwendet werden können. Das sind im Wesentlichen:

BYPASS: Das Bypass Register ist zwischen TDI und TDO involviert (Betriebsmodi).

SAMPLE: Das Boundary Scan Register ist zwischen TDI und TDO involviert (Betriebsmodi).

EXTEST: Das Boundary Scan Register ist zwischen TDI und TDO involviert (Testmodus).

Für einen Test werden die verschiedenen BSD Pfade entweder in Parallel-, Ketten oder einer Mischform betrieben. Bei der Kettenschaltung werden die TDI und TDO Ports in einer Reihe betrieben, was den geringsten Routing Aufwand hat. Parallele Schaltungen haben den Vorteil, dass bei einer defekten Scan Kette die weiteren Pfade unabhängig voneinander betrieben werden können. Der MIMD Prozessor und die Netzwerkschnittstelle [1], [39] besitzen zwei unabhängige BSD Pfade die innerhalb des Prototypen Trap 1 in einer Kette angeordnet sind. Da der MIMD Prozessor Verbindungen zu allen wichtigen Funktionseinheiten besitzt, kann im Bedarfsfall das Verhalten von einzelnen Komponenten durch den BSD Pfad emuliert werden, so dass Fehler innerhalb des Prototypen gefunden werden können.

9 *Designflow*

Dieses Kapitel beschreibt den Designflow des MIMD Prozessors von der Beschreibung in VHDL bis zum Layout. Eingegangen wird auf die Ergebnisse der Hardwaresynthese und die Besonderheiten, die beim Layout des Prozessors zu beachten sind.

9.1 Hardwaresynthese

Die Synthese des MIMD Prozessors wurde mit dem Design Compiler von Synopsys [51] parallel zum Entwurf der Module durchgeführt. Das Augenmerk war dabei grundsätzlich auf eine möglichst kompakte Architektur gerichtet. Kritische Komponenten, wie der Multiplizierer und der Dividierer, wurden separat synthetisiert. Die Synthese wurde „bottom up“ durchgeführt und mittels Skripte gesteuert. Als minimale Taktfrequenz wurden 120 MHz als Timing Constraint gesetzt. Dieses Constraint mußte erreicht werden und stellt die Zielfrequenz für die verwendete 0,18 µm Standardzellentechnologie dar. Das Constraint einzelner Submodule wurde deutlich kürzer bemessen, da nicht alle Ausgänge einzelner Submodule durch Register getrieben werden und somit sich der kritische Pfad aus mehreren Modulen zusammensetzt.

Der VHDL-Entwurf des MIMD Prozessors ist technologieunabhängig. Abgebildet wurde der Entwurf auf eine 0.18 µm CMOS Standardzellen-Bibliothek mit sechs Lagen Metall und auf eine Altera FPGA Technologie. In der nachfolgenden Tabelle sind die Ergebnisse der Hardwaresynthese für die ausgewählten Zieltechnologien dargestellt:

Modul	Standardzellen	Flip Flops	Fläche	Altera Logikzellen
Pipe 1	436	109	0,018 mm^2	393
Pipe 2	12111	616	0,37 mm^2	8384
ALU	9656	97	0,27 mm^2	6889
CPU	12806	725	0,40 mm^2	8779
MIMD	53712	10185	1,20 mm^2	36965

Tab. 11: Charakteristische Synthesegrößen

Die obige Tabelle zeigt die Komplexität des Entwurfs. Die implementierte ALU jeder einzelnen CPU nimmt erwartungsgemäß den größten Anteil am Entwurf ein. Innerhalb dieses Moduls benötigt der Multiplizierer etwa 25 % der integrierten Zellen. Durch ihn wird die Multiplikation von 32 Bit Daten in Hardware realisiert. Wird anstatt einer parallelen Multiplikation ein sequentielles Verfahren verwendet, läßt sich der Entwurf durch einen minimalen Hardwareaufwand implementieren. Die Verarbeitungszeit für die Multiplikation ist dann aber ungleich höher (z.B. 32 Takte für einen Restoring Algorithmus) und kommt für die angestrebte Anwendung nicht in Frage.

Kritisch, bezüglich dem gesetzten Timing, ist ebenfalls der Multiplizierbaustein innerhalb der ALU. Er besitzt bei der gewählten Technologie eine Antwortzeit, die für einen Baustein des Typs „Carry-Save-Array“ - was einer Standardimplementierung entspricht - bei 10 ns liegt. Ein Geschwindigkeitsgewinn ist durch die Implementierung schneller Multiplizierer möglich. Deshalb wurde für die Multiplikation das Verfahren nach „Wallace“ (siehe Kap. 4.5.3.1) gewählt. Der implementierte Multiplizierer hat einen kritischen Pfad von 3,6 ns. Dabei belegt er etwa genausoviel Chipfläche wie das „Carry-Save Array“-Modul.

Für den Toplevel des Entwurfs detektiert Synopsys einen kritischen Pfad, wonach ein Systemtakt von max. 140 MHz möglich ist. Das Timingverhalten des MIMD Prozessors wird mit Hilfe des Timinganalysetools Pearl [60] überprüft. Es liefert quasi die gleichen Ergebnisse wie der Design Compiler von Synopsys. Besonderes Augenmerk war auf Werte zum Erzeugen der Leseadresse der Speicherbausteine gelegt, da diese Adressen spätestens zur fallenden Taktflanke stabil an den Speichern anliegen müssen. Da die Verzögerungszeit der Verbindungsleitungen nicht berücksichtigt werden konnte, sollte der Pfad in der Summe den Wert von 8 ns nicht überschreiten, um ein korrektes Verhalten zu gewährleisten. Analog wurde das Verhalten der Instruktionsspeicher überprüft. Tabelle 12 zeigt die kritischen Pfade des Designs und die gemessenen Werte des jeweiligen Pfades.

Modul	kritischer Pfad	Signallaufzeit	Erklärung
Instruktionsspeicher	clk -> addres out	ca. 2,5 ns	von steigender Flanke bis Gültigkeit der Adresse
Datenspeicher	clk -> address out	ca. 2,5 ns	von steigender Flanke bis Gültigkeit der Adresse
Toplevel	alle Pfade	ca 7 ns	Kritischster Pfad im Prozessor

Tab. 12: Timing Analyse kritischer Pfade

Eine Timing-Analyse unter Berücksichtigung der Verzögerungszeiten der Verbindungsleitungen war nicht möglich, da die parasitären Kapazitäten nicht extrahiert werden konnten. Eine detailierte Analyse des Timing-Verhaltens ist daher erst direkt am Prototypen möglich.

9.2 Integration des Boundary Scan Pfads

Das Einfügen des Boundary Scan Pfads erfolgt mit Hilfe des Test Compilers [51], der die notwendigen Elemente zur Verfügung stellt. Gesteuert wird der Vorgang über Skripte, die auf dem Toplevel nach der Hardwaresynthese angewendet werden. Die Implementierung ist durch das Aussparen von einigen Eingangs- und Ausgangspins nicht mehr konform mit dem IEEE Standard, was aber generell keine Einschränkung darstellt. Die BSD Zellen werden nicht in den Toplevel des Gesamtdesigns eingefügt, sondern vielmehr in zwei der Submodule (MIMD Prozessor und die Netzwerkschnittstelle). Diese voneinander unabhängigen BSD-Logiken werden in Kette geschaltet. Dementsprechend erhält der MIMD Prozessor die Chip Identität 0 und das Netzwerk Interface die Identität 1. Als Manufactory ID wird der Wert 0x0 gewählt, als Part-Number der Wert 0x7461.

9.3 Floorplan und Layout

Das Layout des MIMD Prozessors wurde mit Silicon Ensemble durchgeführt. Als Technologie wurde eine 0,18 µm CMOS Standardzellentechnologie der Firma UMC verwendet. Dabei beansprucht das Design bei einer Packungsdichte (engl. *Row Utalisation*) von 74 % eine Chip Fläche von 3,79 mm^2, was die geschätzten Werte des Design Compilers nicht bestätigt. Ein kompakteres Design war nicht möglich, da sich ansonsten die Standardzellen nicht mehr verdrahten (engl. *routen*) ließen. Insgesamt beinhaltet der Block 53712 Standardzellen, wovon 10185 Flip-Flops sind. Der MIMD Prozessor beinhaltet dabei sechs Clocktrees. Der Skew sollte möglichst den Wert von ca. 150 ps nicht überschreiten, was für alle Taktbäume erreicht werden konnte. Als Clocktree-Elemente wurden spezielle Treiber und Inverter verwendet, die sich durch ein ausgewogenes Zeitverhalten auszeichnen. Um zu vermeiden, dass es zu sog. “Voltage Drops” kommt, wurden zusätzlich vertikale Verbindungen für VDD und VSS eingefügt. Abbildung 55 zeigt das Layout des MIMD Prozessors, welches in das Layout des Prototypen Trap 1 eingebettet ist.

Dieser beinhaltet den Preprozessor, den MIMD Prozessor, die Netzwerkschnittstelle, den Instruktionsspeicher und den Datenspeicher. Zusätzlich wurden Prototypen eines neuentwickelten ADC’s integriert, die insgesamt fünf Eingangskanäle des Preprozessors mit Daten versorgen.

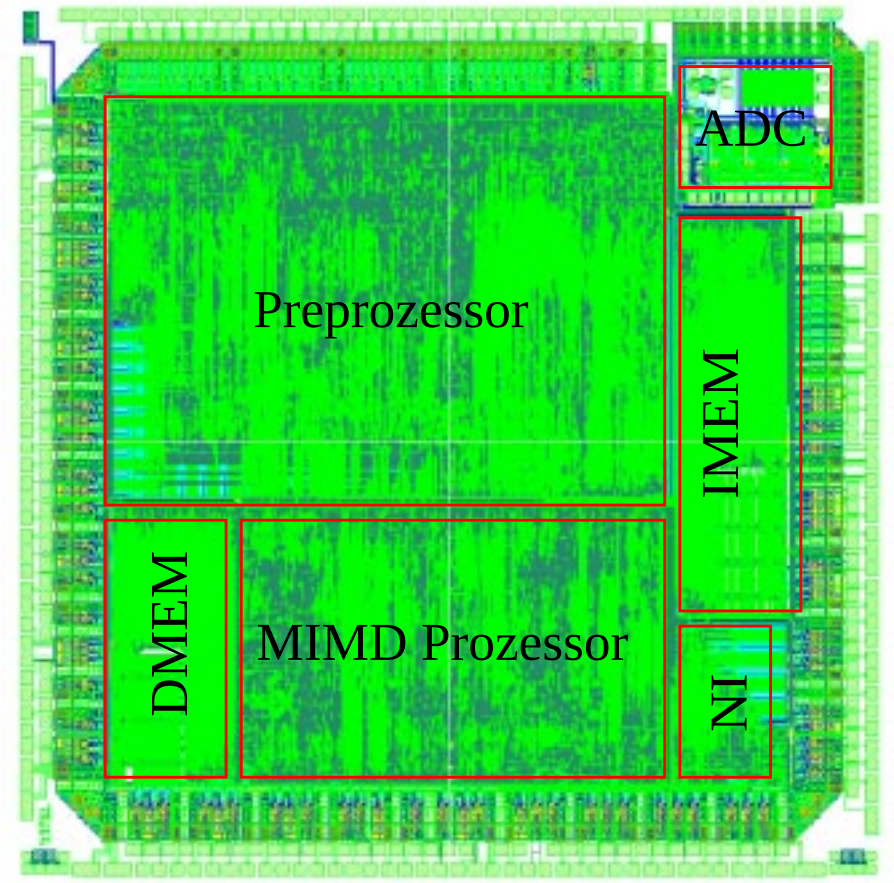


Abb. 55: Floorplan des Prototypen Trap 1

Problematisch bei dem Gesamtlayout sind die zum Teil schmalen Verdrahtungskanäle, mit denen die einzelnen Blöcke verbunden werden müssen. Ein sinnvolles Pinout der Makroblöcke ist deshalb unumgänglich, da ansonsten das Verbinden der einzelnen Blöcke unmöglich ist. Dementsprechend werden die Pins der Submodule so angeordnet, dass sie sich möglichst Punkt zu Punkt gegenüberstehen.

Der Clocktree wurde auf Block Ebene von Hand aufgebaut. Als Treiber wurden spezielle Elemente verwendet, die trotz ihrer starken Treiberleistung eine minimale Gatterlaufzeit besitzen. Die Äste des Baumes wurden so aufgebaut, so dass am Rootpin des jeweiligen Submoduls die gleichen Verzögerungszeiten auftreten. Die Buffer werden innerhalb der Verdrahtungskanäle zwischen den Submodulen so plaziert, dass die Wege zu den einzelnen Modulen etwa die gleiche Länge besitzen. Innerhalb der Submodule wurde für kritische Einheiten (die beispielsweise Schieberegister enthalten) ein minimaler Pfad von 400 ps eingefügt. Das wird erreicht, indem Buffer bei der Synthese eingebaut werden. Das Problem eines großen Clock Skews kann dadurch weitestgehend eliminiert werden.

10 *Zusammenfassung*

In der vorliegenden Arbeit wurde ein MIMD Prozessor mit Vorverarbeitungseinheit konzeptionell entworfen und realisiert. Beide Einheiten sind primär für die Datenverarbeitungskette des TRD Detektors des ALICE Experiments vorgesehen. Mittels des TRD soll in 6 µs festgestellt werden, ob ein Elektron-Positron-Paar den Detektor durchflogen hat oder nicht. Ferner muss zwischen Elektronen und Pionen unterschieden werden. Der Detektor besteht aus 1,2 Millionen Datenkanälen, welche Datenwerte mit einer Frequenz von 10 MHz liefern. Aufgeteilt ist der Detektor in Gruppen zu je 18 Datenkanälen, wobei zusätzlich die Daten von drei weiteren Kanälen der angrenzenden Nachbarn eingekoppelt wird, so dass ein Datenaustausch zwischen den Gruppen vermieden werden kann. Ingesamt verarbeitet jede Gruppe des Auslesesystems also die Daten von 21 Kanälen.

Die Datenwerte stammen aus Driftkammern, welche mit einer Frequenz von 10 MHz über einen Zeitraum von 2 µs Werte liefern. Die Daten stammen von geladenen Teilchen, welche entlang ihrer Flugbahn im Gas der Driftkammer Elektronen-Cluster deponieren. Diese driften entlang eines elektrischen Feldes zu einem Kathodenpad und influenzieren dort eine Ladung, die ausgelesen wird und zur Spurrekonstruktion in der r-ϕ-Ebene des Detektors verwendet wird. Dabei werden nur diejenigen Spuren berücksichtigt, welche aufgrund ihres hohen Transversalimpulses eine maximale Breite von zwei benachbarten Pads überstreicht.

Zur Spurrekonstruktion muss aus den Daten der Driftkammern die genaue Position der Ladungs-Cluster in ϕ-Richtung berechnet werden, um daran mittels einer linearen Regression eine Gerade zu fitten. Die zweite Koordinate wird durch den zeitlichen Diskretisierungswert bestimmt. Die Unterscheidung zwischen Elektronen und Pionen wird durch eine Übergangsstrahlungssignatur vorgenommen, die vom Radiator der Driftkammer herrüht. Sie resultiert in einem großen Amplitudenwert im letzten Drittel der Driftzeit über mehrere Timebins. Deshalb werden die Amplitudenwerte über die Driftzeit akkumuliert.

Von der zur Verfügung stehenden Verarbeitungszeit von 6 µs fallen 2 µs für die Driftzeit der Elektronen-Cluster in der Driftkammer an. Diese steht fest. Für die Berechnung der linearen Regression sind insgesamt 1,8 µs vorgesehen, da die Ergebnisse noch über ein Netzwerk verschickt und verarbeitet werden müssen. Deshalb werden während der Driftzeit die Summen innerhalb der Regressionsformel berechnet. Die übrigen Operationen werden nach der Driftzeit ausgeführt. Ferner sind weitere Operationen und Algorithmen notwendig. Insgesamt stehen nach der Driftzeit noch 1,8 µs zur Verfügung, um alle weiteren Operationen auszuführen und die Ergebnisse an die Netzwerkschnittstelle zu übergeben.

Deshalb wurde ein Preprozessor entwickelt, der die Vorverarbeitung der Driftkammerdaten übernimmt und die Summen der linearen Regression während der Driftzeit berechnet. Die Summen werden durch Akkumulation der berechneten Werte gebildet und in einem speziellen Speicher abgelegt. Der Preprozessor führt eine digitale Filterung der Eingangsdaten durch und

wählt für jeden Abtastwert der ADC's maximal vier Kanäle aus, aus deren Eingangswerten die Postion des Elektronen-Clusters in der Padebene berechnet wird. Dann werden die weiteren Werte der Regression berechnet und in einem speziellen Speicher akkumuliert.

Aus der Analyse der physikalischen Randbedingungen des Experiments ergab sich, dass nicht mehr als vier hochenergetische Teilchen in einer Gruppe aus 18 Kanälen vorkommen. Deshalb muss die Einheit, welche dem Preprozessor folgt, die Daten von vier Spuren verarbeiten. Pro Datensatz sind etwa 150 Standardinstruktionen auszuführen.

Als Verarbeitungseinheit wurde eine MIMD Architektur gewählt, womit die vier Datensätze durch vier speichergekoppelte RISC CPU's mit verschiedenen Algorithmen verarbeitet werden. In der Entwicklung dieser Architektur liegt der Schwerpunkt der Arbeit. Der Datenaustausch mit dem Preprozessor findet über eine Registerkopplung statt, so dass Transferzyklen vermieden werden können. Die Kommunikation der CPU's untereinander geschieht über ein globales Register File einen Vier-Port-Datenspeicher und einen globalen Bus. Durch die enge Kopplung und den drei Kommunikationskanälen, kann jede CPU mit jeder anderen schnell Daten austauschen. Der Zugriff auf den Vier-Port-Speicher und den globalen Bus geschieht über Lade- und Speicherbefehle, wobei der Zugriff auf den Bus durch einen Arbiter reglementiert wird. Zur Synchronisation der CPU's ist ein spezieller Mechanismus implementiert, der nach dem Barrieren Prinzip funktioniert.

Der MIMD Prozessor wurde aus vier CPU's aufgebaut, da somit bei einer niedrigen Frequenz von 120 MHz gearbeitet werden kann. Die daraus resultierende Registerarchitektur kommt mit einer zweistufigen Pipeline aus, so dass Datenabhängigkeiten vermieden werden können. Würde statt dessen ein Einprozessorsystem verwendet werden, müsste mindestens mit der vierfachen Frequenz das System betrieben werden. Berücksichtigt man auftretende Pipelinehazards, ist eine Systemfrequenz von 600 MHz notwendig, was aus Sicht der Implementierung für ein Standardzellen-Design ein Problem darstellt.

Die RISC CPU verarbeitet einen Instruktionssatz, der aus 72 Instruktionen besteht. Der Datenpfad der CPU ist 32 Bit breit. Alle Prozessorregister und Datenspeicher haben ebenfalls diese Datenwortbreite. Die implementierte ALU verarbeitet 16 Instruktionen auf Festkommadaten. Für die Multiplikation wurde ein Verfahren nach Wallace gewählt, für die Division ein sog. Radix-4 Verfahren. Alle ALU-Operationen können in einem Takt verarbeitet werden. Für die Division sind 18 Takte notwendig.

Die Simulation des Entwurfs erfolgte auf Register-Transfer- und Gatterebene sowie als Implementierung auf einem FPGA. Als Testprogramm wurden verschiedene Testprogramme verwendet. Die Funktionalität des MIMD Prozessors konnte auf diese Weise auf allen Entwurfsebenen vollständig nachgewiesen werden.

Der Entwurf erfolgte in der Hardwarebeschreibungssprache VHDL. Die Synthese wurde mit Hilfe des Design Compilers durchgeführt. Für eine 0,18 µm Standardzellentechnologie mit sechs Lagen Metall (UMC0.18) detektierte Synopsys einen Chipflächenbedarf von 1,2 mm^2. Dabei ist eine maximale Systemfrequenz von ca. 140 MHz möglich. Insgesamt beinhaltet der Entwurf etwa 53000 Standardzellen. Zur Verifikation des Designs wurde der Entwurf ebenfalls auf ein Feld-Programmierbares Gate-Array (FPGA) abgebildet, wobei dann ca. 37000 Logikzellen notwendig

sind. Mit diesem FPGA sind Systemfrequenzen von ca. 6 MHz möglich. Hinsichtlich der beanspruchten Chipfläche ist der verwendete 32 Bit Multiplizier-Addierbaustein als kritisch zu bewerten. Dabei belegt er etwa ein Drittel der benötigten Systemressourcen, sowohl für die Standardzellenvariante als auch für den FPGA Entwurf. Das Layout des MIMD Prozessors verbraucht etwa 3,79 mm^2 bei einer Utilization von etwa 74 %.

Als Entwicklungsumgebung wurde ein Simulator und ein Assembler entwickelt, mit denen die Programme des MIMD Prozessors übersetzt und getestet werden können. Die grafische Benutzer-Schnittstelle ermöglicht eine komfortable Kontrolle.

Der MIMD Prozessor sowie der Preprozessor wurden mit den anderen Einheiten des Prototypen Trap 1 innerhalb eines MPW-Runs in einer 0,18 µm Standardzellen Technologie erfolgreich gefertigt. Prototypen des Preprozessors und des Multi-Port-Memories wurden bereits zu einem früheren Zeitpunkt submittiert und getestet. Die volle Funktionalität der Prototypen konnte in verschiedenen Tests nachgewiesen werden, was am Ende dieser Dokumentation aufgezeigt wurde.

A *Prototypen*

Dieses Kapitel beschreibt zwei der entwickelten Prototypen. Sie ebnen den Weg zu dem Prototypen Trap 1.

A.1 FaRo I

FaRo I ist ein Prototyp des Preprozessors, der in einer 0.35 µm CMOS Technologie mit drei Lagen Metall (AMS0.35) gefertigt wurde. Er verarbeitet die Daten von acht analogen Datenkanälen, wobei er einen festverdrahteten Algorithmus ausführt, der dem des Preprozessors sehr nahe ist. Zwei weitere 9 Bit Datenwerte der angrenzenden Nachbarn werden eingekoppelt, da sie für die Berechnung der Position der außen liegenden Kanäle benötigt werden. Die Verarbeitung endet in einem Fit Register (ähnlich der in Kap. 6.5 dargestellten Architektur). Der Prototyp arbeitet mit zwei Frequenzen, nämlich einem 80 MHz und einem 20 MHz Taktsignal, wobei das zweite aus dem ersten abgeleitet wird.

Der implementierte hardverdrahtete Algorithmus hat die Aufgabe, die Regressionsparameter zu berechnen. Die Ergebnisse werden für jeden Kanal separat in Registern akkumuliert, die über eine Netzwerkschnittstelle ausgelesen werden können. Die Verarbeitung ist in mehrere Pipelinestufen unterteilt. Die implementierten Stufen haben die folgenden Aufgaben:

- Demultiplexen der Eingangsdaten.
- Auswahl der Daten für die Positionsberechnung.
- Speichern der Eingangsdaten.
- Berechnung der Position.
- Generierung der abgeleiteten Werte für die lineare Regression.
- Akkumulierung der berechneten Werte im Fit Register.

Die drei letzten Einheiten sind Bestandteil von zwei identischen Verarbeitungspipelines, die pro Taktzykluss jeweils die Daten eines Kanals verarbeiten können. Die Auswahl der Werte findet im zweiten Block statt. Der erste Block verteilt die Daten eines Dateneingangs auf zwei Datenkanäle. Abbildung 56 zeigt den Prototypen auf Blockebene:

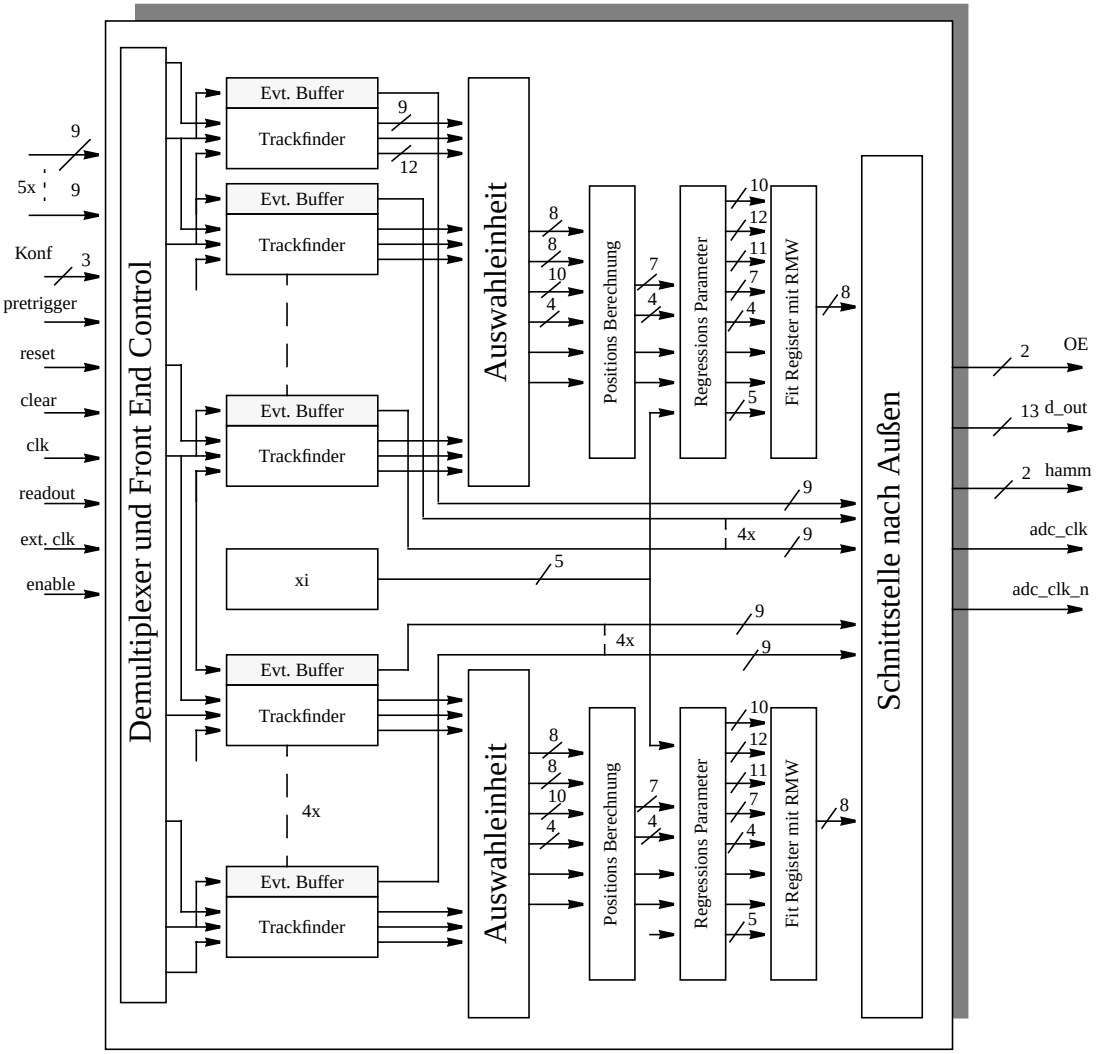


Abb. 56: Blockdiagramm von FaRo I

Das Blockdiagramm zeigt die verschiedenen Stufen innerhalb des Prototypen. Der erste Block (Demultiplexer) verteilt die Eingangswerte auf die Verarbeitungskanäle. Diese bestehen aus den Event Buffern, welche die Eingangswerte speichern und den Einheiten, welche die Datensätze auswählen, die in den beiden Pipelines weiter verarbeitet werden. Die Auswahllogik selektiert aus vier Datensätzen denjenigen Kanal aus, der die größte Amplitudensumme besitzt. Für die Werte dieses Kanales werden die Position (y_i) und die weiteren Werte für den linearen Fit berechnet. Diese werden dann im Fit Register akkumuliert. Danach können alle Werte über eine Schnittstelle ausgelesen werden. Die Initialisierung von FaRo I wird über eine serielle Schnittstelle durchgeführt, mit der alle Konfigurationswerte sequentiell beschrieben werden. Die Verarbeitung der Daten durch den Prototypen geschieht auf ein Startsignal hin (Pretrigger). Anschließend arbeitet FaRo I für eine vorher einstellbare Anzahl an Taktzyklen, was mit der Driftzeit der Elektronen korrespondiert.

Die folgenden Kapitel beschreiben die Architektur und die Funktion der oben aufgeführten Blöcke. Es wird die Architektur auf makroskopischer Blockebene beschrieben. Falls notwendig, werden ausgesuchte Implementierungsdetails dargestellt. Die Funktion jedes einzelnen Blockes wird zusätzlich beschrieben. Ferner wird auf den Designflow und den Test des Prototypen eingegangen.

A.1.1 Eingangsstufe und Auswahleinheit

Die Eingangsstufe des Prototypen versorgt die angeschlossenen Stufen mit Daten. Dafür werden die Werte von einem zehn Bit breiten Eingangsbus auf zwei Datenverarbeitungskanäle verteilt. Die Steuerung erfolgt über die ‘Output Enable’ Leitungen, die direkt an die Eingänge der ADC’s gelegt werden. Zusätzlich ist in dieser Stufe eine sog. Pedestal Korrektur eingebaut, welche einen konstanten Offset von den Eingangswerten abzieht. Dadurch kann der Nullpunkt der Eingangswerte korrigiert werden. Die Architektur ist in Abbildung 57 dargestellt:

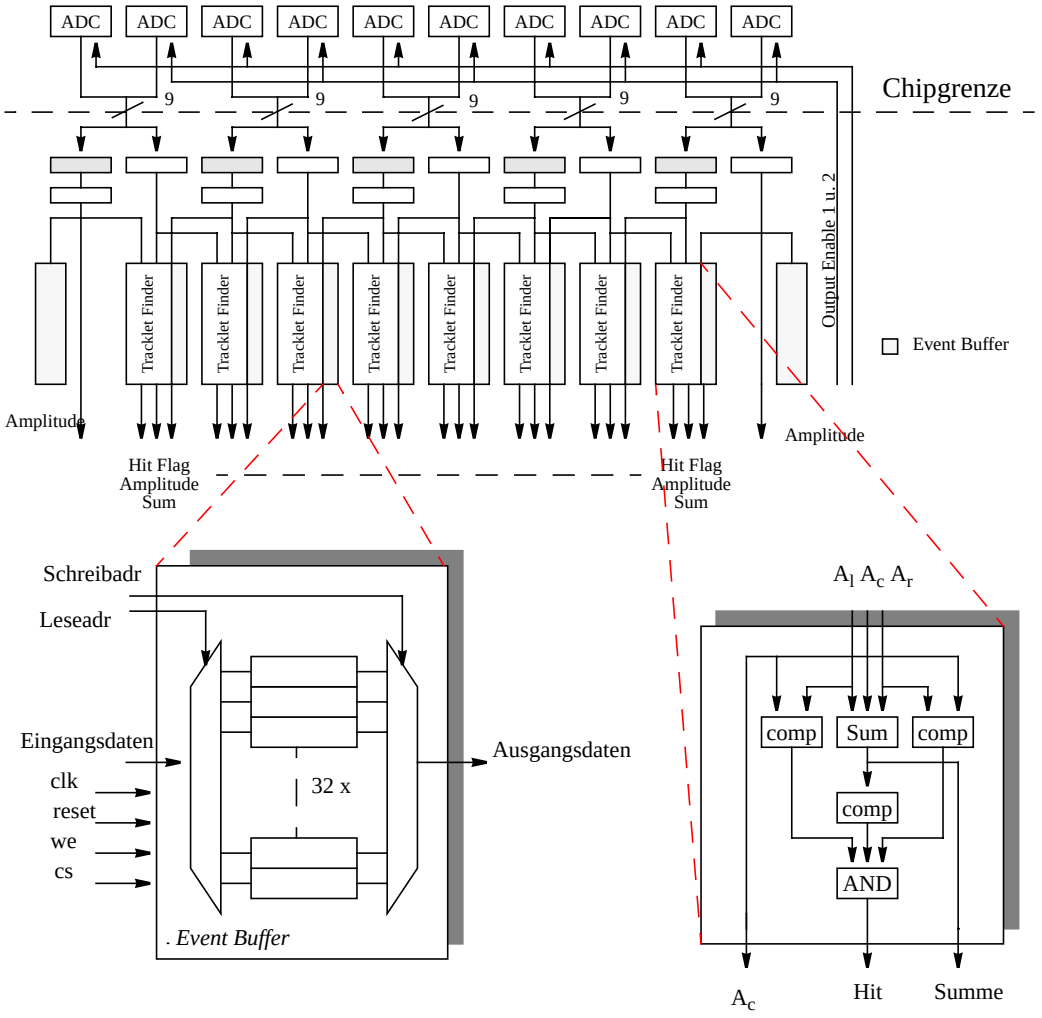


Abb. 57: Architektur der Eingangsstufe

Zwei ADC’s treiben abwechselnd einen Eingangsbus für zwei Kanäle. Die Steuerung erfolgt über die ‘Output Enable’ der ADC’s. Die grau schattierten Register werden mit fallender Taktflanke betrieben, so dass die angeschlossenen Module mit jeder steigenden Taktflanke ein neues Datum der ADC’s erhalten.

Je Zweiergruppe übernimmt der linke Kanal mit fallender Taktflanke die Werte in ein temporäres Register, um zur steigenden Taktflanke den Wert an ein nachgeschaltetes Register zu übergeben, was gleichzeitig mit der Übernahme des linken ADC-Wertes passiert. Daran angeschlossen sind die Subtrahierer, die das Pedestal von dem Eingangswert individuell abziehen (nicht dargestellt). Die Eingangsdaten werden an einen Block, der als *Tracklet Finder* bezeichnet ist, übergeben, der auswählt, ob ein Wert für die Weiterverarbeitung verwendet wird. Damit die Daten eines Kanals für die weitere Verarbeitung berücksichtigt werden, müssen die drei folgenden Bedingungen erfüllt sein:

- Die Amplitude des linken Nachbarn muss kleiner oder gleich sein als der betrachtete Kanal.
- Die Amplitude des rechten Nachbarn muss kleiner sein als der betrachtete Kanal.
- Die Summe der drei beteiligten Amplituden muss größer sein als eine einstellbare Schwelle.

Diese drei Bedingungen werden über Komparatoren ausgewertet und logisch UND verknüpft. Die Summe wird mit zwei Addierern gebildet. Der Auswahleinheit wird das Ergebnis der Auswertung übergeben (1 Bit). Diese wählt aus vier Datenkanälen denjenigen aus, der die größte Summe besitzt und zusätzlich die beiden Bedingungen erfüllt. Für diesen Kanal wird in der nächsten Stufe die Position berechnet. Dafür wird der Einheit die drei Amplitudenwerte, die Summe und die Kanalnummer übergeben. Vorher werden die Werte in Registern zwischengespeichert.

Ferner beinhaltet diese Stufe insgesamt 10 Event Buffer, worin die bereits Pedestal korrigierten Eingangswerte gespeichert werden. Damit kann nach erfolgter Ausführung des fest verdrahteten Algorithmus das Ereignis, welches zu den vorliegenden Werten geführt hat, rekonstruiert werden.

Die Event Buffer sind als Zwei Port Speicher auf Basis von Flip-Flops implementiert. Jedes Speichermodul hat eine Breite von 9 Bit und eine Tiefe von 32 Einträgen. Die Speichermodule haben ein synchrones Verhalten beim Schreiben und ein asynchrones beim Lesen.

Während des Betriebes wird der Speicher vom Eintrag 0x00 beginnend mit den Eingangswerten des entsprechenden Kanals gefüllt. Dafür wird mit dem Startsignal (Pretrigger) ein fünf Bit Zähler zurückgesetzt, der mit jedem langsamen Taktsignal den Wert um ’1’ inkrementiert. Nach erfolgter Verarbeitung befinden sich alle Eingangswerte in den entsprechenden Event Buffern und können sequentiell ausgelesen werden.

A.1.2 Positionsberechnung

Die Position wird aus folgendem Zusammenhang gebildet

$$pos = \exp(\log(A_l - A_r) - \log(A_r + A_l + A_c))$$

und repräsentiert den Schwerpunkt der drei beteiligten Amplitudenwerte. Die Division wird durch Logarithmieren des Dividenden und Divisors mit anschließendem Exponentieren vermieden. Diese drei Funktionen werden durch LUT’s implementiert. Die Summe der Amplitudenwerte (Divisor) wurde bereits in der letzten Stufe berechnet. Die Subtraktion der Amplitudenwerte der benachbarten Kanäle wird in dieser Stufe durchgeführt. Intern arbeitet dieses Modul mit der

vierfachen ADC-Frequenz, so dass trotz der drei Pipelinestufen, die Division in einem ADC-Takt ausgeführt wird. Aus der Position und dem mitgeführten Wert x_i werden die weiteren Fit Parameter berechnet. Das sind:

- $x_i * y_i$: Produkt aus Position (y_i) und diskretisiertem Zeitwert (x_i)
- y_i^2: Quadrat der Position
- x_i^2: Quadrat des Zeitwertes

Implementiert werden diese Operationen mit Standardelementen der Synopsys Bibliothek. Die Ergebnisse werden zusammen mit den weiteren Werten N (Anzahl der Zeitwerte, welche die Eingangsbedingungen erfüllen) und TRD (Neunte Bit des Eingangswertes) an das Fit Register übergeben, das die Werte mit den Ergebnissen der vorangegangenen Taktzyklen akkumuliert.

A.1.3 Fit Register File

Im Fit Register File werden die Werte der Berechnung akkumuliert. Ingesamt beinhaltet diese Einheit 14 Speicherblöcke für die benötigten Werte der linearen Regression. Diese sind als Zwei-Port Speicher aufgebaut mit synchronen Schreib- und asynchronen Leseverhalten. Während jeden Taktes werden zwei Speicherstellen selektiert, die mit den eingekoppelten Werten akkumuliert werden. Die eintreffenden Werte werden, falls erforderlich vorzeichenerweitert und auf die Breite der gespeicherten Werte gebracht. Erfüllt kein Eingangswert die Hitbedingungen, werden im ersten bzw. fünften Speicherplatz Nullen akkumuliert. Ansonsten wird mit jedem Takt ein Wert aus jedem Speicher ausgelesen, akkumuliert und auf die nächste steigende Taktflanke zurückgeschrieben. Abbildung 59 zeigt diesen Block in einer Übersicht:

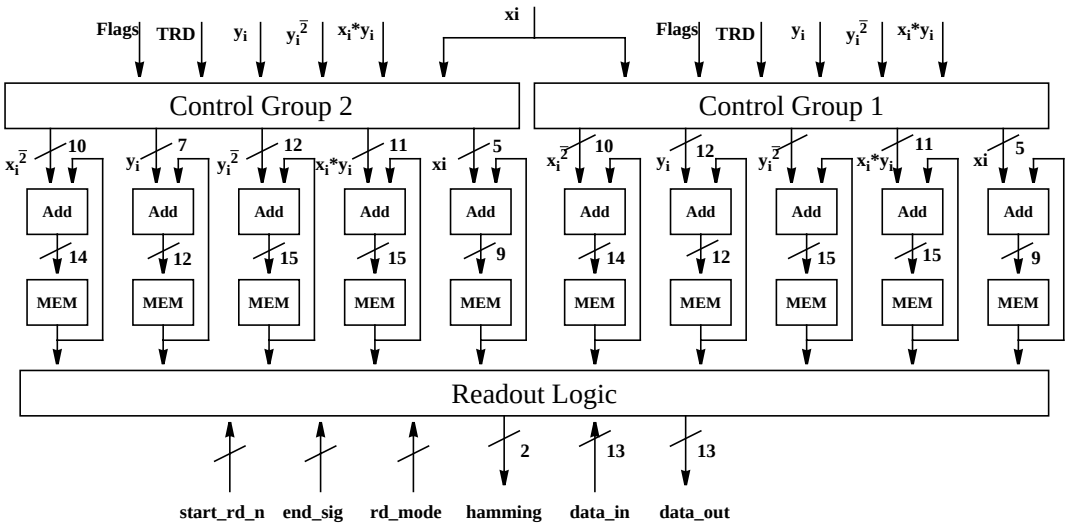


Abb. 58: Blockdiagramm des Fit Register Files

Im Blockschaltbild sind nicht die Speicherelemente dargestellt, welche die akzeptierten Hits die in dem jeweiligen Kanal aufgetreten sind sowie die MSB des ADC Wertes, welcher als TR Signatur interpretiert wird.

A.1.4 Auslese des Prototypen

Der Prototyp kann über einen Datenbus in verschiedenen Modi ausgelesen werden. Das sind:

- Alle Event Buffer mit aufsteigender Reihenfolge
- Das Fit Register File
- Die Konfigurationsparameter
- Die drei genannten hintereinander

Der Auslesemodus wird durch zwei Bits gesteuert, welche die letzten Stellen der Konfigurationsbits in der Kette der Konfigurationsworte sind (siehe oben). Der Vorgang der Auslese erfolgt über eine Schnittstelle die wahlweise die Daten mit oder ohne Hamming-Kontrollbits übergibt. Der Vorgang der Auslese wird über einen separaten Auslesetakt gesteuert. Die Auslese wird über ein spezielles Startsignal gesteuert, das, falls gesetzt, mit dem nächsten Takt beginnt, die Daten dem Auslesemodus entsprechend an die Schnittstelle zu legen. Dabei sind in den 13 Bit breiten Ausgabebus 8 Bit Daten enthalten.

Die Schnittstelle ist dabei so aufgebaut, das mehrere Chips in einer Reihe (Daisy Chain) betrieben werden können. Dafür besitzt die Schnittstelle neben dem Ausgangsport noch einen 13 Bit breiten Eingangsport. Dem letzten Chip der Daisy Chain wird über ein Eingangsflag (end_sig) mitgeteilt, dass dieser der letzte dieser Reihe ist. Ansonsten verläuft die Auslese analog zum Auslesen eines einzelnen Chips. Das Startsignal wird hierbei an alle Chips verteilt (Broadcast). Am Ende der Übertragung wird eine Endsignatur geschickt, so dass der folgende Chip einer Kette mit dem Senden beginnt.

A.1.5 Konfiguration des Prototypen

Die Konfiguration wird über eine drei Bit breite Schnittstelle vorgenommen, wobei die Daten seriell über eine ein Bit breite Leitung übertragen werden. Die weiteren Signale sind das Taktsignal, das ausschließlich die Speicherelemente des Konfigurationspfades mit dem Takt versorgt und ein Reset Signal, um diese Speicherelemente zurückzusetzen. Konfiguriert wird immer der gesamte Chip, d.h. alle Parameter müssen nacheinander in den Chip geschoben werden. Alle Register, die Konfigurationswerte beinhalten, werden dafür in einem großen Schieberegister angeordnet, wobei das letzte Bit innerhalb dieser Kette den Schiebevorgang stoppt, sobald eine ’1’ in das Flip-Flop geschoben wird. Die folgenden Werte werden durch die Einheit konfiguriert:

- Driftlänge (5 Bit): Anzahl der Taktzyklen, die der Prototyp ab dem Startsignal laufen soll.
- Pedestal (10 x 9 Bit): Offset, der vom eingehenden Wert abgezogen wird.
- Threshold (8 x 11 Bit): Schwellenwert, der von der Amplitudensumme überschritten werden muss, damit der Kanal bei der Verarbeitung berücksichtigt wird.
- Auslesemodus (00:= Alles, 01:= Konfiguration, 10:= Event Buffer, 11:= Fit Register).

A.1.6 Designflow

Die Simulation des Prototypen erfolgte auf RTL und Gatter Ebene und wurde mit ModelSim durchgeführt. Dabei konnte das korrekte Verhalten des Prototypen auf beiden Entwurfsebenen nachgewiesen werden. Die Eingangsstimuli wurden mit Hilfe eines C-Programms erzeugt,

welches dem funktionalen Verhalten des Prototypen entspricht. Damit konnten sinnvolle, wie auch zufällige Eingangswerte generiert werden. Das C-Programm liefert zusätzlich die Erwartungswerte. Der Simulationsablauf verläuft automatisch, d.h. die Testbench liest die Stimuli aus einer Datei ein, übergibt diese an das Design und vergleicht die Antwort mit den Erwartungswerten. Treten Fehler auf, werden diese in einer weiteren Datei gespeichert, so dass am Ende der Simulation der Fehler zurückverfolgt werden kann. Insgesamt wurde das Design auf beiden Entwurfsebenen mit einigen Millionen Testvektoren ohne Fehler getestet.

Der rein digitale Prototyp wurde mit dem Design Compiler synthetisiert. Die Synthese wurde über Skripte gesteuert und „bottom up“ durchgeführt. Als Zieltechnologie wurde eine 0,35 um CMOS Technologie mit drei Lagen Metall gewählt (AMS0.35). Als Zielfrequenz wurden 20 MHz gewählt. Der Dividierer arbeitet mit der vierfachen Taktfrequenz. Für das gesamte Design detektiert der Design Compiler einen Flächenbedarf von 5,5 mm^2 und eine maximale Taktfrequenz von fast 100 MHz für die 80 MHz Domäne. Die statische Timing-Analyse wurde mit Primetime für einzelne kritische Pfade überprüft.

Das Layout wurde mit Silicon Ensemble durchgeführt. Aufgrund der vielen Eingangs- und Ausgangs-Pads wird die Größe des Designs durch den Umfang des Pad Rings bestimmt, so dass sich das Plazieren und Verdrahten der Standardzellen als umproblematisch erweist. Einzig das Erzeugen der Taktbäume (engl. *Clock Tree*) bereitet Probleme, da die verwendeten Programme zum Teil die Constraints falsch interpretieren und lange Inverterketten produzieren. Insgesamt verfügt der Prototyp über vier Taktbäume, wobei zwei voneinander abhängig sind (Übergang 20 MHz -> 80 MHz). Abbildung 59 zeigt ein Bild des Prototypen:

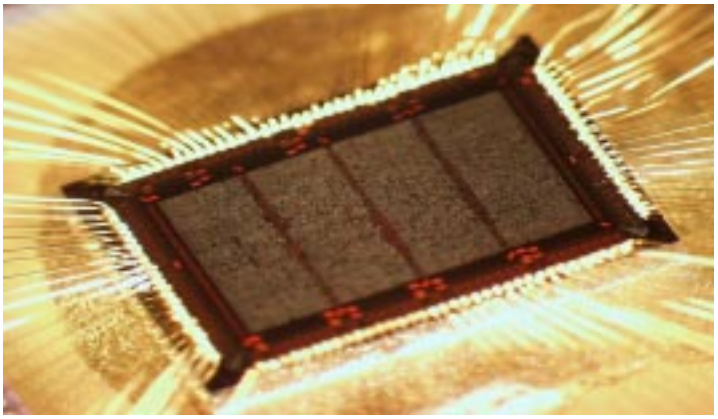


Abb. 59: FaRo I

A.1.7 Test des Prototypen

Für den Test [54] des Prototypen FaRo 1 wurde ein Testboard entwickelt [54], das als sog. Messeningkarte auf ein FPGA-Board aufgesteckt werden konnte. Darauf befindet sich unter anderem der Prototyp und ein programmierbarer Taktbaustein (Roboclock), der die Taktsignale vervielfachen kann. Als Testvektor Generator (engl. *Pattern Generator*) wird ein Lucent FPGA

des Typs Orca 3TP12 verwendet, welches das Verhalten der ADC's emuliert. Ferner steuert dieses FPGA den gesamten Testablauf. Der Test verläuft nach folgenden Schema:

- Konfiguration des FPGA
- Konfiguration des Prototypen
- Trigger mit anschließender Verarbeitung der Daten
- Auslese und Vergleich der Daten

Die FPGA Karte mit der aufgesteckten Testboard wird über einen PCI Bus mit einem PC verbunden. Über den PC werden alle Stufen des Tests gesteuert. Erst werden die Testvektoren, die mit dem C-Programm erzeugt wurden, in die internen Speicherelemente des FPGA geschrieben. Anschließend wird der Prototyp konfiguriert. Dann wird der Test gestartet und die Testvektoren werden an den Prototypen übergeben. Abschließend werden alle Werte der internen Speicher ausgelesen und mit den Erwartungswerten im PC verglichen.

Alle Aktionen können einzeln oder hintereinander mittels Shellskripte ausgeführt werden. Der Prototyp wurde mit 12000000 Testvektoren ohne Fehler gestestet. Abbildung 60 zeigt das FPGA-Testboard und das Board mit FaRo I.

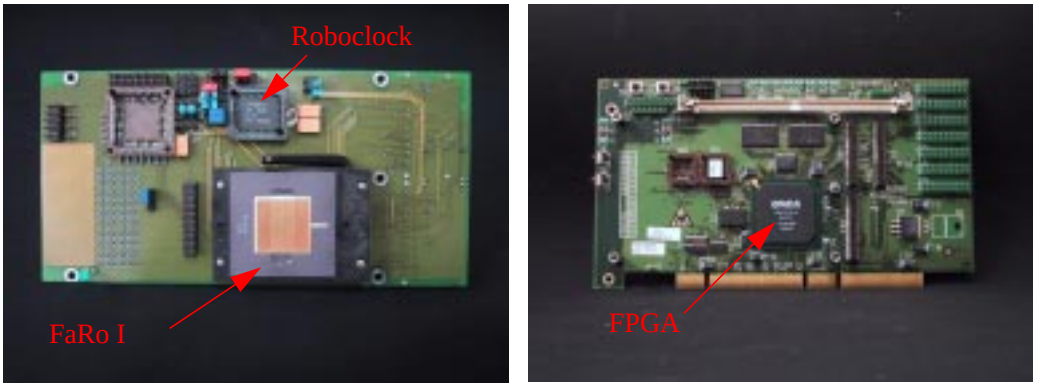


Abb. 60: Testboard von FaRo I und FPGA Testboard

Ferner wurden die Funktionalität des Prototypen sowie die elektrischen Eigenschaften auf einem Multi Chip Module (MCM) getestet. Insbesondere das Zusammenspiel mit einem Prototypen des Vorverstärkers wurde getestet. Auf dem Chip befindet sich der Prototyp FaRo I, ein Prototyp des Vorverstärkers (Pasa I) sowie vier kommerzielle ADC's. Insgesamt wurde dadurch ein Prototyp der Auslesekette bis zum MIMD Prozessor mit Prototypen aufgebaut, die der endgültigen Realisierung sehr Nahe kommt. Es konnte das Zusammenspiel einwandfrei nachgewiesen werden. Das digitale Rauschen, das vom FaRo- bzw. Roboclock-Chip kommt, beeinflusst die ADC's nicht. Es war kein Unterschied bei 8 Bit Quantisierung erkennbar. Abbildung 61 zeigt das MCM mit den Prototypen:

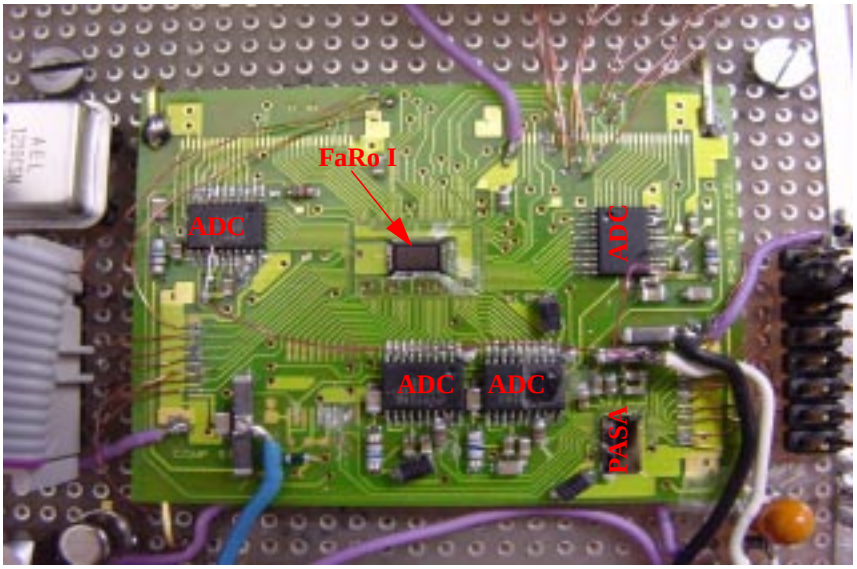


Abb. 61: MCM mit Prototypen

A.2 CSRAM

Das CSRAM ist ein Prototyp eines Datenspeichers mit vier Lese- und einem Schreib-Port. Er diente zur Evaluierung des Prinzips eines Quad Port Memories, das im aktuellen Prototypen Trap 1 als Daten- und Instruktionsspeicher zum Einsatz kommt. Die vier Ports können unabhängig voneinander betrieben werden. Das Speichermodul hat eine Größe von 16x2 Bit, die aus Flip-Flops des Typs DFA2 der AMS0.35 µm Standardzellentechnologie aufgebaut sind. Alle Eingangswerte werden in einer Registerbank zwischengespeichert, so dass die gelesenen Werte einen Takt später am Ausgang des Speichers sichtbar sind. Beim Schreiben werden die Adresse, das Datum und das WE Signal gespeichert. Das Takt- sowie das Reset-Signal werden nicht zwischengespeichert. Das Schreiben auf eine Speicherstelle wird erreicht, indem das Datum an alle Speicherstellen gelegt wird und das WE Signal mit Hilfe eines Adressmultiplexers an die entsprechende Speicherstelle durchgeschaltet wird. Bezogen auf den Zeitpunkt des Eintreffens der Daten, wird das Datum zwei Takte verzögert in den Speicher übernommen. Beim Lesen wird, analog zum Schreiben, die Adresse in ein internes Register übernommen, anschießend ab der nächsten steigenden Taktflanke dekodiert, um danach das ausgewählte Datum auf den Ausgang zu schalten. Abbildung 62 zeigt den Aufbau des Speichermoduls und das Layout.

Beim Test des Speichers sollte neben der einwandfreien Funktionalität vor allem geprüft werden, ob es Abhängigkeiten zwischen den Ports beim gleichzeitigen Zugriff auf verschiedene oder gleiche Speicherstellen gibt. Innerhalb des Tests wurden folgende Szenarien durchgespielt:

1. Sequentieller Zugriff über einen Port auf den gesamten Speicher
2. Paralleler Zugriff über zwei Ports auf verschiedene und gleiche Speicherstellen des Speichers
3. Paralleler Zugriff über alle Ports auf verschiedene und gleiche Speicherstellen des Speichers

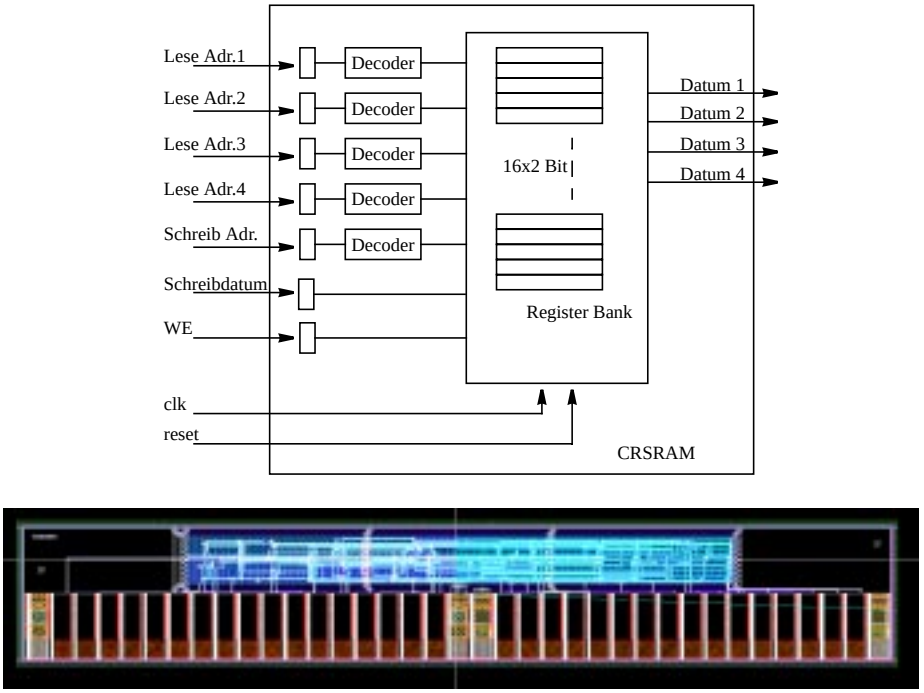


Abb. 62: Blockschaltbild und Layout des CRSRAM Moduls

Für die Messung wird ein Logikanalysator und ein Oszillograph sowie ein Taktgenerator verwendet. Der Messaufbau ist in Abbildung 64 dargestellt. Für die Messung der Zugriffszeiten wird eine Zeile mit “11” beschrieben, die anderen Zeilen mit Null. Die Zugriffszeit ist die Zeit von der steigenden Taktflanke bis zum Eintreffen der Daten am Logikanalysator bzw. Oszillograph.

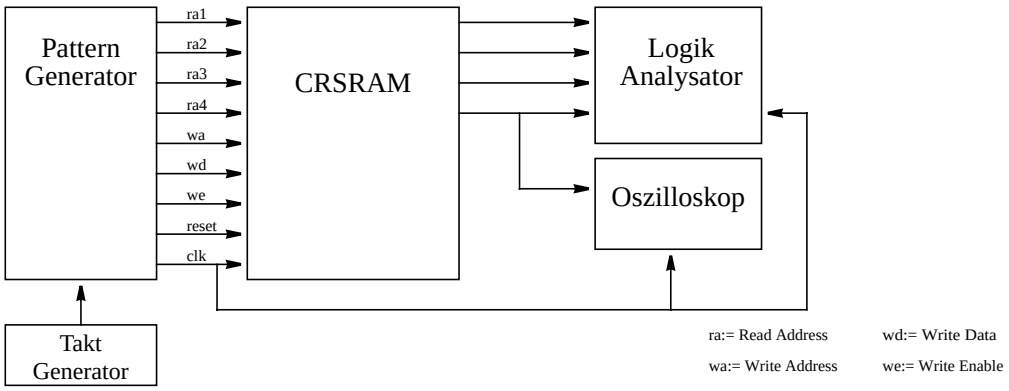


Abb. 63: Testaufbau des CRSRAM

Die gemessenen Zugriffszeiten lagen in allen Fällen zwischen 3 und 5 ns, was mit der Simulation gut übereinstimmt. Die beiden Bits einer Zeile lagen im Rahmen der Messgenauigkeit jeweils nach der selben Zeit am Ausgang vor. Zwischen den einzelnen Ports gab es geringe Unterschiede, so lagen die Daten an Port 1 nach 4-4,5ns vor, während an Port 4 teilweise nur 3,3ns vergangen

waren. Eine wesentliche Veränderung der Zugriffszeiten bei gleichzeitigem Lesen von mehr als einem Port konnte nicht festgestellt werden.

Die Ergebnisse für die drei Testszenarien sind in Abbildung 65 [15] dargestellt.

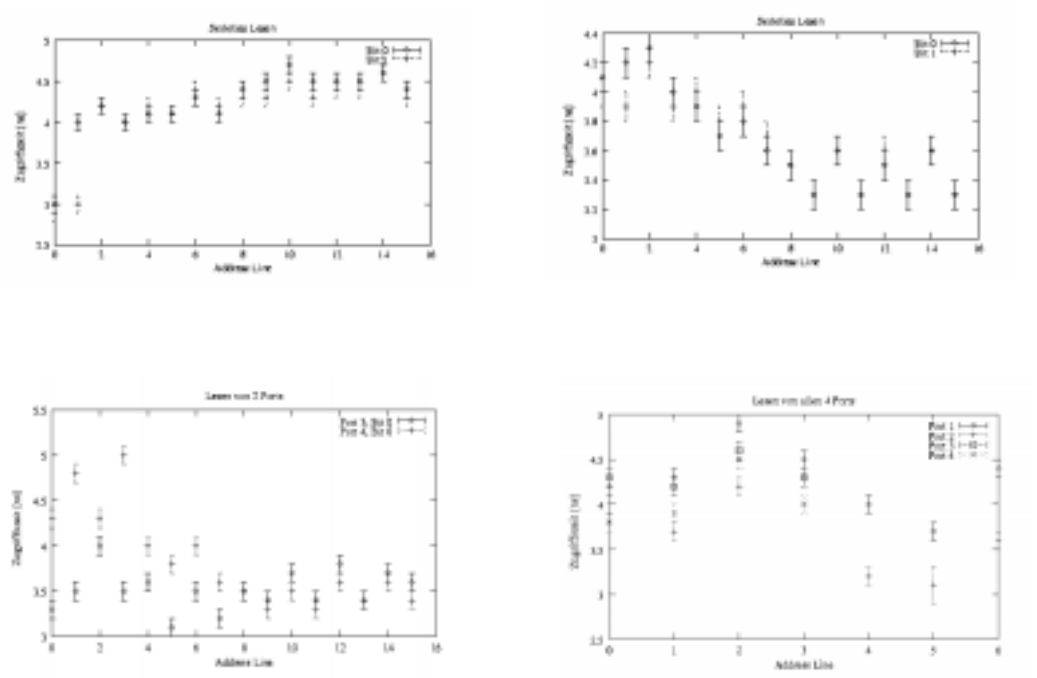


Abb. 64: Zugriffszeiten für den Zugriff auf Port 1 und 4 und zwei und vier Ports

B *Literaturverzeichnis*

[1] ALICE TRD Technical Proposal, CERN/LHCC 95-71

[2] ALICE TRD Technical Design Report, CERN/LHCC 2001-021

[3] MC68020 32-Bit Microprocessor User's Manual, Prentice-Hall, 1985

[4] H. Bähring: Mikrorechner-Systeme, Springer, 1991

[5] Bergmann, Schäfer: Teilchen, de Gruyter, 1992

[6] P.R. Bevington: Data Reduction and Error Analysis for the Physical Sciences, McGraw-Hill Book Company, 1969

[7] A. Bode, W. Händler: Rechnerarchitektur II, Strukturen, Springer, 1983

[8] I.N. Bronstein, K.A. Semendjajew, G. Musiol, H. Mühlig: Taschenbuch der Mathematik, Harry Deutsch Verlag, 1999

[9] U. Brüning: Vorlesung Rechnerarchitektur I, Universität Mannheim 2001

[10] H. Brunner: Introduction to Microprocessors, Reston Publishing, 1982

[11] A. Chandrakasan, W.J. Bowhill: Design of High-Preformance Microprocessor Circuits, IEEE Computer Society Press, 2000

[12] D.E. Culler, J.P. Singh: Parallel Computer Architecture, Morgan Kaufmann, 1999

[13] J. de Cuveland: Entwicklung einer ALU für einen 32 Bit RISC Prozessor, Miniforschung am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg, 2001

[14] W. Ehrhard: Parallelrechnerstrukturen, Teubner Verlag, 1990

[15] F. Ferner: Messung der Zugriffszeiten des CRSRAM Chips, Praktikum am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg, 2001

[16] M. Flynn: Some Computer Organisations and Their Effectiveness, IEEE Transactions on Computers, Sep. 1972

[17] C.V. Freiman: Statistical Analysis of Certain Binary Division Algorithms, Proc. IRE 49, S. 91-103

[18] R. Gareus: Slow Control - Serial Network and its implementation for the Transition Radiation Detector's slow control network, Diplomarbeit am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg 2002

[19] T. Gross, R. O'Hallaron: iWARP, MIT Press, 1997

[20] M. Gutfleisch: Digitales Frontend und Preprozessor im Trap1 Chip des Level 1 Triggers für das ALICE Experiment am LHC, Diplomarbeit am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg 2002

[21] J.P. Hayes: Computer Architecture and Organization, McGraw-Hill, 1997

[22] R.A. Iannucci: Multithreaded Computer Architecture, Kluwer Academic Publisher, 1994

[23] P. Kammerer: Skript zur Vorlesung Betriebssysteme I, TU-Darmstadt, 1996

[24] K. Kleinknecht: Detektoren für Teilchenstrahlung, Teubner Verlag, 1992

[25] V. Lindenstruth: Vorlesung Technische Informatik II, Heidelberg, 2000

[26] V. Lindenstruth: Vorlesung Chip Design, Heidelberg, 2001

[27] V. Lindenstruth: Vorlesung Parallele Computer Architektur, Heidelberg, 2000

[28] F. Lesser: A MIMD Based Multi Threaded Real-Time Processor for Pattern Recognition, Proceedings of the DSD 2001, Warschau, 2001

[29] F. Lesser: A MIMD Multi Threaded Processor, Proceedings 13th IEEE Hot Chips Conference, Palo Alto, USA, August 2001

[30] I. Koren: Computer Arithmetic Algorithmus, Brookside court publisher, 1998

[31] S. Martens: Programmierung eines Assemblers und Simulators für einen MIMD Prozessor, Praktikum am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg, 2001

[32] B. Parhami: Computer Arithmetic, Oxford University Press, 2000

[33] D.A. Patterson, J.L. Hennessy: Computer Organization & Design Morgan Kaufmann, 1998

[34] D.A. Patterson, J.L. Hennessy: Computer Architecture, Morgan Kaufmann, 1996

[35] K.P. Parker: The Bondary-Scan Handbook, Kluwer Academic Publisher, 1992

[36] C. Reichling: Entwicklung eines Quadport Memory für den Einsatz in einem Triggerprozessor am CERN, Diplomarbeit am Lehrstuhl für Technische Informatik, Universität Heidelberg, Heidelberg 2001

[37] H. Rohling: Einführung in die Informations- und Codierungstheorie, Teubner Verlag, 1995

[38] U. Rückert: Vorlesung Technische Informatik, Universität-GH Paderborn, 1996

[39] R. Schneider: 64k Networked Multi-Threaded Processors and their Real-Time Application in High-Energy-Physics, Proceedings of SCI 2002, Orlando, USA, July 2002

[40] W. Stallings: Computer Organization and Architecture, Prentice-Hall, 1996

[41] D. Sima, T. Fountain, P. Kacsuk: Advanced Computer Architectures, Addison-Wesley, 1997

[42] M.J.S. Smith: Application-Specific Integrated Circuit, Addison-Wesley, 1997

[43] O. Spaniol: Arithmetik in Rechenanlagen, Teubner Verlag, 1976

[44] A. S. Tanenbaum: Computer Networks, Prentice Hall, 1996

[45] T. Ungerer: Datenflußrechner, Teubner Verlag, 1993

[46] T. Ungerer: Parallelrechner und parallele Programmierung, Spektrum Verlag, 1997

[47] J.F. Wakerly: Digital Design, Prentice Hall, 2001

[48] K.Waldschmidt: Parallelrechner, Teubner Verlag 1995

[49] J.P. Wessels: The ALICE Transition Radiation Detector, Proceedings of TRD's of the 3rd Millenium, 2001

[50] R. Zimmermann: Computer Arithmetic, Principles, Architectures and VLSI Design, Lecture Notes, ETH Zürich, 1999

Datenquellen aus dem Internet

[51] Online Dokomentation des Design Compilers: http://www.synopsys.com/products/logic/design_compiler.html

[52] Online Dokomentation zu Prime Time: http://www.synopsys.com/products/analysis/analysis.html

[53] Online Dokomentation zu DesignWare: http://www.synopsys.com/products/designware/docs

[54] Online Dokumentation FaRo I: http://www.alicetrd.uni-hd.de/faro1

[55] Online Dokumentation zu Altera FPGA's: http://www.altera.com

[56] Online Dokumentation des GTK-Toolkits: http://www.gtk.org

Datenbücher

[57] AMS, Standard Cell Databook - Dokumentation zur Standardzellen-Bibliothek der Firma Austria Micro Systems International

[58] High Performance 0,18 µm Standard Cell Library (UMCL18U250), Dokumentation zur Standardzellen-Bibliothek der Firma Virtual Silicon, Rev. 2.2, Juni 2001

[59] Dokumentation zu ModelSim, Versin 5.3b

[60] Dokumentation zu Cadence Design Tools

Interne Mitteilungen

[61] C. Reichling: Charakteristische Größen von Vier-Port-Speichern in UMC 0,18 µm CMOS

[62] L. Musa: Design and Implementation of the ALTRO Chip

C *Abbildungsverzeichnis*

D *Tabellenverzeichnis*

E *Die CD's*

Die beigefügten CD's enthalten neben der Dokumentation im Framemaker- und pdf-Format die erstellten VHDL-Files, Skripte, Simulationsumgebungen und Ausgabedateien der verschiedenen Tools des Designflows. Ferner befinden sich die Daten der beiden Prototypen auf den CD's.

Für eine detaillierte Übersicht und tiefergreifende Erläuterungen sei auf die zum Teil vorhandenen README-Files hingewiesen, die den einzelnen Unterverzeichnissen beigefügt sind.

Danksagung

Ohne die Hilfe und Unterstützung von vielen netten Menschen wäre diese Arbeit nicht möglich gewesen.

Insbesondere möchte ich mich bei meinem Doktorvater Herrn Prof. Dr. Lindenstruth bedanken, der mir in den fast vier Jahren der Zusammenarbeit durch sein fundiertes Fachwissen, seiner motivierenden Art und durch sein Gespür für das Wesentliche ein besonderes Vorbild war. Auf seinen Ideen und Tipps fußt der Erfolg meiner Arbeit.

Ebenso möchte ich mich bei Dr. Venelin Angelov und Dr. Markus W. Schulz bedanken. Markus Schulz hat den Dividierer von FaRo I entwickelt und war maßgeblich an dem konzeptionellen Entwurf des MIMD Prozessors beteiligt. Venelin Angelov hat den Entwurf und die Realisierung aller Module mitgetragen. Sein Wissen und seine Erfahrung waren mir oft eine große Hilfe.

Rolf Schneider und Christian Reichling sind ebenfalls Doktoranden im TRD Projekt und haben mich während der gesamten Zeit fachlich und praktisch unterstützt. Christian Reichling hat die Vier-Port-Speicher des MIMD Prozessors entwickelt. Rolf Schneider war maßgeblich an der Entwicklung von FaRo I beteiligt und hat mich ebenso bei der Entwicklung des MIMD Prozessors unterstützt.

Außerdem möchte ich mich bei Robin Gareus und Marcus Gutfleisch bedanken. Robin Gareus hat die Konfigurationseinheit des MIMD Prozessors entwickelt und getestet sowie den Test des MIMD Prozessors auf RTL und Gatelevel-Niveau durchgeführt. Marcus Gutfleisch hat den Preprozessor entwickelt, getestet und realisiert. Beide Arbeiten wurden in Rahmen von Diplomarbeiten durchgeführt.

Jan de Cuveland war wesentlich an der Entwicklung des Prototypen FaRo I beteiligt. Er hat nahezu den gesamten Test des Prototypen auf allen Entwurfsebenen durchgeführt. Ferner hat er die ALU des MIMD Prozessors entwickelt, wobei die Entwicklung des Radix-4 Dividierers besonders zu erwähnen ist.

Bei Stefan Martens bedanke ich mich für die Entwicklung des Assemblers und Simulators, der auch nach verschiedenen Designänderungen diese beiden Module auf dem aktuellen Stand gehalten hat. Außerdem hat er den Test des MIMD Prozessors durch den Entwurf verschiedener Testprogramme unterstützt.

Ebenso danke ich Béatrice Bähr, die durch ihre Hilfsbereitschaft und ihr Organisationstalent zu jeder Zeit eine große Unterstützung war.

Des weiteren möchte ich mich bei Michael Keller, Cornelius Schumacher, Martin Feuerstack-Raible, Ralf Achenbach, Alexander Walsch, Frederick Ferner, Felix Rettig, der Crew der

Elektronikwerkstatt und allen anderen, die hier nicht namentlich genannt werden, recht herzlich bedanken.

Ein besonderer Dank gilt meiner Frau Cordula Lesser, die durch ihre endlose Geduld und durch ihr genaues Korrekturlesen wesentlich an dem Erfolg meiner Arbeit Teil hat. Meinen Eltern danke ich ganz herzlich für ihren Rat, Zuspruch und die finanzielle Unterstützung während der gesamten Zeit meiner Ausbildung.