Preemptive Multitasking auf FPGA-Prozessoren

Ein Betriebssystem für FPGA-Prozessoren

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

Vorgelegt von
Dipl.-Ing. (FH) Harald C. Simmler
aus Mannheim

Mannheim 2001

Dekan: Professor Dr. Guido Moerkotte, Universität Mannheim
Referent: Professor Dr. Reinhard Männer, Universität Mannheim
Korreferent: Professor Dr.-Ing. Norbert Wehn, Universität Kaiserslautern

Tag der mündlichen Prüfung: 06. Juli 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Seit dem Einsatz der Computer für die Verarbeitung von Daten besteht ein stetig steigender Bedarf an Rechenleistung in nahezu allen Einsatzbereichen. Diese Forderung nach immer mehr Rechenleistung wird jedoch nur zum Teil durch die sich ständig verbessernden von-Neumann Prozessoren erfüllt. Verschiedene Anwendungen aus unterschiedlichen Bereichen, wie z.B. der Bildverarbeitung, der Prozesssteuerung oder der Simulationen, erfordern die Verarbeitung der Daten in Echtzeit bzw. in einer vorgegebenen Zeit, die von den von-Neumann Prozessoren nicht garantiert oder erreicht werden kann. Um diese Anwendungen unter den gegebenen Randbedingungen verarbeiten zu können werden spezifische Lösungen realisiert. Zu diesem Lösungen zählen Spezialprozessoren(DSPs), anwendungsspezifische Prozessoren(ASICs), aber auch Parallelrechner und Netzwerk-Cluster.

Eine weitere Alternative zu den oben genannten Lösungen ist der Einsatz von programmierbaren Logikbausteinen. Diese Logikbausteine gehören zu der Gruppe der 'Field Programmable Logic Arrays' (FPGA) und weisen eine starke Ähnlichkeit zu ASIC Bausteinen auf. Aufgrund der gleichen konzeptionellen Basis erreichen die FPGA-Bausteine ähnliche Leistungssteigerungen wie vergleichbare ASICs, dennoch unterscheiden sich die FPGA-Bausteine von den ASICs durch ihre Eigenschaft der

Rekonfigurierbarkeit. Beide Bausteine können jede beliebige kombinatorische oder arithmetische Funktion realisieren. Während jedoch beim ASIC Baustein diese Funktionen einmal programmiert und nicht wieder verändert werden kann, speichert der FPGA-Baustein die Funktionen in einem flüchtigen Speicher. Durch eine Veränderung dieses Speicherinhalts können somit auch die Funktionen an die jeweilige Aufgabe dynamisch angepaßt werden.

Diese beiden Eigenschaften der FPGA-Bausteine – hohe Rechenleistung und hohe Flexibilität – werden zur Verarbeitung von unterschiedlichen rechenintensiven Anwendungen genutzt. Darauf basierende Systeme werden im allgemeinen als FPGA-Prozessoren bezeichnet und sind in verschiedenen Ausführungen verfügbar. Am häufigsten eingesetzt werden die FPGA-Prozessoren als PCI-Einsteckkarte, um in Form eines Koprozessores spezifische Anwendungen zu beschleunigen. Durch den Einsatz von anwendungsspezifischen Rechenwerken, die auf den FPGA-Bausteinen ausgeführt werden, sind Beschleunigungsfaktoren zwischen 10 und 1000 erreichbar. Anwendungsfelder in denen heutzutage FPGA-Prozessoren Einsatz finden sind: die Bildbearbeitung/Bildverarbeitung, die Kryptographie, die Mustererkennung, die Bioinformatik, der Prototypenbau, die Simulation komplexer physikalischer Vorgänge (Astronomie, Atomphysik), die Volumenvisualisierung, u. v. m.

Gegenüber den anderen, zum Teil sehr speziellen Lösungen, verfügt der FPGA-Prozessor über mehrere Vorteile. Zum einen Ermöglicht der Aufbau als PCI-Einsteckkarte eine einfache Integration des Koprozessors in bestehende Rechnersysteme und in die Anwendungen. Zudem wird durch die freie Programmierbarkeit der Bausteine eine parallele Verarbeitung auf Ebene der Daten (SIMD), aber auch auf Ebene der Operationen (MIMD) erreicht. Die weitere Eigenschaft der Reprogrammierbarkeit ermöglicht es zusätzlich den FPGA-Prozessor universell für die verschiedensten Anwendungen zu nutzen.

Abhängig von der jeweiligen Anwendung werden einzelne Bereiche der Datenverarbeitung zur Beschleunigung auf den FPGA-Prozessor ausgelagert. In der Regel handelt es sich dabei um sehr rechenintensive Programmteile, die in der Anzahl der zu verarbeitenden Daten, aber auch in der Anzahl der auszuführenden Operationen stark variieren kann. Ge-

messen an der Datenmenge und der Anzahl der auszuführenden Operationen werden die ausgelagerten Programmteile in drei Abstraktionsebenen unterteilt, die nachfolgend aufgelistet sind:

Maschinenbefehlsebene: Bei der Maschinenbefehlsebene werden einzelne Werte übergeben, die mit wenigen Operationen verarbeitet werden. Ein Beispiel ist die Berechnung des größten gemeinsamen Teilers auf dem FPGA-Prozessor. Ausgelagerte Programmteile dieser Art ergänzen dynamisch den Befehlssatz des Prozessors durch anwenderspezifische Befehle.

Funktionsebene: Programmteile, die zu der Funktionsebene zählen, berechnen vollständige rechenintensive Funktionen eines Programms auf dem FPGA-Prozessor. Diese Programmteile sind gekennzeichnet durch die Ausführung von mehreren Operationen, unabhängig von der Datenmenge. Dazu gehören z.B. NP-vollständige Berechnungen oder die Filterung von Bilddaten.

Programmebene: Kennzeichnend für die Programmebene ist die Ausführung des gesamten Programmablaufs auf dem FPGA-Prozessor, bei dem alle Daten verarbeitet werden. Ein Beispiel dafür ist der Gensequenzvergleich gegenüber einer Genomdatenbank.

Für die Realsisierung der Anwendungen kommen alle drei oben genannten Ebenen zum Einsatz. Aus Gründen der Effektivität und des zu erreichenden Beschleunigungsfaktors basieren heutzutage die Anwendungen meistens auf der Funktions- und der Programmebene.

1.1 Motivation

Die FPGA-Prozessoren verfügen über die Eigenschaft der dynamischen Rekonfigurierbarkeit, die es ermöglicht unterschiedliche Anwendungen auf nur einer Einsteckkarte ausführen zu können. Derzeit wird diese Eigenschaft in den meisten Fällen nur dazu genutzt, um nachträgliche Änderungen der Anwendung oder zusätzliche Verbesserungen beim Programmablauf zu realisieren.

In der Regel werden die FPGA-Prozessoren nur von einer oder von wenigen Anwendungen zur Beschleunigung eingesetzt, da jede Anwendung den FPGA-Prozessor vollständig und ausschließlich verwendet. Diese exklusive Nutzung entsteht durch die direkte Verwendung des Geräte-Treibers der den Zugriff auf den FPGA-Prozessor bereitstellt.

Aufgrund der steigenden Anzahl an allgemeinen Anwendungen, wie z.B. Bildbearbeitung, Videokompression oder Kryptographie, die durch den Einsatz eines FPGA-Prozessors beschleunigt ausgeführt werden, ist in Zukunft die exklusive Verwendung des FPGA-Prozessors durch eine Anwendung inakzeptabel, da viele Anwendungen auf den FPGA-Prozessor zugreifen. Zudem kann durch die Ausführung einer sehr rechenaufwenigen Anwendung der FPGA-Prozessor für Stunden oder Tage blockiert werden.

Durch die Einführung eines Betriebssystems zur Steuerung des FPGA-Prozessor wird eine allgemeine Nutzung aus verschiedenen Anwendungen heraus ermöglicht. Dieses Betriebssystem besteht aus verschiedenen Programmen die den exklusiven Zugriff auf die konkurrierenden Anwendungen verteilt und somit die Verarbeitung der unterschiedlichen Anwendungen auf dem FPGA-Prozessor ermöglicht.

1.2 Stand der Technik

In der Vergangenheit haben sich mehrere Forschungsgruppen weltweit mit der Fragesstellung eines Betriebssystems für FPGA-Prozessoren und dem Bau von FPGA-Bausteinen zur Unterstützung eines Betriebssystems auseinandergesetzt. Exemplarisch werden hier die für diese Arbeit wichtigsten Konzepte kurz beschrieben und deren Möglichkeiten aufgezeigt.

Rekonfigurierbare Logik für Computersysteme

An der Universität Tübingen (G. Haug und W. Rosenstiel) [HR98a, HR98b] wurde eine Betriebssystemerweiterung für das vorhandene LINUX Betriebssystem realisiert, das jeder Anwendung die Nutzung des FPGA-Prozessors ermöglicht.

Die Erweiterung basiert auf einem eigenen FPGA-Prozessor der als PCI-Einsteckkarte aufgebaut ist. Durch Modifikationen des Scheduling-Algorithmus und Erweiterungen der Systemfunktionen wird die Rekonfigurierbarkeit sehr intensiv genutzt, denn mit jedem Prozesswechsel auf dem Host-Prozessor wird gleichzeitig auch der FPGA-Baustein rekonfiguriert. Diese Vorgehensweise ist notwendig, denn die Anwendungen, die auf dem FPGA-Prozessor ausgeführt werden, zählen zu der Maschinenbefehlsebene und bestimmten mit dem Euklidschen Algorithmus den größten gemeinsamen Teiler aus zwei Werten.

Dieser Betriebssystemansatz zeigt, daß unter Verwendung eines speziellen FPGA-Bausteins (Xilinx XC6200) die Ausführung von verschiedenen Anwendungen auf einem FPGA-Prozessor möglich ist. Gesteuert wird die Verarbeitung durch das existierende Betriebssystem, da die Anwendungen der Maschinenbefehlsebene zur gleichen Zeit auf dem Host- und dem FPGA-Prozessor verarbeitet werden müssen. Der Einsatz ist jedoch auf Anwendungen der Maschinenbefehlsebene begrenzt, da das Konzept nur die Rekonstruktion der Register innerhalb des FPGA-Bausteins unterstützt und eine Übertragung von großen Datenmengen nicht effektiv durchgeführt werden kann.

Zusammenfassend kann man sagen, daß die Arbeit das Prinzip eines Multitasking auf FPGA-Bausteinen zwar bestätigt hat, aber dennoch für die Umsetzung von realen Anwendungen nur eine sehr bedingte Aussagekraft besitzt.

Dynamisch Rekonfigurierbare FPGAs

An der University of Edinburgh (G. Brebner) [Bre97b, Bre98a] wurde auf Basis des gleichen FPGA-Bausteins XC6200 ein weiterer Ansatz entwickelt. Das dort entstandene Paradigma unterscheidet sich vom vorhergehenden Ansatz durch die Aufteilung der zur Verfügung stehenden FPGA-Ressourcen. Die Maschinenbefehle werden auf sogenannte 'Swappable Logic Units' (SLU) abgebildet, die vergleichbar mit Speicher-Pages dynamisch ausgewechselt werden können. Die Verarbeitung der einzelnen Anwendungen wird parallel und unabhängig von der Anwendung auf dem Host-Prozessor ausgeführt.

Auch dort haben die Arbeiten gezeigt, daß logische Maschinenbefehle parallel auf einem FPGA-Baustein verarbeitet werden können. In Vergleich zu dem ersten Ansatz wird zusätzlich gezeigt, daß die parallele Verarbeitung der Anwendungen durch eine betriebssystemgesteuerte Aufteilung der vorhandenen FPGA-Ressourcen durchgeführt werden kann. Diese partielle Nutzung der vorhandenen FPGA-Ressourcen erfordert jedoch einen Plazierungsalgorithmus. Aufgrund der ausgeführten logischen Funktionen innerhalb der SLUs wird der Austausch der einzelnen SLUs durch ein einfaches überschreiben realisiert.

Wie schon zuvor konnte auch hier das Prinzip des Multitaskings zwar bestätigt werden, aber dennoch besitzt es nur eine bedingte Aussagekraft, da es nur logische Maschinenbefehle verarbeiten kann. Zudem ist auch bei diesem Ansatz die Ausführung von Anwendungen der Funktions- oder Programmebene aufgrund der fehlenden Datenübertragung nicht möglich.

FPGA Unterstützung für konkurrierende Prozesse

Der FPGA-Betriebsmanager der Wright State University (J. Jean) [JTY+98] basierte auf einer PCI-Einsteckkarte die über acht gleiche FPGA-Module verfügt. Die Aufgabe des Betriebssystem ist die Verteilung der FPGA-Module an die konkurrierenden Anwendungen mit dem Ziel, die Gesamtlaufzeit aller Anwendungen durch eine parallele Nutzung der vorhandenen FPGA-Module zu optimieren.

Dieser Ansatz zeigt, daß Anwendungen über die Maschinenbefehlsebene hinaus durch ein Betriebssystem gesteuert werden können. Berechnet wird ein NP-vollständiges Problem das keinen großen Datentransfer benötigt. Der Ansatz geht über ein reine Lastenverteilung hinaus und ermöglich es den FPGA-Prozessor aus verschiedenen unabhängigen Anwendungen heraus zu nutzen.

Geeignet ist dieses Konzept sowohl für Anwendungen der Maschinenbefehlsebene als auch der Funktionsebene, die nur im geringen Umfang auf einen Datentransfer angewiesen sind. Diese Einschränkung begründet sich durch die limitierte Datentransferrate die auf dem PCI-Bus zur Verfügung steht. Eine Aufteilung dieser verfügbaren Datenrate kann

zu unterbrochenen Verarbeitungsabläufen führen, die die Effektivität der Anwendungen senkt. Daher ist auch dieser Betriebssystemansatz nur für eine kleine Gruppe von Anwendungen einsetzbar.

Multitasking unterstützende FPGA-Bausteine

Neben der Forschung auf dem Gebiet der Betriebssysteme für FPGA-Prozessoren wurden auch neue FPGA-Architekturen entwickelt, die mehrere Konfigurationen speichern können. Diese sogenannten 'Context Switching' FPGA-Bausteine [TCJW97, SV98] sind in der Lage die gesamte Konfiguration innerhalb von nur einem Takt (5–30ns) zu wechseln. Diese Entwickungen wurden dabei aus folgenden Gründen vorangetrieben:

Redundanz: Der Aufbau redundater Systeme (z.B. Weltraumsonden, Satelliten).

Logik-Erweiterung: Die limitierte Anzahl an Logikzellen wird durch ein zeitliches Multiplexen vervielfacht.

Multitasking: Zeitlich unterteilte Verarbeitung mehrerer Anwendungen auf FPGA-Bausteinen.

Trotz der guten Eigenschaften hinsichtlich des Multitasking Betriebs haben sich diese Bausteine nicht durchsetzen können. Die Gründe dafür sind die schnell steigenden FPGA-Ressourcen, die geringe Ausnutzung der Siliziumfläche und die mangelnde Unterstützung durch die Entwicklungswerkzeuge. Durch bessere Fertigungtechnologien verfügen moderen FPGA-Bausteine über weit mehr Logikressourcen als alle zusammengenommenen Konfigurationen eines 'Context-Switching' FPGA-Bausteins und machen die Logikerweiterung somit überflüssig. Die Integration mehrerer Konfigurationsspeicher reduziert bei einer gleichbleibenden Siliziumfläche zudem die Anzahl der Logikressourcen.[1] Dies wiederum führt zu großen und damit sehr teuren Bausteinen mit einer Vergleichsweise schlechten Anzahl an Logikressourcen. Der letzte Punkt ist die mangelnde Unterstützung der bereitgestellten Funktionalität durch die Entwicklungswerkzeuge.

[1] Diese Verhältniss ist eines der am besten gehütesten Geheimnisse der FPGA-Hersteller.

Zusammenfassung

Die oben beschriebenen Ansätze haben gezeigt, daß die Ausführung mehrerer konkurrierender Anwendungen auf einem FPGA-Prozessor durch ein Betriebssystem gesteuert werden kann.

Dennoch können diese Betriebssysteme nur Anwendungen der Maschinenbefehlsebene oder der Funktionsebene ausführen, die nur sehr wenig Daten verarbeiten. Wie eine Analyse vorhandener FPGA-Prozessor Anwendungen zeigt, verarbeiten die Anwendungen auf den FPGA-Prozessoren zum Teil sehr große Datenmengen, die zwischen dem Speicher und dem FPGA-Prozessor übertragen werden müssen. Diese Anwendungen gehören ausschließlich zu der Klasse der Funktions- und der Programmebene.

Die Analyse hat zudem ergeben, daß die Ausführungszeiten der Anwendungen zum Teil Stunden oder Tage betragen. Mit Ausnahme des ersten Betriebssystems kommt die sogenannte 'Overlay' Technik zum Einsatz, die die ausgeführte Anwendung im FPGA-Baustein überschreibt. Die Anwendung dieser Technik ermöglicht es dem Betriebssystem *nicht* eine laufende Anwendung vorzeitig zu unterbrechen und diese zu einem späteren Zeitpunkt an dieser Stelle weiter fortzusetzen. Somit können durch die langen Ausführungszeiten blockierende Situationen enstehen, die für eine gemeinsame Nutzung inakzeptabel sind.

Der letzen Punkt bezieht sich auf die zur Verwendung kommenden FPGA-Bausteine. Zwei der vorgestellten Betriebssysteme verwenden einen speziellen FPGA-Baustein und sind aus diesem Grund nicht allgemein übertragbar auf andere u.a. modernere FPGA-Prozessoren. Ebenfalls nur bedingt einsetzbar sind die 'Context Switching' FPGA-Bausteine, da sie nicht verfügbar sind und nur unzureichend durch die Entwicklungswerkzeuge unterstützt werden.

1.3 Zielsetzung

Das Ziel, das durch diese Arbeit realisiert werden soll, ergibt sich direkt aus den umfangreichen Einschränkungen der zuvor beschriebenen Betriebssysteme. Es soll ein allgemeingültiges Betriebssystems für FPGA-Prozessoren aufgebaut werden, das keine Einschränkungen bezüglich der

Art der Anwendungen oder der verwendeten FPGA-Prozessoren besitzt. Das entstehende Multitasking Betriebssystem soll in der Lage sein, mehrere unabhängige Anwendungen parallel zu verarbeiten, indem die verfügbare Rechenzeit des FPGA-Prozessors unter den konkurrierenden Anwendungen aufgeteilt wird. Bei der Realisierung wird auf die folgenden Punkte im Speziellen geachtet:

Task-Switching: Das Betriebssystem ist in der Lage ganze Anwendungen in der Ausführung auf dem FPGA-Baustein anzuhalten und zu einem späteren Zeitpunkt fortzusetzen.

FPGA-Prozessor Unterstützung: Das Betriebssystem unterstützt alle FPGA-Prozessoren, die bestimmte Anforderungen erfüllen.

Anwendungsunterstützung: Durch das Betriebssystem werden alle Arten von Anwendungen verarbeitet. Um dies zu gewährleisten werden alle an der Ausführung beteiligten Komponenten (FPGA-Baustein und RAMs) gesichert.

Entwicklungsumgebung: Der Entwicklungsprozess von Anwendungen, die den FPGA-Prozessor nutzen, wird durch eine Entwicklungsumgebung automatisiert und somit beschleunigt.

Die sich daraus ergebenden Teilziele sind wie folgt definiert:

1. Untersuchung der Machbarkeit des Task-Switchs auf FPGA-Bausteinen.

2. Aufstellen der Anforderungen die an die FPGA-Bausteine und die FPGA-Prozessoren gestellt werden.

3. Konzeptionierung und Realisierung des Betriebsystems das die oben genannten Punkte unterstützt.

4. Aufbau eines neuen FPGA-Prozessors, der die Verarbeitung des Betriebssystems im besonderen Maße unterstützt.

Diese Arbeit beschreibt das erste allgemeingültige Betriebssystem für FPGA-Prozessoren das es mehreren konkurrierenden Anwendungen ermöglicht, den FPGA-Prozessor gleichzeitig zu nutzen.

Der Inhalt dieser Dissertation gliedert sich folgendermaßen: Kapitel 2 beschreibt die Grundlagen des Multitasking und den Aufbau und die Anwendungsentwicklung von FPGA-Prozessoren. Kapitel 3 erläutert das Multitaskingkonzept für FPGA-Bausteine und deren Umsetzung am Beispiel eines modernen FPGA-Bausteins. Kapitel 4 evaluiert die möglichen Konzeptionen eines Betriebssystem und deren Einbindung in bestehende Betriebssysteme. Die Umsetzung der Konzeption wird in Kapitel 5 beschrieben. In Kapitel 6 wird die Konzeption eines speziellen Multitasking FPGA-Prozessors erläutert und der Aufbau dieses FPGA-Prozessors aufgezeigt.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Gründe und Vorteile, die für den Einsatz des Multitaskings auf FPGA-Koprozessoren sprechen, kurz beschrieben. Die grundlegenden Techniken, die in ähnlicher Weise auch bei dem bekannten Multitasking-Betriebssystem Verwendung finden, werden anhand der Umsetzung in modernen Prozessorarchitekturen dargestellt.

Nach Erläuterung der Grundprinzipien des Multitasking wird der allgemeine Aufbau und die Funktionsweise der FPGA-Bausteine näher erläutert. Gegliedert ist dies in eine detaillierte Beschreibung der bausteinübergreifenden gemeinsamen Eigenschaften und das Vorgehen bei der Konfiguration der FPGA-Bausteine. Zusätzlich werden weitere Besonderheiten der einzelnen FPGA-Bausteine genannt.

Diese Bausteine bilden die Basis für die nachfolgend beschriebenen FPGA-Prozessoren. Die Klassifizierung und die Einteilung in zwei Gruppen von FPGA-Prozessoren wird erläutert.

Auf die Beschreibung der FPGA-Prozessoren folgt eine umfangreiche Erläuterung zur Programmierung der FPGA-Prozessoren. Diese bildet die Grundlage zur Durchführung des Multitaskingbetriebs auf FPGA-Prozessoren. In weiteren Abschnitten werden die Möglichkeiten der Eingabe und die Umsetzung des gesamten Entwicklungsablaufs vom der Netzliste zu einem konfigurierbaren Design anhand eines Beispielpro-

gramms näher erläutert. Abschließend wird kurz auf die Verifikation der Schaltungen eingegangen.

2.1 Multitasking

Bei der Klassifizierung von Betriebsarten wird generell in die Singletasking- oder Multitasking-Betriebsart unterschieden. Im Singletasking-Betrieb kann zu einem Zeitpunkt nur ein Programm ausgeführt werden. Ein Beispiel hierfür ist MS-DOS. Bei einem Multitasking-Betrieb, wie er in vielen modernen Betriebssystemen, z.B. Windows NT oder Unix, Anwendung findet, können mehrere unabhängige Programme parallel gestartet und verarbeitet werden.

Der Vorteil des Multitasking-Betriebs gegenüber dem Singletasking-Betrieb ist gegeben durch eine bessere Prozessorauslastung, einer höheren Systemausfallsicherheit, einer Verkürzung der Antwortzeiten und die Vermeidung von blockierenden Zuständen. Die bessere Prozessorauslastung entsteht durch den ermöglichten Wechsel zu einem ausführungsbereiten Prozess, sobald der laufende Prozess z.B durch einen I/O-Zugriff auf ein angefordertes Datenwort warten muß und so der Programmablauf blockiert wird. Die Systemausfallsicherheit entsteht durch die Unabhängigkeit der Prozesse innerhalb eines Multitasking Betriebssystems und deren zeitlich verschachtelten Ausführung. Dadurch führt ein Fehlverhalten eines Prozesses nicht zum Absturz des gesamten Systems. Die Verkürzung der Antwortzeiten ist vor allem für Dialogsysteme wichtig. Sie werden aufgrund er häufig entstehenden I/O-Zugriffe im Besonderen durch ein Multitasking-Betriebssystem begünstigt. Der letzte und wichtigste Punkt ist die Vermeidung von blockierenden Situationen. Ein sehr rechenintensiver Prozess verursacht im Singletasking-Betrieb eine große Verzögerung aller nachfolgender Prozesse. Solche Situationen gilt es durch den Einsatz eines Multitasking-Betriebs zu vermeiden.

Neben der Klassifizierung in den Single- und Multitasking-Betrieb werden die Betriebssysteme weiterhin unterteilt in verdrängende (preemptive) und nichtverdrängende (nicht preemptive) Systeme. In nicht preemptiven Systemen kommen Prozesswechsel nur bei einer freiwilligen Abgabe des Prozessors durch den laufenden Prozess zustande. Ein

Vertreter der nicht preemptiven Systeme ist Windows 3.x. Preemptive Betriebssysteme, wie z.B. Windows NT oder Linux, entziehen dem laufenden Prozess auch ohne dessen Zustimmung den Prozessor. Gesteuert wird die Prozessorverteilung durch die sogenannten Scheduling-Strategie.

2.1.1 Hardware Unterstützung

Die Unterbrechung der Prozessausführung auf dem Prozessor wird durch mehrere Faktoren ermöglicht. Ein Faktor ist die sequentielle Ausführung der Programme auf dem Prozessor. Jede Programmoperation die auf dem Prozessor ausgeführt wird stellt eine atomare, nicht unterbrechbare, Operation dar. Abstrakte Programmfunktionen werden mit Hilfe eines Compilers auf die begrenzte Anzahl der Maschinenbefehle abgebildet, wodurch sehr viele mögliche Unterbrechnungspunkte innerhalb eines Programms entstehen.

Darüberhinaus ist der Zustand eines Prozesses durch den Befehlszähler, das Flag Register, die Daten-, Befehls- und Stackzeiger, die Segmentregister und die allgemeinen Register vollständig definiert [Bra94]. Die Sicherung dieser Daten in einem speziellen Speicher, der *Task-Status-Segment (TSS)* genannt wird, ermöglicht es, den Prozess bei einer erneuten Prozessorzuteilung an der unterbrochenen Stelle fortzusetzen.

Der dritte Faktor ist die hardwaremäßige Unterstützung durch die Speicherverwaltung in Multitasking Betriebssystemen. Die Trennung von virtuellem und physikalischem Speicher stellt jedem Prozess einen eigenen geschützen Adressraum zur Verfügung, der mit den gesicherten Daten-, Befehls- und Stackzeigern adressiert wird.

Der Prozesswechsel auf den Intel-Prozessoren wird durch den speziellen **JMP TSS** Befehl eingeleitet. TSS bezeichnet das dem jeweiligen Prozess zugeteilte Status-Segment, in dem die zu sichernden Registerwerte gespeichert werden. Die Ausführung des Befehls, der in Abbildung 2.1 dargestellt ist, erfolgt in drei Stufen:

1. Alle Prozessorregister werden in dem TSS des ausscheidenden Prozesses gesichert.

2. Das Task-Register wird mit den TSS des eintretenden Prozesses geladen.

3. Alle Prozessorregister des neuen Prozesses werden aus dem TSS geladen.

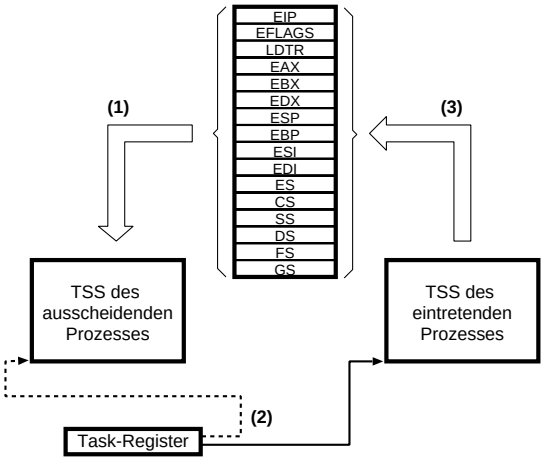


Abbildung 2.1: Ausführungsreihenfolge des 'JMP TSS' Maschinenbefehls der am Beispiel des Intel 80486 Prozessors. *(Entnommen aus [Eic96])*

2.1.2 Betriebssystemunterstützung

Der oben skizzierte Mechanismus zum Wechseln der Prozesse auf dem Intel-Prozessor stellt die notwendige Hardwarevoraussetzung zur Unterstützung für das Betriebssystem dar. Alle weiterführende Maßnahmen zur Umsetzung der Multitaskingstrategie werden innerhalb des Betriebssystems umgesetzt.

Ein zentraler Punkt des Betriebssystems ist die Verwaltung der einzelnen Prozesse. Für diese Verwaltung sind sowohl betriebssystemabhängige Informationen als auch hardwareabhängige Informationen not-

wendig. Zu den Betriebssysteminformationen gehören z.B. die Prozessidentifikationsnummer, der Prozesszustand und die Prozesspriorität. Angaben zu dem virtuellen Adressraum und den geöffneten Dateien sind weiter Bestandteile der Betriebssysteminformation. Die hardwareabhängigen Bestandteile sind u.a. der Prozessorzustand und der Zustand von geöffneten Geräten. Diese gesamten Informationen werden für jeden Prozess in einer separaten Prozesstabelle innerhalb des Betriebssystems zusammengefaßt und durch das Betriebssystem ausgewertet.

Die Ausführung der Prozesse wird von dem Betriebssystem gesteuert und kontrolliert. Die sogenannte Scheduling-Strategie[1] bestimmt meist dynamisch welcher Prozess auf dem Prozessor verarbeitet wird. Die Kontrolle der einzelnen Prozesse erfolgt während der Ausführung des Betriebssystems, die entweder durch den Aufruf einer Systemfunktion oder durch einen Zeitgeber aktiviert wird. Durch diesen Mechanismus wird sichergestellt, daß wartende Prozesse sofort gewechselt werden und das sehr rechenintensive Prozesse nach einer maximalen Zeit – bei Linux beträgt diese 20ms [RPG+99] – unterbrochen werden.

2.2 FPGA-Bausteine

Field Programmable Gate Array (FPGA) sind elektronische Bausteine die eine Unterklasse der programmierbaren Logikbausteine bilden. Sie entstanden als eine Weiterentwicklung der Programmable Array Logic (PALs) und sind seit ca. 1985 kommerziell verfügbar. Eine aktuelle Liste der FPGA-Bausteine kann in [Opt] eingesehen werden. Eingesetzt werden die FPGA-Bausteine zur Ausführung von kombinatorischen aber auch arithmetischen Funktionen die im nachfolgenden als Schaltungen bezeichnet werden. FPGA-Bausteine lassen sich durch folgende Merkmale charakterisieren:

Programmierbarkeit: Das wichtigste Merkmal der FPGA-Bausteine ist deren Programmierbarkeit. Die FPGA-Bausteine erlaubt es den

[1] Auswahlverfahren, das den nächsten zu verarbeitenden Prozess aus den wartenden Prozessen auswählt.

Entwickler jede Art von kombinatorischen oder arithmetischen Schaltungen, mit oder ohne Register, umzusetzen.

Arrayartige, zellenbasierte Struktur: Die FPGA-Bausteine verfügen über ein meist quadratisches Array von logischen Zellen die zur Ausführung der Schaltung entsprechend konfiguriert werden. Durch ein Zusammenschalten von mehreren Zellen über ein ebenfalls programmierbares Verbindungsnetzwerk können auch komplexere Schaltungsfunktionen, wie z.B. ein 32Bit-ALU, umgesetzt werden.

Rekonfigurierbar: Die Funktion der einzelnen Zellen und die Verbindungen des Netzwerks werden durch den Inhalt der sog. Konfigurationsebene bestimmt. Durch ein Umladen dieses SRAM basierten Speichers kann die Funktionalität innerhalb von wenigen 100ms verändert werden.

Abbildung 2.2 zeigt die für FPGA-Bausteine typische Architektur. Die Architektur wird getrennt in zwei unterschiedlichen Ebenen.

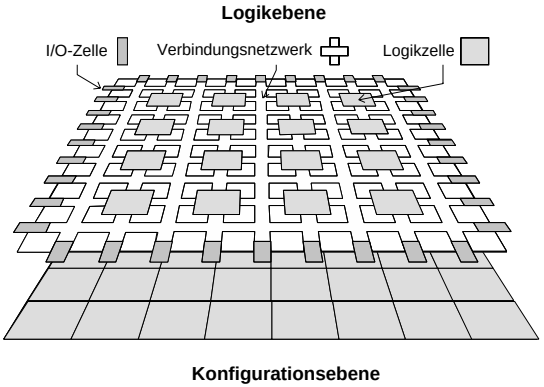


Abbildung 2.2: Aufbau der internen FPGA-Architektur *(Entnommen aus [Nof96])*

Die obere Logikebene zeigt den Zusammenhang der drei einzelnen ausführenden Komponenten: I/O-Zellen, Logikzellen und Verbindungsnetzwerk.

I/O-Zellen: Diese Zellen befinden sich am Rand des Arrays und bilden die bidirektionale Schnittstelle des FPGA-Bausteins zu der umgebenden Schaltung. Jede I/O-Zelle verfügt im allgemeinen über die notwendigen Ausgangstreiber und zwei Register.

Logikzellen: Auf die arrayförmig angeordneten Logikzellen wird die programmierte Schaltung abgebildet. Diese Logikzellen sind herstellerabhängig und variieren in der Ausführung. Im allgemeinen besitzen alle Logikzellen eine programmierbare kombinatorische Einheit (3–12 Eingänge/Zelle) und eine speichernde Einheit (1–8 Register/Zelle). Details der jeweiligen Ausführung sind für die Arbeit nicht relevant und können in den jeweiligen Datenbüchern nachgesehen werden [Xil99a, Alt00].

Verbindungsnetzwerk: Das Verbindungsnetzwerk ist das dritte Grundelement eines FPGA-Bausteins. Es verbindet die einzelnen Logikzellen miteinander, um die gesamte Schaltung abzubilden und zudem auch die Logikzellen mit den I/O-Zellen um die Funktionalität der Schaltung ausserhalb des FPGA-Bausteins zu nutzen.

Parallel zu dieser funktionalen Logikebene ist in der Abbildung 2.2 die untere Konfigurationsebene dargestellt. Sie repräsentiert den Konfigurationsspeicher des FPGA-Bausteins. Zwischen dieser Konfigurations- und der Logikebene besteht ein direkter Zusammenhang. D.h. jedes einzelne Bit des Konfigurationsspeichers bestimmt *eine* Funktion des Verbindungsnetzwerks oder einer Zelle auf der Logikebene. Der Inhalt dieses Speichers bestimmt somit die Funktionalität des gesamten Bausteins. Abbildung 2.3 stellt diesen Zusammenhang noch einmal graphisch dar.

Neben den allgemeinen charakteristischen Merkmalen der FPGA-Bausteine unterscheiden sich die einzelnen Bausteine-Familien durch Variationen im Aufbau der Zellen und des Verbindungsnetzwerks und durch zusätzliche bausteinspezifische Erweiterungen. Die wichtigsten Variationen und Erweiterungen sind nachfolgend aufgelistet und kurz erläutert:

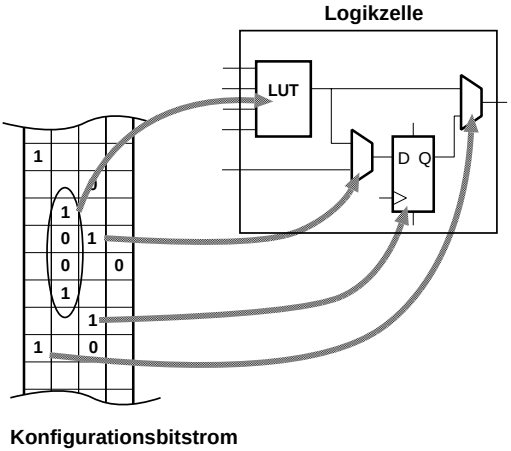


Abbildung 2.3: Vereinachte Darstellung des Zusammenhangs zwischen der Logik- und der Konfigurationsebene.

Logikzellengröße: Unterschieden wird in sog. 'Corse Grain'- und 'Fine Grain'-Architekturen. Bausteine mit einer 'Course Grain'-Architektur verfügen über große Logikzellen mit 8–24 Eingängen und 2–8 internen Registern. Angeboten werden diese z.B. von Xilinx [Xil99a] oder Lucent [Luc98a]. Demgegenüber bestehen die Logikzellen in einer 'Fine Grain'-Architektur meist aus nur einem Register und einer entsprechend verkleinerten kombinatorischen Einheit mit 3–8 Eingängen. Hauptvertreter sind Actel [Act99], Atmel [Atm00] oder auch die XC6200 Serie von Xilinx [Xil96].

Partielle Konfiguration: Verschiedene FPGA-Bausteine ermöglichen es dem Entwickler nur einen Teil der zur Verfügung stehenden Logikzellen zu verwenden und nur diese zu konfigurieren. Durch diese partielle Rekonfiguration können einzelne Prozesse parallel und unabhängig von anderen Prozessen bzw. der Konfiguration anderer Prozesse ausgeführt werden. Darüberhinaus werden auch die Konfigurationszeiten verringert, da nur ein Teil der gesamten Konfi-

gurationsdaten zu übertragen ist. Bausteine von Atmel [Atm99a] und von Xilinx [Xil96, Xil99b] ermöglichen eine partielle Rekonfiguration, aber aufgrund der unzureichenden Unterstützung durch die Entwicklungswerkzeuge können diese nicht praktisch genutzt werden.

Readback: Ein Readback wird, außer von Altera, von allen FPGA-Bausteinherstellern bereitgestellt. Er wird eingesetzt, um die aktuell geladene Konfiguration aus dem FPGA-Baustein auszulesen und diese auf eine korrekte Übertragung hin zu überprüfen. Zudem enthalten die Readbackdaten auch die aktuellen Zustände der Registern und der kombinatorischen Einheiten innerhalb des FPGA-Bausteins. Somit wird das Auslesen des internen Zustands der Schaltung ermöglicht.

Interne RAMs: Aufgrund der vielfach zum Einsatz kommenden RAM-Elemente innerhalb einer Schaltung wurden festverdrahtete RAM-Blöcke mit in das Array aus Logikzellen integriert. Diese ersetzen auf eine sehr effektive Weise die zu RAM-Elementen programmierten und verschalteten Logikzellen. Auch sie sind in nahezu allen modernen FPGA-Bausteinfamilien zu finden.

Multi-Context FPGAs: Multi-Context FPGA-Bausteine verfügen über mehrere Ebenen von Konfigurationsspeichern, die es erlauben die Funktionalität der Bausteine innerhalb von nur einem Taktzyklus (5-30ns) zu wechseln. Durch die unabhängigen Konfigurationsspeicher wird z.B. der einfache Aufbau von redundanten Systemen ermöglicht. Weitere Ziele sind die Vergrößerung der Logikressourcen und die Nutzung dieser Ressourcen zu unterschiedlichen Zeiten. Bausteine dieser Art sind sehr spezielle Entwicklungen, die sowohl von Sanders[SV98] als auch von Xilinx[Xil96] zu Evaluierungszwecken entwickelt, aber nie verkauft wurden.

Eingebetteter Microprozessor: Zusätzlich zu der Logikebene ist in modernen FPGA-Bausteinen auch ein Microprozessor integriert. Eine enge Kopplung zwischen dem FPGA und dem Microprozessor

erlaubt es nun auch komplexere Anwendungen verteilt laufen zu lassen. Derzeit wird nur ein Baustein der Firma Atmel [Atm99b] angeboten, der über einen eingebetteten Microprozessor verfügt.

Unabhängig von der Größe der FPGA-Bausteine und den speziellen Erweiterungen ist die Konfiguration der Bausteine. Erst die Konfiguration mit den Konfigurationsdaten bestimmt die Funktionalität des FPGA-Bausteins während des Betriebs. Die Erstellung des Konfigurationsbitstroms wird in Abschnitt 2.4 detailliert beschrieben. Das Laden der Konfiguration in den Konfigurationsspeicher erfolgt über eine entsprechende Schnittstelle. In der Regel stehen dazu mehrere Schnittstellen zur Verfügung. JTAG [AK96] ist eine standardisierte bitserielle Schnittstelle, die sowohl zum Testen der Bausteine im gesamten System, als auch zur Konfiguration eingesetzt wird. Eine weitere bitserielle Schnittstelle erlaubt es den Konfigurationbitstrom direkt aus einem angeschlossenen ROM auszulesen. Diese Art der Konfiguration wird meist bei Systemen angewendet, die nach dem Einschalten automatisch starten. Mehrere FPGA-Baustein Familien wie z.B. Xilinx Virtex[Xil99b] oder Lucent Orca 3C[Luc98b] verfügen zudem über eine weitere 8Bit breite Konfigurationsschnittstelle, die eine vergleichsweise schnelle Konfiguration ermöglicht.

2.3 FPGA-Prozessoren

Basierend auf der hohen Flexibilität und der Eigenschaft der Rekonfigurierbarkeit der FPGA-Bausteine sind seit ca. 10 Jahren verschiedene FPGA-Prozessoren entwickelt worden. Das Haupteinsatzgebiet dieser FPGA-Prozessoren ist die Beschleunigung vorhandener Anwendungen durch die parallele und systolische Verarbeitung der Daten in Hardware.

Die Vielzahl der unterschiedlichen entstandenen FPGA-Prozessoren kann in zwei Kategorien eingeteilt werden: Stand-Alone FPGA-Prozessoren und FPGA-Koprozessoren. Beide FPGA-Prozessor Kategorien und ihre jeweiligen spezifischen Merkmale werden nachfolgend näher erläutert.

2.3.1 Stand-Alone FPGA-Prozessoren

Diese Klasse an FPGA-Prozessoren werden meist für einen bestimmten Einsatz wie z.B. die Genomdatenanalyse[Tim], die ASIC-Prototypenentwicklung[Sta] oder als Triggerprozessor [HKL+95] in der Hochenergiephysik eingesetzt. Daneben zählen auch alle Evaluation-Boards, die von den verschiedenen Herstellern angeboten werden, zu den Stand-Alone FPGA-Prozessoren. Insgesamt machen sie mit ca. 60% den größten Teil aller FPGA-Prozessoren aus. Aufgrund der unterschiedlichen Einsatzgebiete sind diese Stand-Alone FPGA-Prozessoren sehr individuell und spezifisch aufgebaut.

Stand-Alone Systeme zählen zu den Datenflußrechnern, welche die jeweilige Anwendung unabhängig von einem Host-Prozessor ausführen. Es besteht keine Kopplung zwischen dem FPGA-Prozessor und dem Host-Prozessor, da dieser nur zur Konfiguration und zur Steuerung des Stand-Alone Systems eingesetzt wird. Für die vorliegende Arbeit ist der Einsatz dieser Stand-Alone Systeme nicht relevant.

2.3.2 FPGA-Koprozessoren

Auch in der Klasse der FPGA-Koprozessoren werden von verschiedenen Herstellern FPGA-Prozessoren angeboten, die allgemein verwendbar sind. Im Gegensatz zu den Stand-Alone Prozessoren basieren die FPGA-Koprozessoren auf einer starken Bindung zwischen dem Host-Prozessor und dem FPGA-Baustein. Die Mehrzahl (57%) der FPGA-Koprozessoren basiert auf dem PCI[SA95] Bus und kann somit schnell und effektiv Daten zwischen dem Host-Prozessor und dem FPGA-Baustein austauschen.

Zum Einsatz kommen die FPGA-Koprozessoren in den unterschiedlichsten Bereichen. Volumenvisualisierung[VHM+99], Mustererkennung [Hez, MSS00], Bioinformatik [SBM00] oder Krypthographie [SKS+00] sind nur einige Bereiche in denen sie erfolgreich eingesetzt werden. Dieses breite Einsatzspektrum der FPGA-Koprozessoren wird durch die hohen PCI-Datentransferrate von bis zu 120MByte/s[2] ermöglicht. Dadurch

[2] Der PCI-Bus besitzt eine theoretische Daterate von 132MByte/s die in der Praxis jedoch nur bei sehr großen Datenpaketen annähernd erreicht werden kann.

sind die FPGA-Koprozessoren sowohl zur Ausführung von datenintensiven als auch für rechenintensive Anwendungen gut geeignet. Desweiteren wird durch die Verwendung des PCI Busses eine sehr einfache und schnelle Integration in vorhandene Systeme ermöglicht.

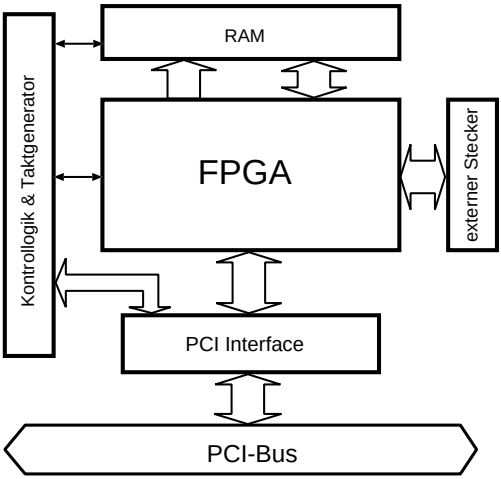


Abbildung 2.4: Allgemeiner Aufbau eines FPGA-Koprozessors

Der typische Aufbau eines FPGA-Koprozessors ist in Abbildung 2.4 dargestellt. Als zentrales Element zur Ausführung der Anwendungen kommt ein oder mehrere FPGA-Bausteine zum Einsatz. Verbindungen bestehen zu dem PCI-Bus, einem lokalen RAM und einem externen Stecker. Die PCI-Bus Verbindung ist in der Regel über einen PCI-Interfacebaustein an den Host-Prozessor angeschlossen. Das ebenfalls meist direkt angeschlossene RAM kann als statisches oder dynamisches RAM ausgeführt sein und der externe Stecker wird zum Anschluß weiterer Komponenten verwendet.

Diese Klasse der FPGA-Koprozessoren ist für diese Arbeit wichtig, denn nur diese verfügen über den notwendigen schnellen Datenaustausch zwischen Host- und FPGA-Prozessor, der von vielen Anwendungen benötigt wird.

2.4 Applikationsentwicklung für FPGA-Prozessoren

Das Konfigurieren eines einzelnen FPGA-Bausteins erfolgt durch Laden eines Bitstroms in den Konfigurationsspeicher. Aufgrund des sehr komplexen und in Teilen geheimen Aufbaus des Konfigurationsbitstroms erfolgt die Umsetzung einer Schaltung auf den FPGA-Baustein auf einer höheren, abstrakteren Ebene.

Dieser Abschnitt erläutert den prinzipiellen Ablauf des gesamten Entwicklungsprozesses für FPGA-Bausteine und damit auch den der FPGA-Prozessoren.

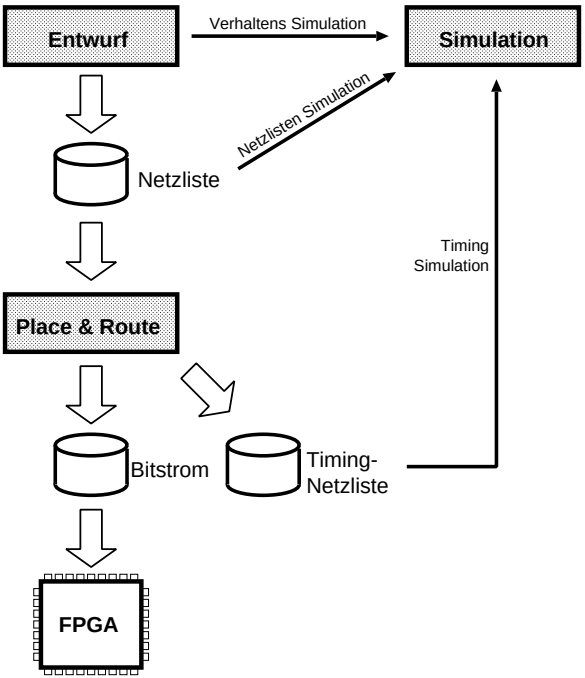


Abbildung 2.5: Entwicklungsprozess einer FPGA-Schaltung

Abbildung 2.5 zeigt die für die Entwicklung relevanten Stufen der Programmierung.

In der ersten Stufe entsteht ein anwendungsbezogener Entwurf der die Anforderungen, die an die FPGA-Schaltung gestellt werden, beschreibt. Zur Beschreibung dieses Entwurfs stehen dem Entwickler mehrere, in ihrer Abstraktion unterschiedliche, Beschreibungssprachen zur Verfügung. Im Detail werden diese in Abschnitt 2.4.2 beschrieben. Zur Validierung und zur Verifikation des erstellten Entwurfs besteht die Möglichkeit diesen mittels einer Simulation zu überprüfen. Die Simulation, die auf der Ebene der Entwurfsbeschreibung erfolgt, wird auch 'Verhaltens-Simulation' genannt. Nähere Beschreibungen zur Simulation sind in Abschnitt 2.4.3 zusammengefaßt.

Nach der erfolgreichen Simulation des Entwurfs wird dieser von der gewählten Programmiersprache mittels eines Synthesewerkzeugs in eine Netzliste umgewandelt. Diese Umsetzung erfolgt in zwei Schritten.

Der erste Schritt generiert aus der zugrundeliegenden Beschreibung ein sog. Register-Transfer(RT) Modell. Abbildung 2.6 zeigt diesen ersten Schritt der Umsetzung für ein in VHDL beschriebenen 2Bit Multiplizierer mit Ausgangsregister.

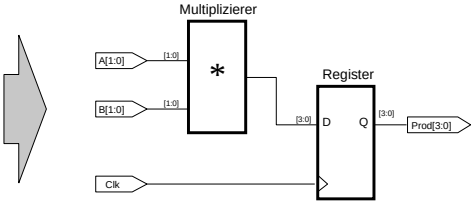
Das RT Modell bestehen aus zwei Grundelementen: Speichernde Komponenten (Register) und logische Komponenten (Logik). Die Register speichern mit jedem Takt den anliegenden Zustand und erzeugen somit eine synchrone Schaltung. Die Logik zwischen den Registern hingegen dient dazu die kombinatorischen und/oder arithmetischen Funktionen auszuführen, die auf die gespeicherten Variablen angewendet werden sollen. Vergleichbar ist dies zum Aufbau von Mealy- und Moore-Automaten [tH95], die zur Realisierung von Finite-State-Machines (FSM) verwendet werden.

In dem zweiten Schritt von der Umwandlung der Beschreibung in eine Netzliste wird das entstandene RT Modell in eine FPGA-Baustein spezifische Netzliste umgewandelt. Dies ist notwendig, da das RT Modell auf idealisierten Voraussetzungen, wie z.B. Logikelemente mit beliebig vielen Eingängen, basiert. Die idealisierte RT Modell Beschreibung wird dabei auf die bei der ausgewählten Zielarchitektur vorhandene Logikzellengröße abgebildet. Abschließend wird diese Abbildung in einer

```
entity Multiply is
  port (A:      in std_logic_vector(1:0);
        B:      in std_logic_vector(1:0);
        Clk:    in std_logic;
        Prod:   out std_logic_vector(3:0)
       );
end Multiply;


architecture Behave of Multiply is
begin
   Reg: process(clk)
   begin
      if(clk'event and clk=1) then
        Prod <= a * b;
      end if;
   end process;
end Behave;
```

VHDL



RT-Modell

Abbildung 2.6: Umsetzung eines 2Bit Multiplizieres in ein RT-Modell

Netzliste abgespeichert. Die Abbildung 2.7 verdeutlicht diesen zweiten Schritt noch einmal anhand des 2Bit Multiplizierer Beispiels. Basierend auf der erstellten Netzliste kann der Entwickler eine weitere Simulation durchführen. Diese 'Netzlisten-Simulation' basiert im Gegensatz zu der 'Verhaltens-Simulation' auf deterministischen Komponenten und läßt erste Aussagen über ein späteres Timingverhalten der Schaltung zu. Gerade dieses Timingverhalten der Logikkomponenten bestimmt die Leistungsfähigkeit der späteren Schaltung.

Nach erfolgter Verifikation und Optimierung des Entwurfs müssen die Komponenten der Netzliste auf die vorhandenen Logikzellen des ausgewählten Bausteins abgebildet werden. Sogenannte Place&Route-Werkzeuge übernehmen diese Aufgabe. Die Place&Route-Werkzeuge sind aufgrund der unterschiedlichen Architekturen der FPGA-Bausteine sehr bausteinabhängig und werden ausschließlich von den Bausteinherstellern angeboten. Durchgeführt wird dieser Schritt ebenfalls in zwei Stufen.

In der ersten Stufe werden die in der Netzliste beschriebenen Kom-

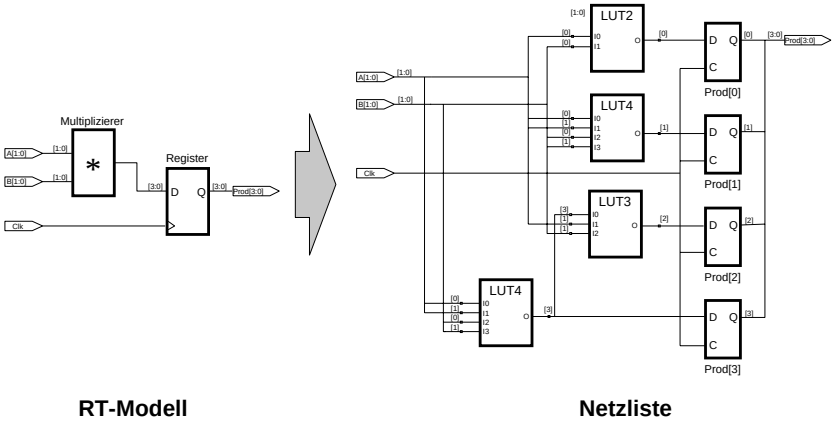


Abbildung 2.7: Umsetzung des RT-Modells eines 2Bit Multiplizierers in eine Netzliste

ponenten auf das Zellenarray plaziert (Placement). Der dabei zum Einsatz kommende Algorithmus versucht, unter Berücksichtigung von vorgegebenen Plazierungsinformationen, den Abstand zwischen verbundenen Komponenten möglichst gering zu halten. In der zweiten Stufe (Routing) werden dann die Netzwerkverbindungen, aber auch Verbindungen innerhalb der einzelnen Zellen erstellt. Das vorgegebene Verbindungsschema der Schaltung ist ebenfalls in der zugrundeliegenden Netzliste enthalten. Das Umsetzen der 'virtuellen' Komponentenverbindungen erfolgt durch das Setzen von Multiplexern und sog. Switch-Matrizen die zum einen Verbindungen zwischen horizontalen und vertikalen Signalleitungen innerhalb des Netzwerks ermöglichen und zum anderen den Anschluß der Logikzellen an diese Netzwerke herstellen.

Die Umsetzung der Netzliste des 2Bit Multiplizierers auf einen Xilinx Vitex FPGA-Baustein ist in Abbildung 2.8 zu sehen. Das im rechten Bild abgebildete Array stellt dabei nur einen Ausschnitt des insgesamt zur Verfügung stehenden Zellen-Arrays dar.

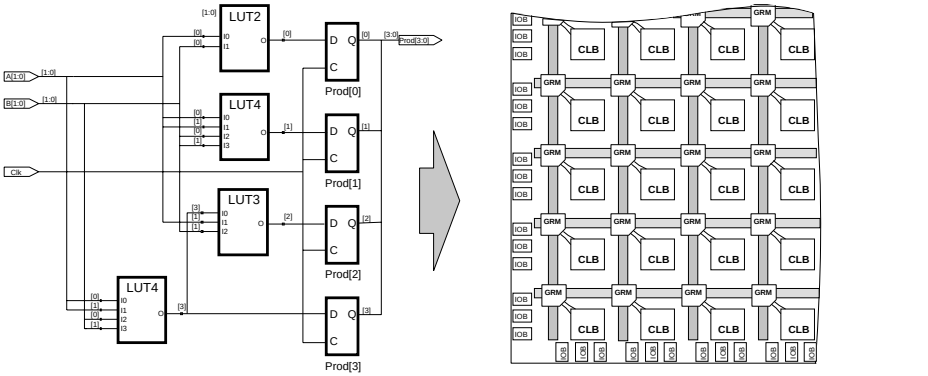


Abbildung 2.8: Plazieren und Verbinden der in der Netzliste beschriebenen Komponenten auf dem FPGA-Baustein. *Das Zellenarray des FPGA-Bausteins ist aus Gründen der Übersichtlichkeit ohne die plazierte Schaltung dargestellt.*

Nach der erfolgreichen Plazierung der Netzliste auf dem ausgewählten FPGA-Baustein wird diese mit Hilfe eines Bitstromgenerators in einen ladbaren Bitstrom umgewandelt. Der FPGA-Baustein kann danach mit diesem Bitstrom konfiguriert werden. Darüberhinaus kann aus der plazierten und gerouteten Schaltung wieder eine Netzliste erstellt werden, die sämtliche Timing-Informationen enthält. Basierend auf dieser Timing-Netzliste ist der Entwickler nun in der Lage eine 'Timing-Simulation' durchzuführen. Diese 'Timing-Simulation' ist vor allem bei Schaltungen wichtig in denen die zeitlichen Anforderungen sehr hoch sind.

Der Place&Route Schritt benötigt ca. 60–80% der Zeit von der Synthese des Entwurfs bis zu dem fertigen Konfigurationsbitstrom. Abhängig von der Schaltungskomplexität der Netzliste dauert ein Place&Route-Durchlauf zwischen ein paar Minuten und mehreren Stunden[Sin00].

Bedingt durch den mehrstufigen Programmierablauf, der in Abbildung 2.5 dargestellt ist, hängt das Ergebnis stark von den einzelnen Stufen ab. Neben dem Entwurf der Architektur, die im folgenden Abschnitt erläutert wird, ist das Place&Route-Werkzeug maßgeblich bestimmend für die spätere Leistungsfähigkeit der Schaltung und zudem auch für die Verwendbarkeit der FPGA-Bausteine selbst[3].

2.4.1 Entwurf

Der Entwurf der Schaltung für eine Anwendung ist der erste und zugleich wichtigste Schritt bei der Entwicklung. In diesem Abschnitt werden die Auswahlkriterien erläutert, die der Architekturentscheidung zugrundeliegen. Die hier beschriebenen Kriterien beziehen sich auf die Entwicklung von Anwendungen für FPGA-Koprozessor sind aber auch für Anwendungen für 'Stand-Alone' FPGA-Prozessoren gültig.

Primäres Ziel bei der Entwicklung von Anwendungen für FPGA-Koprozessor ist es, eine möglichst hohe Leistungsfähigkeit zu erreichen. Dies wird ermöglicht durch eine ausführliche Analyse der Anwendung, bei der vor allem die zu verarbeitenden Daten und Datenformate, der rechenintensive Programmteil mit seinen Rechenoperationen und das Zusammenwirken mit dem weniger rechenintensiven Programmteil untersucht wird. Basierend auf den analysierten Daten und den gegebenen Anforderungen, wie z.B. Timing-Vorgaben, wird eine Architektur entwickelt, die sowohl den Host-Prozessor als auch den FPGA-Koprozessor optimal ausnutzt. Die angewandten Kriterien für eine optimale Nutzung sind dabei für jeden Teilbereich (Host-Prozessor, FPGA-Koprozessor und Kommunikation) unterschiedlich.

Die Kriterien für den Host-Prozessor sind im wesentlichen bezogen auf die Art und Weise der Datenspeicherung. Dabei sollte ein zusammenhängendes Format gewählt werden, das eine effektive Datenübertragung ermöglicht und die direkte Datenverarbeitung auf dem FPGA-Koprozessor zulässt. Bei der Ausgliederung der rechenintensiven Pro-

[3]Sobald die Place&Route-Werkzeuge die Netzlisten nicht oder nur schlecht auf die Arraystruktur abbilden, kann auch der FPGA-Baustein nur bedingt eingesetzt werden. Dies gilt für die Bausteine der Xilinx XC6200 Familie[HR98b, Bre97a] und Bausteine der Atmel AT40K Serie [Sim98].

grammteile ist darauf zu achtet, daß der Host- und der FPGA-Koprozessor parallel arbeiten. Dies ist z.B. der Fall, wenn der Host-Prozessor einen Datenblock vorbereitet während der FPGA-Prozessor einen anderen Datenblock verarbeitet.

Das alleinige Kriterium für die optimale Kommunikation zwischen Host- und FPGA-Prozessor ist die Datenblockgröße. Wie in Abschnitt 5.6.2 noch gezeigt wird, ist der Datendurchsatz auf dem häufig verwendeten PCI-Bus nur bei großen Datenblöcken optimal.

Der FPGA-Koprozessor hat die Aufgabe, die übertragenen Daten entsprechend der Anwendung schnell zu verarbeiten. Aus diesem Grund beziehen sich die Kriterien für eine effektive und damit optimale Umsetzung ausschließlich auf die Schaltung des FPGA-Bausteins. Drei grundlegende Konzepte der Schaltungsarchitektur, die nachfolgend näher beschrieben werden, kommen dabei zum Einsatz. Abbildung 2.9 zeigt diese Grundkonzepte anhand von einfachen Beispielen.

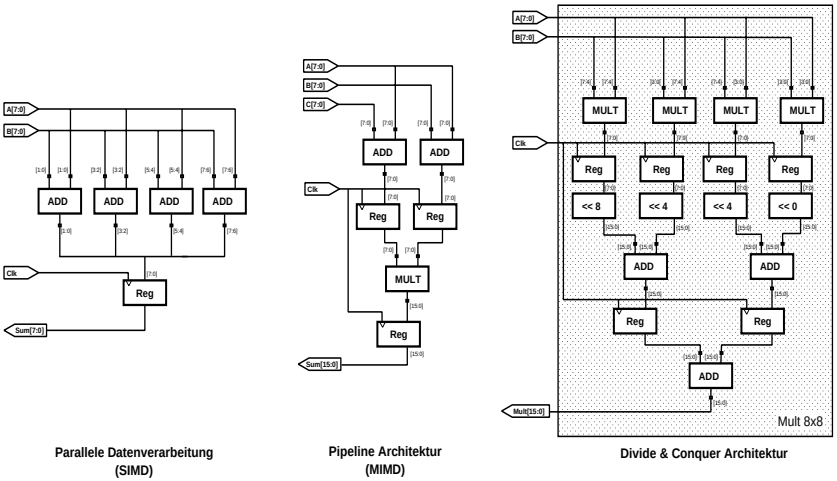


Abbildung 2.9: Grundlegende Schaltungskonzepte

Parallele Verarbeitung der Daten ist ein wichtiges Kriterium für eine optimale Umsetzung der Anwendung. Die parallele Datenverarbeitung bezieht sich aber nur auf gleichzeitig zu verarbeitende Daten (SIMD), ähnlich wie bei der MMX-Erweiterung der Intel Prozessoren [Inc99].

Pipeline Verarbeitung der Daten erlaubt es, über die parallele Datenverarbeitung hinaus, auch Operationen parallel zu verarbeiten (MIMD). Die einzelnen Operationen, die auch voneinander anhängen können, werden nacheinander von unabhängigen Stufen verarbeitet und die jeweiligen Zwischenergebnisse in Registern zwischengespeichert. Daher ist auch dieses Grundkonzept ein sehr wichtiges Kriterium.

Divide & Conquer ist ein Konzept das komplexe und umfangreiche logische oder arithmetische Funktionen in kleinere Funktionen aufteilt, die innerhalb von mehreren Takten abgearbeitet werden. Der Aufbau dieser komplexen Funktionen folgt dabei den Richtlinien der Pipeline Verarbeitung.

Während die ersten beiden Konzepte auf eine möglichst optimale Verarbeitung der Daten gerichtet sind, wird das letzte Divide & Conquer Konzept eingesetzt, um die Taktrate und damit die Verarbeitungsgeschwindigkeit der Schaltung zu erhöhen. Vereinfacht kann man sagen, daß die Leistungsfähigkeit der gesamten Schaltung mit dem Grad der Parallelität, aber auch mit der Taktrate zunimmt.

Die oben genannten Kriterien für die zu entwickelnde Schaltung alleingenommen führen aber zu einem in der Regel nicht optimalen Gesamtsystem. D.h. die Verarbeitung der Daten kann optimal sein, aber durch eine sehr geringe Datenrate auf dem PCI-Bus, bedingt durch kleine zu übertragene Datenblöcke, ist der Gesamtdurchsatz nur geringfügig höher als bei einer reinen Host-Prozessor Lösung. Aus diesem Grunde gilt es bei dem Entwurf der gesamten Architektur alle kritischen Punkte zu bedenken, um eine leistungsstarke, optimale Anwendung zu erhalten.

2.4.2 Beschreibungssprachen

Zur Beschreibung der, in der Entwurfsphase entstandenen Architektur, stehen dem Entwickler verschiedene Beschreibungssprachen zur Verfügung. Alle diese Sprachen bilden die beschriebene Schaltung zuerst in ein RT Modell und danach in eine FPGA-Baustein spezifische Netzliste ab. Die Unterschiede der einzelnen Sprachen liegen zum einen in der Abstraktion zu dem RT Modell und zum anderen in der Möglichkeit der Entwurfssimulation.

Nachfolgend werden die in drei unterschiedliche Abstraktionsebenen unterteilten Beschreibungssprachen kurz beschrieben:

Modulgeneratoren gehören zu den strukturbeschreibenden Sprachen die eine, zu den RT Modellen sehr ähnliche, Struktur aufweisen. Zu diesen Modulgeneratoren zählen die Sprachen CHDL [KBH+98] oder PamBlox [MMF98], aber auch Schaltplaneingaben wie z.B. ORCAD. Kennzeichnend für die Modulgeneratoren ist eine Anzahl von Basiskomponenten die entsprechend der Anwendung miteinander verschaltet werden. Zu diesen Basiskomponenten gehören unter anderem Logikelemente, Register, Addierer, Zähler und kleine RAM-Blöcke. Durch Zusammenschalten der Basiskomponenten wird der Entwickler in die Lage versetzt, die Bauteilbibliothek dynamisch durch komplexere Bauteile, wie z.B. Multiplizierer, zu erweitern.

Die Verifikation und Simulation der Schaltung erfolgt direkt aus der Entwicklungsumgebung oder, wie im Fall von CHDL, direkt aus der beschriebenen Schaltung heraus.

Verhaltensbeschreibende Sprachen wie z.B. VHDL [Per98, PT97] oder Verilog [Pal96] bieten dem Entwickler die Möglichkeit, Struktur- mit Verhaltensbeschreibung zu kombinieren. Aufgrund der Standardisierung dieser beiden Sprachen stellen die Hersteller der FPGA-Bausteine eigene Bibliotheken mit allen Basiskomponenten (Logic, Register, usw.) bereit. Diese Basisfunktionen können durch Verhaltensbeschreibungen ergänzt werden, was zu einer einfacheren Schaltungsbeschreibung führt. Auch bei den zwei genannten

Sprachen – VHDL und Verilog – ist der Entwickler in der Lage eine eigene Bauteilbibliothek zu generieren.

Die Simulation der entwickelten Schaltungen erfolgt auch hier mittles einer Beschreibung einer Simulationsumgebung in der gleichen Sprache. Eine Besonderheit, die verschiedene VHDL/Verilog Umgebungen bieten, ist die zusätzliche Simulationsmöglichkeit von Netzlisten. Dadurch ist man in der Lage, die drei verschiedenen Simulationen mit nur einem Simulationsprogramm durchzuführen.

Hochsprachen hingegen bieten dem Entwickler eine Beschreibungsebene, die vergleichbar zu einer imperativen Programmiersprache wie 'C' ist. Sowohl HandelC [Han99, Han98] als auch PPC [Zoz97], zwei Beispiele hardwarebeschreibender Hochsprachen, verwendet eine Untermenge der 'C'-Syntax als Basissyntax, die um parallele und pipelinebeschreibende Befehle erweitert wurden. In beiden Beispielsprachen besteht die Möglichkeit, Funktionen zu beschreiben. Umgesetzt werden die beschriebenen Schaltungen im Fall von HandleC direkt in eine Netzliste und bei PPC in eine entsprechend strukturierte VHDL Bescheibung.

Die unterschiedlichen Ausgangsbeschreibungen machen sich bei der auch hier notwendigen Simulationsumgebung bemerkbar. Während HandelC einen eigenen auf der HandleC-Syntax basierenden Simulator benötigt, kann PPC die verfügbaren Simulationsumgebungen der VHDL Werkzeuge nutzen.

In der Praxis werden vor allem die angegebenen Modulgeneratoren und verhaltensbeschreibende Sprachen verwendet. Sie verfügen über sehr gute Simulationsmöglichkeiten und erlauben aufgrund ihrer Abstraktionsebene die Entwicklung von sehr leistungsstarken Schaltungen. Die Hochsprachen hingegen sind konzipiert, um dem Softwareentwickler die Möglichkeit zu geben, die vorhandenen Algorithmen einfach und schnell auf Hardware abzubilden. Verbunden mit dieser sehr hohen Abstraktion ist die nicht optimale Umsetzung, die die Leistungsfähigkeit der erzeugten Schaltungen einschränkt.

2.4.3 Design Verifikation

Der gesamte Entwicklungsprozess einer FPGA-Schaltung ist begleitet durch die Simulation des Entwurfs, der Netzlisten und der plazierten Schaltung. Die ausführliche Simulation innerhalb des Entwicklungsprozesses ist notwendig, da im Betrieb auf dem FPGA-Prozessor die Funktionalität nur als 'Black-Box' durch Anlegen von Signalen getestet werden kann. Ziel der einzelnen Simulationen ist es, sowohl der Entwurf als auch die Schaltung auf deren korrekte Ausführung hin zu verifiziert und hinsichtlich der geforderten Anforderungen zu evaluieren.

Innerhalb des, in Abbildung 2.5 dargestellten, Entwicklungsprozesses werden daher mehrfach Simulationen durchgeführt. Die 'Verhaltens-Simulation' überprüft den Entwurf auf dessen korrekte Verarbeitung der Daten und das Zusammenwirken der einzelnen unabhängigen Komponenten. Diese Simulation basiert auf der jeweiligen abstrakten Entwurfsbeschreibung und lässt daher nur eine funktionale Verifikation zu. Nach dem Übersetzen der Beschreibung in eine Netzliste folgt die 'Netzlisten-Simulation', die ebenfalls die Funktionalität der gesamten Schaltung auf der Basis der Logikzellen-Abbildung verifiziert. Darüberhinaus werden durch die Abbildung auf eine Netzliste erstmals Aussagen über das zeitliche Verhalten und damit über die zu erwartende Leistungsfähigkeit der Schaltung möglich. Das endgültige zeitliche Verhalten der Schaltung lässt sich aber erst nach dem Place&Route-Vorgang durch die 'Timing-Simulation' bestimmen. Die Timing-Netzliste enthält zu diesem Zweck die Verzögerungszeiten der FPGA internen Logikelemente und Netzwerkverbindungen.

Alle drei oben genannten Simulationen werden auf der Ebene der Eingangs- und Ausgangssignale der jeweiligen FPGA-Schaltung durchgeführt. In der nachfolgenden Abbildung 2.10 ist die 'Verhaltens-Simulation' des 2Bit Multiplizieres für verschiedene Eingangszahlen dargestellt.

Die Auswahl der Testdaten mit denen die Schaltung verifiziert werden erfolgt in der Regel durch den Entwickler. Eine sogenannte 'Testbench' ermöglicht es dem Entwicker, die ausgewählten Testdaten direkt oder aus einer Datei zu lesen und sie in Form von Eingangsignalen an die zu testende Schaltung anzulegen. Ebenfalls über die 'Testbench' werden

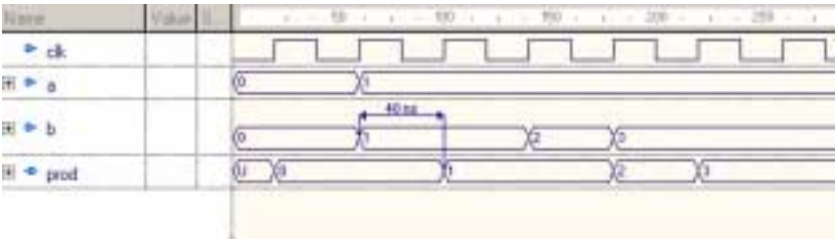


Abbildung 2.10: Waveform Output der 'Verhaltens-Simulation' des 2 Bit Multiplizierers

Ausgangssignale und Ergebnisse ausgelesen und abgespeichert, um z.B. im Fall von Bilddaten diese nach der Simulation anzuzeigen.

Für FPGA-Koprozessoren ist es jedoch wichtig die Simulation aus Sicht des Host-Prozessor durchführen zu können, um das Zusammenwirken der einzelnen an einer Anwendung beteiligten Komponenten verifizieren und evaluieren zu können. Aus diesem Grund wurden für die Beschreibungssprachen CHDL und VHDL spezielle Umgebungen erstellt, die eine Simulation des gesamten FPGA-Koprozessors ermöglichen.

2.5 Zusammenfassung

In diesem Kapitel wurden die Grundlagen des Multitasking Betriebs beschrieben, welche es ermöglicht mehrere unabhängige Prozesse parallel zu verwalten und zu verarbeiten. Die hardwareseitige Unterstützung der Prozessoren wurde am Beispiel des Intel 80486 Prozessors aufgezeigt. Zusätzlich wurden die Aufgaben des Betriebssystems zur Durchführung des Multitasking Betriebs kurz erläutert.

Eine zweite, für diese Arbeit wichtige, grundlegende Beschreibung erläutert den Aufbau und die Eigenschaften von FPGA-Bausteinen und von FPGA-Prozessoren. Neben den allgemeinen Merkmalen der FPGA-Bausteine werden verschiedene, bausteinfamilientypische Variationen und Erweiterungen genannt.

Der für die Umsetzung von Anwendungen notwendige Entwicklungsprozess wird anhand des Beispiels eines 2Bit Multiplizierers schrittweise näher erläutert. Beschrieben werden die einzelnen Stufen, die ein Design innerhalb dieses Entwicklungsprozesses durchläuft und die Werkzeuge, die dazu verwendet werden. Unter dem Gesichtspunkt der Anwendungsentwicklung für FPGA-Koprozessoren werden die dafür notwendigen Kriterien beschrieben. Abschließend werden die verschiedenen Beschreibungssprachen kurz beschrieben und die Form der Schaltungsverifikation erläutert.

Kapitel 3

Preemptive Multitasking auf FPGAs

Durch die im vorherigen Kapitel beschriebene Eigenschaft der Rekonfiguration der FPGA-Bausteine ist es möglich, verschiedenste Anwendungen auf einem FPGA-Prozessor beschleunigt auszuführen. Dies wird derzeit in den verschiedensten Bereichen durchgeführt, wobei der FPGA-Prozessor zu einem Zeitpunkt jeweils nur für eine spezifische Anwendung eingesetzt wird.

Die zusätzliche schnelle Rekonfiguration der Bausteine ermöglicht es nun, mehrere dieser Anwendungen auf einem FPGA-Prozessor parallel auszuführen, indem der FPGA-Baustein die einzelnen Anwendungen zeitlich voneinander getrennt verarbeitet. Diese, von preemptiven Multitasking Systemen bekannte Ausführung durch ein Betriebssystem bietet somit auch für FPGA-Prozessoren die Vorteile, die durch die parallele Ausführung entstehen. Obgleich die FPGA-Bausteine für eine echte parallele Nutzung geeignet sind, muß ein solches Betriebssystem auch hier in der Lage sein, eine Anwendung in der Ausführung zu unterbrechen und zu einem späteren Zeitpunkt wieder fortzuführen.

Das Ziel ist es, basierend auf der Grundlage der Rekonfiguration der FPGA-Bausteine und der Rekonstruktion der Anwendungen auf

dem FPGA-Prozessoren, ein Betriebssystem aufzubauen, das ein preemptives Multitasking mehrerer unabhängiger Anwendungen auf einem FPGA-Koprozessor ermöglicht. Zugleich ist dieses Betriebssystem allgemeingültig, d.h. es verarbeitet alle Anwendungen und basiert auf verfügbaren FPGA-Koprozessoren.

Dieses Kapitel beschreibt die Umsetzung der Idee eines preemptiven Multitaskings für FPGA-Bausteine mit allen Randbedingungen und Anforderungen an die zur Verwendung kommenden FPGA-Koprozessoren. Zu Beginn des Kapitels werden die speziellen FPGA-Bausteine beschrieben, die ein solches preemptives Multitasking unterstützen, aber auch die bisherigen Betriebssysteme für FPGA-Prozessoren werden näher erläutert. Im Anschluß daran wird das für FPGA-Prozessoren neuartige Konzept des preemptiven Multitasking beschrieben. Im Detail wird auf die Umsetzung der Funktionalität und den Ablauf eines Prozesswechels eingegangen. Durchgeführte Messungen bilden abschließend die Grundlage für ein darauf aufbauendes Betriebssystem.

3.1 Motivation

Die Rekonfigurierbarkeit der FPGA-Bausteine ist der Schlüsselfaktor für eine allgemeine Verwendbarkeit der FPGA-Prozessoren. D.h. viele Anwendungen aus verschiedensten Bereichen wie z.B. der Bildverarbeitung, der Kryptographie oder der Bioinformatik können durch den Nutzen eines FPGA-Prozessors eine Beschleunigung erfahren, die den Nutzen der Anwendungen weiter steigern. Derzeit ist die Nutzung der Ressource FPGA-Prozessor jedoch immer nur einer Anwendung vorbehalten, die von den Vorzügen der beschleunigen Datenverarbeitung profitieren kann. Dies begründet sich durch die direkte Verwendung des FPGA-Prozessors über den Geräte-Treiber, der exklusiv nur von jeweils einem Prozess geöffnet werden kann.

Durch die Einführung eines preemptiven Betriebssystems für FPGA-Prozessoren wird dieser mehreren Anwendungen gleichzeitig zur Verfügung gestellt. Das Betriebssystem, das aus mehreren Programmen besteht, steuert dabei die Ausführung der Anwenderprogramme auf dem FPGA-Prozessor. Daraus resultierend entstehen durch die Verwendung eines solchen preemptiven Betriebssystems folgende Vorteile:

- Mehrfachnutzung des FPGA-Prozessor durch verschiedene Anwendungen und Benutzer
- Bessere Auslastung des FPGA-Prozessors
- Vermeidung von blockierenden Situationen
- Priorisierte Verarbeitung der Prozesse
- Standardisierung der Benutzerschnittstelle
- Unterstützung mehrerer FPGA-Prozessoren

Diesen Vorteilen des preemptiven Betriebssystem steht ein systembedingter Overhead gegenüber. Dieser Overhead entsteht durch die Zeit, in der der FPGA-Prozessor nicht durch die Anwendung genutzt werden kann und verlängert die Ausführungszeiten der einzelnen Anwendungen. Das Betriebssystem, das während dieses Overheads ausgeführt wird, verwaltet dabei die einzelnen Prozesse und schaltet von einem auf den anderen Prozess um.

Aus den genannten Vorteilen und den Einschränkungen durch ein solches Betriebssystem ergeben sich mehrere Randbedingungen, die beim Aufbau eines Betriebssystems beachtet werden müssen:

- Das Betriebssystem muß in der Lage sein, eine laufende Anwendung auf dem FPGA zu unterbrechen und ihn zu einem späteren Zeitpunkt fortzusetzen (preemptives Multitasking).
- Die Zeit für einen Prozesswechsel muß gering sein, um eine minimale Beeinflußung der Ausführungszeit zu erreichen.
- Der systembedingte Overhead für die einzelne Operation muß vernachlässigbar sein.
- Das Betriebssystem soll verschiedene FPGA-Koprozessoren unterstützen.
- Eine einheitliche Benutzerschnittstelle muß für den Zugriff auf den FPGA-Prozessor bereitgestellt werden.

3.2 Stand der Technik

In der Vergangenheit wurden verschiedene unterschiedliche Ansätze verfolgt, um den FPGA-Prozessor für mehreren Anwendungen gleichzeitig nutzen zu können. Alle diese Ansätze basieren auf der Programmierbarkeit und der Rekonfigurierbarkeit der FPGA-Bausteine und nutzen diese beiden Eigenschaften um mehrere Anwendungen gleichzeitig zu verarbeiten. Obwohl alle das gleiche Ziel verfolgen basieren die entstandenen Systeme auf drei unterschiedlichen Grundkonzepten:

- Der hardwarebasierte Ansatz eines Multi-Context FPGA-Bausteins, der mehrere Konfigurationen gleichzeitig speichert
- Ein Betriebssystem, das eine echte parallele Nutzung von FPGA-Logik durch verschiedene Prozesse ermöglicht
- Ein Betriebssystem zur Verwaltung mehrerer nacheinander auszuführender Prozesse

Die einzelnen Konzepte und die daraus entstandenen Systeme werden in den nachfolgenden Abschnitten näher erläutert.

3.2.1 Multi-Context FPGA-Bausteine

Multi-Context FPGA-Bausteine wurden von zwei Herstellern unabhängig voneinander entwickelt und sollten mehrere, damals bestehende Einschränkungen eleminieren.

Die Motivation für die Entwicklung eines Multi-Context FPGAs bei der Firma Sanders [SV98] und der Firma Xilinx [TCJW97] waren:

1. Die Logikkapazität des FPGA-Bausteins zu erhöhen,
2. durch zeitliches multiplexen mehrere Anwendungen verarbeiten zu können und
3. durch die gleiche Konfigurierung der mehreren Konfigurationsebenen redundante Systeme aufzubauen.

Die Vorgehensweise bei den beiden Multi-Context FPGAs ist im wesentlichen die Gleiche. Durch das Einführen von zusätzlichen Konfigurationsebenen, in denen die Programmierung der Logikebene gespeichert ist, ist der FPGA-Baustein in der Lage, seine Funktionalität innerhalb von wenigen Nanosekunden zu wechseln. Abbildung 3.1 illustriert die entstandene FPGA-Architektur. Zusätzlich zu der Konfigurationsebene

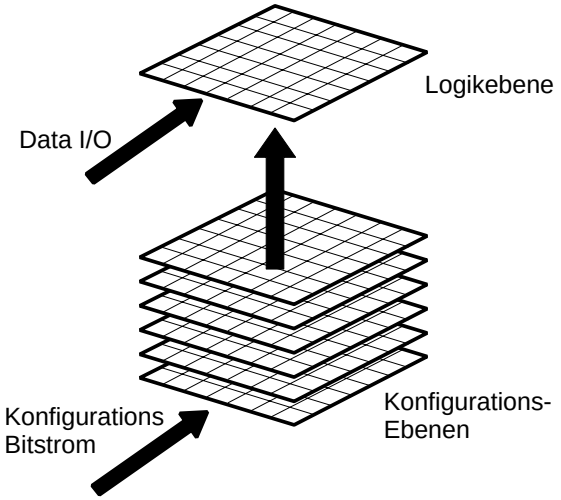


Abbildung 3.1: Konzeption der Multi-Context FPGA-Bausteine.

wurde auch die Logikebene für den Multi-Context Betrieb verändert. Die Register wurden so erweitert, daß sie die Registerinhalte jeder Konfiguration statisch speichern. Eine weitere Veränderung ermöglicht es, diese Registerinhalte von einer zur anderen Konfiguration zu übertragen. Dieser konfigurationsübergreifende Datentransfer erfolgt über die Register. Durch diese verteilte Speicherung der gesamten Registerinhalte einer Konfiguration ist ein Wechsel der Anwendung innerhalb von nur einem Takt möglich. Die Konfigurationswechsel werden von außen ausgelöst und können somit auch den externen Zustand der Anwendung berücksichtigen.

Der große Vorteil für den Einsatz dieser Bausteine in einem Multitasking Betriebssystem ist die Möglichkeit, eine Konfiguration innerhalb von nur wenigen Nanosekunden (5-30ns) zu verändern. Darüberhinaus erfolgt die Konfiguration der Anwendung als Hintergrundprozess während der Baustein einen andere Anwendung ausführt.

Demgegenüber verfügen diese FPGA-Bausteine nur über eine kleine Anzahl an Logikzellen. Aufgrund der schnellen Entwicklung der FPGA-Bausteine verfügt ein moderner Baustein mit einer Konfigurationsebene über mehr Ressourcen als der Multi-Context FPGA-Baustein mit allen Konfigurationsebenen zusammen und der heutige Baustein kann diese im Gegensatz zu der gemultiplexten Logik voll und permanent nutzten. Darüberhinaus waren die Unterstützung durch die Entwicklungswerkzeuge nur unzureichend, so das z.B. der Datenaustausch zwischen den einzelnen Konfigurationen, nur mit großem Aufwand genutzt werden konnte. Bedingt durch den Entwicklungsstatus der Bausteine sind auch die wichtigen Place&Route-Werkzeuge nur mit Einschränkungen nutzbar.

Diese Multi-Context FPGAs verfügen über eine sehr gute Unterstützung für ein Multitasking Betriebssystem, da sie schnell zwischen zwei Anwendungen umschalten können. Dennoch bestehen bei der Betrachtung des gesamten Ablaufs von der Konzeptionierung über die Simulation bis hin zur Ausführung der Anwendungen erhebliche Mängel, die eine einfache Anwendungsentwickung unmöglich machen.

Aufgrund der schnellen Entwicklung in dem Bereich der FPGA-Bausteine und des großen Aufwands bei der Optimierung der Place&Route Software wurden die Entwickungen beider Multi-Context FPGA-Bausteine von den jeweiligen Firmen eingestellt. Derzeit sind keine Bausteine verfügbar, die eine ähnliche Funktionalität bereitstellen.

3.2.2 Parallele Nutzung der FPGA-Logik

Die parallele Nutzung der Logik innerhalb eines FPGA-Bausteins wird bei dem Ansatz von G. Brebner verfolgt um mehrere Anwendungen gleichzeitig ausführen zu können [Bre97b]. Das Ziel ist die Entwicklung eines Frameworks, das sowohl für die Entwicklung der einzelnen An-

wendungen als auch deren Plazierung und gegenseitige Kommunikation beinhaltet.

Verwendet wird ein spezieller FPGA-Baustein von der Firma Xilinx mit der Bezeichnung XC6216 [Xil96]. Dieser verfügt über eine schnelle 32Bit parallele Konfigurationsschnittstelle, die eine für den Ansatz notwendige partielle Rekonfiguration des Bausteins ermöglicht. Desweiteren umfasst der Konfigurationsbitstrom des genannten Bausteins aufgrund des verhältnismäßig einfachen Aufbaus der Logikzellen und des Verbindungsnetzwerks nur ≈100kBit. Diese beiden Eigenschaften des Bausteins ermöglichen ein schnelles Wechseln der Konfiguration in ≈0,2ms.

Die parallel auszuführenden Anwendungen werden bei der Entwicklung auf kleine quadratische Logikblöcke abgebildet, die 'Swappable Logic Units' (SLU) genannt werden. In ihrer Position festgelegte Schnittstellen ermöglichen eine Kommunikation benachbarter SLUs untereinander bzw. den Zusammenschluß mehrerer SLUs zu komplexeren Funktionen. Dies ist zur Verdeutlichung in Abbildung 3.2 dargestellt. Das Betriebssystem steuert die Verwendung der einzelnen SLUs und deren Positionierung innerhalb des FPGA-Bausteins. Eine weitere Aufgabe des Betriebssystems ist die Durchführung der Kommunikation zwischen dem aufrufenden Prozess auf dem Host-Prozessor und der konfigurierten SLU auf dem FPGA-Prozessor. Begünstigt durch die Eigenschaft des direkten Zugriffs über die Konfigurationsschnittstelle auf die internen Register innerhalb des FPGA-Bausteins ist das Betriebssystem in der Lage jede verwendete SLU individuell mit Daten zu versorgen. Dieser Datenaustausch basiert stets auf den Registern. Eine Verwendung von externen Bausteinen wie RAMs ist nicht vorgesehen.

Die Funktionalität des Betriebssystem wurde für logische und arithmetische Funktionen gezeigt, die auf Basis von Registerwerten arbeiten. Aus Sicht des Host-Prozessors wurde somit der Befehlssatz mit speziellen benutzerdefinierten Maschinenbefehlen erweitert, die auf den FPGA-Prozessor ausgeführt werden.

Bei dem Ansatz erfolgt eine echte parallele Verarbeitung der Daten in unabhängigen Logikbereichen, den SLUs. Zusammen mit dem entwickelten Betriebssystem wird die Durchzuführung einzelner, komplexer Maschinenbefehle auf dem FPGA-Prozessor erreicht.

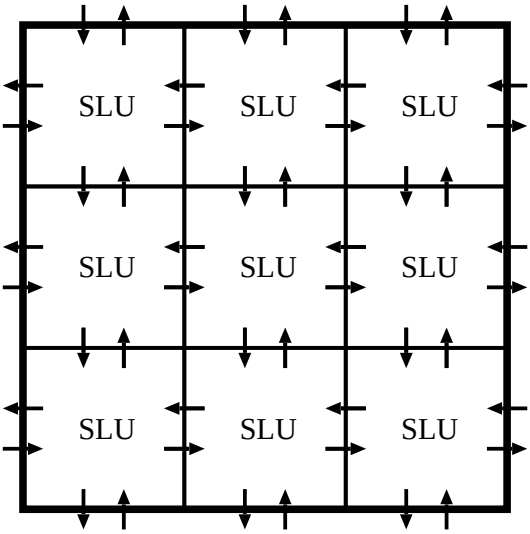


Abbildung 3.2: Aufteilung der vorhandenen Logikfläche zur Ausführung der SLUs

Dennoch ist bedingt durch die registerbasierte Kommunikation[1] nur eine sehr geringe Datenübertragungsrate zwischen den Host- und dem FPGA-Prozessor zu erreichen, was eine effektive Verwendung des Betriebssystems ausschließlich auf Maschinenbefehle mit nur wenigen Parametern eingeschränkt. Durch diese Einschränkung reduzieren sich die Aufgaben des eingesetzten Betriebssystems auf die Verwaltung der einzelnen SLUs und die Koordinierung der Zugriffe. Eine Sicherung des Zustandes im Falle eines SLU Wechsels ist nicht notwendig, da die Maschinenbefehle kombinatorisch sind und durch die Registerwerte definiert sind. Somit kann das Betriebssystem die einzelnen SLUs jederzeit überschreiben, um sie für andere Anwendungen einzusetzen.

[1] Die Datenrate bei einzelnen Registerzugriffen liegt für Lesezugriffe bei $\approx$ 6Mbyte und für Schreibzugriffe bei $\approx$ 25MByte [Mue00].

Dieses entwickelte Betriebssystem zeigt, daß für eine Klasse von Anwendungen die parallele Verarbeitung auf dem FPGA-Prozessor erfolgen kann. Zu diesem Zweck ist ein umfassendes Gesamtkonzept entwickelt worden. Dennoch ist dieses Konzept aufgrund des zum Einsatz kommenden Bausteins XC6216 mit seiner einzigartigen Konfigurationsschnittstelle und der partiellen Rekonfigurierbarkeit nur bedingt allgemeingültig. Der Einsatz dieses Bausteins schränkt zudem die Programmierbarkeit weiter ein, denn die Unterstützung der Entwicklungswerkzeuge ist für diesen Baustein mangelhaft [Bre97a, HR98b]. Die Ausnutzung der partiellen Rekonfigurierbarkeit erfordert den Einsatz eines Algorithmus zur Plazierung der SLUs auf den FPGA-Baustein. Dieser ist vor allem bei Anwendungen mit mehreren verbundenen SLUs komplex und führt nur selten zu einer vollen Auslastung der zur Verfügung stehenden Logikfläche. Als letzte Eigenschaft des Betriebssystems ist die Einschränkung auf nur eine Klasse von Anwendungen zu nennen. Nur registerbasierte Maschinenbefehle können effektiv ausgeführt werden, denn Anwendungen, die große Datenmengen austauschen, erfordern die exklusive Nutzung der Kommunikationsmechanismen zwischen Host- und FPGA-Koprozessor bzw. die Nutzung von lokalen RAM-Bänken. Diese werden durch das Konzept nicht abgedeckt.

Die mangelnde Unterstützung durch die Entwicklungswerkzeuge, maßgeblich durch das Place&Route-Werkzeug, hat die mögliche Verwendung des Bausteins XC6216 soweit eingeschränkt, daß dieser seit 1999 nicht mehr erhältlich ist.

Ein ähnlicher Ansatz für ein Betriebssystem, das im Gegensatz dazu auf einem FPGA-Baustein ohne spezielle Unterstützung basiert, wurde von J. Jean [JTY+98] umgesetzt. Im Unterschied zum vorher beschriebenen Betriebssystem wird dort der FPGA-Baustein nicht partiell rekonfiguriert. Somit steht jeder Anwendung die gesamte Logik des Bausteins zur Verfügung während das Betriebssystem nur eine verwaltende Funktion ausführt.

Im Ergebnis zeigt dieses Betriebssystem, daß eine Verteilung verschiedener Anwendungen auf die verfügbaren FPGA-Bausteine möglich ist. Dargestellt wird die Funktionalität des Betriebssystems anhand von Anwendungen, die zwar von ihrer Abstraktion vergleichbar mit Funktio-

nen sind, aber dennoch nur einen sehr geringen Datentransfer erfordern. Das Betriebssystem selbst arbeitet nicht preemptiv und muß mit der Verarbeitung der ausführungsbereiten Anwendungen warten, bis die gerade auszuführende Anwendung beendet wird. Gerade durch diesen nicht preemptiven Ansatz können blockierende Situationen entstehen, die die Ausführungszeiten der nachfolgenden Anwendungen erhöhen und den Beschleunigungsfaktor verringern. Eine weitere Einschränkung ist gegeben, sobald mehrere Anwendungen auf jeweils einem FPGA-Baustein verarbeitet werden und jede dieser Anwendungen größere Datenmengen verarbeiten. In dieser Situation bildet die Schnittstelle zwischen dem Host- und dem FPGA-Koprozessor mit der maximalen Datentransferrate eine Engstelle, die ebenfalls die gesamte Verarbeitungszeit aller Anwendungen erhöht.

3.2.3 Multitasking Betriebssystem

Das Ziel dieses von G. Haug entwickelten Systems [HR98b] ist es den FPGA-Koprozessor fest mit der Ausführung der Anwendung auf den Host-Prozessor zu synchronisieren. Eine solche feste Beziehung der Anwendungsausführung auf beiden Prozessoren ermöglicht es den bestehenden Befehlssatz des Host-Prozessors mit speziellen Maschinenbefehlen, die auf dem FPGA-Koprozessor ausgeführt werden, dynamisch zu erweitern.

Der hier zum Einsatz kommende FPGA-Koprozessor basierte auch auf dem XC6216 Baustein von Xilinx, wobei nur die schnelle Konfigurationsschnittstelle eingesetzt wird. Daher steht jedem Prozess die gesamte Logik des Bausteins zur Verfügung. Eine Verwendung weiterer, über die Grenzen des FPGA-Bausteins hinausgehende Elemente wie z.B. lokales RAM ist nicht vorgesehen. Aufgrund dieser Einschränkung ist auch hier nur eine geringe Datenkommunikation, basierend auf den Austausch von Registerwerten, zu erwarten.

Die Prozesswechsel werden über eine Erweiterung des Linux Betriebssystems mit den Prozesswechseln des Host-Prozessors synchronisiert, d.h. mit jedem Prozesswechsel auf dem Host wird auch die Konfiguration auf dem FPGA-Koprozessor gewechselt. Diese enge Verknüpfung

ergibt sich aus der notwendigen Verfügbarkeit der Maschinenbefehle für die ausgeführte Anwendung auf dem Host-Prozessor. Eine Besonderheit bei diesem Prozesswechsel ist die Speicherung und spätere Wiederherstellung des internen Zustandes durch direkte Registerzugriffe auf den FPGA-Baustein.

Durch den Wechsel der FPGA-Funktionalität ist es möglich, die Maschinenbefehle eines Host-Prozessors dynamisch und anwendungsabhängig zu erweitern. Verbunden damit ist aber ein hoher systematischer Overhead, der durch den ständigen Wechsel der FPGA Konfiguration kommt.

Bedingt durch die zum Einsatz kommende, registerbasierte Kommunikation zwischen den beiden beteiligten Prozessoren ist dieses Betriebssystem nur zur Ausührung von Maschinenbefehlen einsetzbar. Die fehlende Möglichkeit größere Datenmengen effektiv übertragen zu können verhindert auch hier die Verwendung des Betriebssystems durch die weitaus größere Anzahl von Anwendungen die große Datenpakete, wie z.B. Bilddaten, verarbeiten.

Im Ergebnis hat dieses Multitasking Betriebssystem die Möglichkeit aufgezeigt einen, auf dem FPGA-Koprozessor laufenden Prozess in dessen Ausführung anzuhalten und ihn zu einem späteren Zeitpunkt wieder fortzusetzen. Dies wird auch hier gezeigt auf dem schon beschriebenen FPGA-Baustein XC6216 der über eine einzigartige und für diesen Einsatz konzipierte Konfigurationsschnittstelle verfügt. Das Betriebssystem beschränkt sich auf eine Klasse von Anwendungen die aufgrund der schlechten Datenrate bei einzelnen Registerzugriffen nur bedingt für eine Auslagerung auf den FPGA-Koprozessor geeignet sind. Operationen, die auf einzelnen Registern ausgeführt werden, können mit der vollen Rechenleistung moderner Prozessoren verarbeitet werden[2] und sind nur bei sehr komplexen Maschinenbefehlen zur Auslagerung geeignet.

3.2.4 Bewertung

Über die in den vorherigen Abschnitten beschriebenen Multi-Context

[2] Ein mit 1GHz laufender Prozessor verarbeitet ca. 700 Maschinenbefehle in der Zeit einer Write-Read Kombination bei den gegebenen Datenraten (lesend 6MByte/s und schreibend 25MByte/s).

FPGABausteine und Betriebssysteme für FPGA-Koprozessor kann zusammenfassend gesagt werden, daß sie im Prinzip alle die Funktionalität zur Verarbeitung mehrerer unabhängiger Anwendungen bereitstellen, aber dennoch aufgrund der individuellen Einschränkungen entweder nur für eine Klasse von Anwendungen oder für spezielle FPGA-Koprozessoren geeignet sind.

Die beiden vorgestellten Multi-Context FPGA-Bausteine sind durch das Wechseln der Konfiguration in wenigen Nanosekunden sehr gut für einen Einsatz in einem preemptiven Multitasking System geeignet, aber trotzdem erweisen sie sich für den praktischen Einsatz als unbrauchbar. Die Gründe hierfür sind:

- Geringe Anzahl von Logikressourcen
- Mangelnde Verfügbarkeit von Entwurfsmethoden
- Schlechte Place&Route-Werkzeug Unterstützung
- Nichtverfügbarkeit der Bausteine

Alle drei beschriebenen Betriebssysteme zeigen, daß eine Verarbeitung von mehreren unabhängigen Anwendungen aus der Klasse der Maschinenbefehlsebene durchgeführt werden kann. Die Anwendungen werden dabei parallel bzw. zeitlich unterteilt auf den FPGA-Koprozessoren ausgeführt. Dennoch ist auch der praktische Einsatz der Betriebssysteme aufgrund folgender Einschränkungen nicht gewährleistet:

- Geringe Unterstützung bei der Entwurfserstellung
- Keine Nutzung von externen Bausteinen wie z.B RAMs
- Geringes Abstraktionsniveau der Kommunikationsschnittstelle (Registerzugriffe)
- Verwendung eines speziellen FPGA-Bausteins mit einzigartiger Konfigurationsschnittstelle
- Geringe bzw. nicht vorhandene Unterstützung zur Verarbeitung von Funktionen und ganzen Anwendungen auf den FPGA-Koprozessoren.

Mit Ausnahme des Ansatzes von G. Haug wird bei den vorgestellten Betriebssystemen ausschließlich die sogenannte 'Overlay'-Technik eingesatzt. Diese in Abbildung 3.3 illustrierte Technik überschreibt die Konfi-

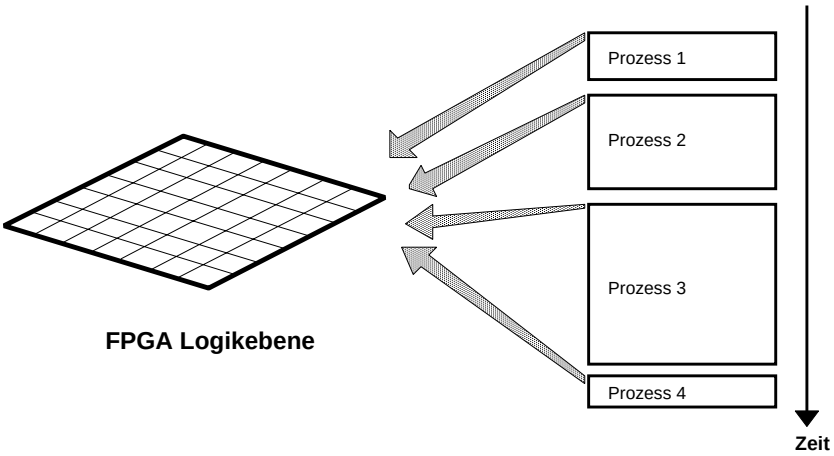


Abbildung 3.3: Funktionsweise der 'Overlay'-Technik bei einem Prozesswechsel

guration des verwendeten FPGA-Bausteins bzw. der SLU mit den neuen Konfigurationsdaten, ohne die Zustände der vorherigen Anwendung zu sichern. Ein solches Vorgehen unterbindet jedoch die Möglichkeit, eine laufende Anwendung in dessen Ausführung anzuhalten, um diesen zu einem späteren Zeitpunkt in dem gleichen Zustand, wie der bei der Unterbrechung bestand, fortzusetzen. Bei entsprechenden daten- und rechenintensiven Anwendungen[3] entstehen durch die 'Overlay'-Technik blockierende Situationen, die die Ausführungszeiten aller nachfolgenden Anwendungen verlängert. Zu sehen ist das anhand der beiden Prozesse 3 und 4 in der Abbildung 3.3. Durch die lange Ausführungszeit von Pro-

[3]Solche Anwendungen verwenden in der Regel eine Abstraktionsebene die auf Funktionen oder ganzen Programmen basieren.

zess 3 kann sich die Ausführungszeit von Prozess 4 um ein mehrfaches der eigentlichen Prozesszeit verlängern und somit die Beschleunigung gegenüber der Ausführung auf dem Host-Prozessor ins Negative umkehren.

3.3 Preemptiver Multitasking Ansatz

Die im folgenden beschriebene Arbeit beseitigt den negativen Einfluß der 'Overlay'-Technik durch den Aufbau eines Betriebssystems, das auf der Basis des preemptiven Multitaskings arbeitet und alle Klassen von Anwendungen verarbeiten kann.

Preemptive Betriebssysteme wie z.B. WindowsNT oder Linux werden heutzutage auf vielen modernen von-Neumann Prozessoren standardmäßig eingesetzt. Sie bieten dem Benutzer viele Vorteile, die zum größten Teil auch auf dem FPGA-Koprozessor übertragbar sind.

Dieser neue Betriebssystemansatz konzentriert sich auf die Anwendungen, die von einer Verarbeitung auf dem FPGA-Koprozessor im besonderen Maße profitieren. Diese sind einer Studie zufolge Anwendungen, die entweder sehr rechenintensiv oder sehr datenintensiv sind oder eine Kombination aus beiden Anforderungen vereinen. Details dieser Untersuchungen sind in Kapitel 4 zu finden. Klassifizieren lassen sich diese untersuchten Anwendungen ausschließlich in die Funktions- und in die Programmebene, die von den bisherigen Betriebssystemen nicht oder nur unzureichend unterstützt werden. Ein weiterer, für die Funktionsweise der analysierten Anwendungen sehr wichtiger Punkt ist die intensive Verwendung von externen Elementen wie z.B. RAMs.

Das in Rahmen dieser Arbeit entstandene Betriebssystem konzentriert sich auf diese beiden Klassen von Anwendungen und deren Verarbeitung auf verfügbaren FPGA-Koprozessoren. Im Mittelpunkt steht die Verwirklichung des preemptiven Multitasking Ansatzes auf Standard FPGA-Bausteinen. Darüberhinaus wird neben dem Betriebssystem selbst auch der weitere für die Praxis wichtige Prozess der Anwendungsentwicklung mit in das Gesamtkonzept aufgenommen.

3.3.1 Statusrekonstruktion einer Schaltung

Zur Vermeidung von blockierenden Zuständen verfolgt das umgesetzte Betriebssystem einen preemptiven Ansatz. Dieser setzt jedoch voraus, daß der Zustand einer Anwendung, die auf dem FPGA-Koprozessor ausgeführt wird, in ihrer Ausführung unterbrochen und zu einem späteren Zeitpunkt wieder hergestellt werden kann.

Die Grundlage, die eine solche Rekonstruktion einer beliebigen Anwendung und damit einer Schaltung ermöglicht, ist der synchrone Aufbau basierend auf dem Register-Transfer Modell (RT-Modell). Dieses Modell besteht aus zwei unterschiedlichen Grundelementen:

Speicherelemente in Form von Registern, die den jeweiligen Zustand über eine Taktperiode stabil halten. Verbunden sind sie jeweils mit Ein- bzw. Ausgängen der Logikelemente. RAMs zählen ebenfalls zu den speichernden Elementen.

Logikelemente, die die kombinatorischen oder arithmetischen Funktionen ausführen. Jedes Logikelement generiert Ausgangssignale, die auf den Eingangssignalen und den internen Funktionen basieren.

Die gesamte Schaltung besteht aus einer Vielzahl dieser beiden Grundelemente die meist alternierend hintereinander geschaltet sind. In Abbildung 3.4 ist der typische Aufbau eines solchen RT-Modells zu sehen. Die Logikelemente sind passive Elemente, d.h. sie generieren kombinatorisch Ausgangssignale, die nur auf den auszuführenden arithmetischen oder logischen Funktionen und auf den Zuständen der Eingänge beruhen. Die Register demgegenüber speichern den jeweiligen Zustand aktiv ab und bestimmen indirekt das Ausgangssignal des nachfolgenden Logikelements.

Somit ist es zur Rekonstruktion einer Schaltung ausreichend, alle Registerzustände zu sichern und diese bei der erneuten Ausführung wieder herzustellen. Dieser Vorgang ist in Abbildung 3.4 graphisch dargestellt. Zur Sicherung der Zustände wird der Taktgenerator abgeschaltet, so daß sich die Registerinhalte stabil verhalten. Diese Taktabschaltung hat zusätzlich den Vorteil, daß weitere speichernde Komponenten wie z.B.

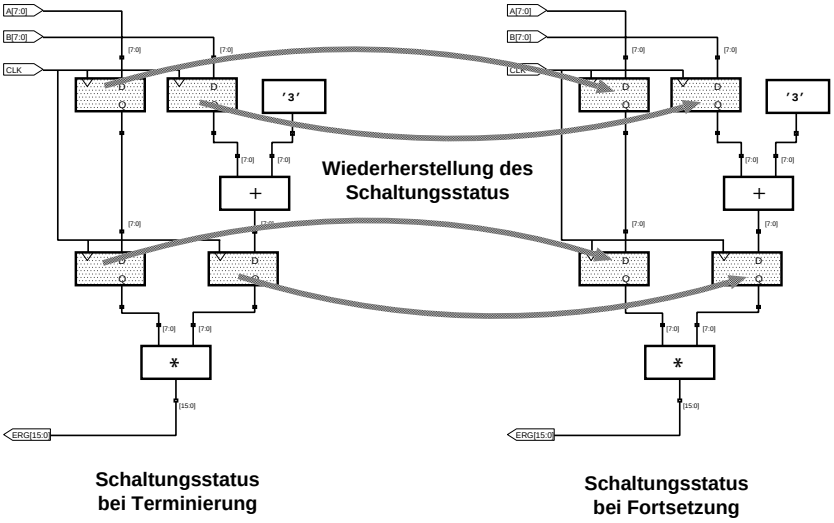


Abbildung 3.4: Darstellung der Statusrekonstruktion bei einer Schaltung. *Das illustrierte RT-Modell setzt folgende Funktion um:* $erg = a * (b + 3)$

RAMs, die sich auch ausserhalb des FPGA-Bausteins befinden können, sich nicht verändern und mitgesichert werden können.

Aufgrund des bei allen Entwicklungssprachen zugrundeliegenden RT-Modells ist es *nicht* notwendig für die Rekonstruktion des Schaltungszustands besondere Unterscheidungen zu treffen.

3.3.2 Zielsetzung des neuen Betriebssystemansatz

Die Zielsetzungen die dieser Arbeit zugrundeliegen, ergeben sich aus den zahlreichen Einschränkungen der bestehenden Betriebssysteme und einer Untersuchung verschiedener Anwendungen für FPGA-Koprozessoren.

> *Ziel ist es, basierend auf der Grundlage der Rekonstruktion von Schaltungen und der Rekonfiguration von FPGA-Bausteinen,*

ein Betriebssystem aufzubauen, das ein preemptives Multitasking mehrerer unabhängiger Prozesse auf dem FPGA-Koprozessor ermöglicht. Dieses Betriebssystem sollte alle Anwendungen ausführen können und auf verfügbaren FPGA-Koprozessoren basieren.

Randbedingung für dieses Ziel ist zum einen die Baustein- bzw. FPGA-Koprozessor Auswahl und zum anderen die Wahl der Betriebssystemarchitektur. Das neue Betriebssystem basiert auf der Verwendung von standardisierten FPGA-Bausteinen. Bauteilspezifische oder einzigartige Merkmale, wie z.B. die 32Bit Konfigurationsschnittstelle des XC6216 von Xilinx, werden zugunsten einer allgemeinen Verwendbarkeit nicht eingesetzt. Ein weiterer Vorteil, der sich durch die Verwendung von Standardbausteinen ergibt, ist die Verfügbarkeit optimierter Entwicklungswerkzeuge, die für den Entwicklungsprozess unerläßlich sind. Dies gilt insbesondere für die Place&Route-Werkzeuge.

Das Ziel beinhaltet auch den erstmaligen Aufbau einer einheitlichen Schnittstelle zur Ansteuerung von FPGA-Koprozessoren. Möglich wird dies durch den allgemeinen Ansatz, der die Unterstützung verschiedener FPGA-Koprozessoren gewährleistet und durch eine Abstraktion der Zugriffe durch das Betriebssystem. Ein, über das Betriebssystem hinausgehende, sekundäres Ziel ist die Erstellung einer Entwicklungsumgebung für Anwendungen. Diese Umgebung ist speziell auf die Gegebenheiten bei der Anwendungsausführung zugeschnitten und ermöglicht eine schnelle und einfache Anwendungsentwicklung. Als selbstverständliches Ziel dieses allgemeinen preemptiven FPGA-Betriebssystems gilt es, den durch das Betriebssystem entstehende Overhead so gering wie möglich zu halten. Ein Einsatz des Betriebssystems ist nur zu rechtfertigen, wenn die Beeinflussung der Ausführungszeit gering gegenüber der eigentlichen Prozesszeit ist.

3.4 Anforderungen

Zur Erfüllung der oben genannten Ziele müssen an alle daran beteiligten Komponenten und an das Betriebssystem unterschiedliche Anforderun-

gen gestellt werden. Im speziellen werden im folgenden alle Randbedingungen und Anforderungen für die Komponenten genannt:

- FPGA-Baustein
- FPGA-Koprozessor
- Betriebssystem und
- FPGA-Schaltung

Eine detailierte Beschreibung aller Anforderungen und Randbedingungen der einzelnen genannten Komponenten sind in den folgenden Abschnitten zusammengefaßt.

3.4.1 FPGA-Baustein Anforderungen

Als wichtigster, ausführender Baustein muß der FPGA-Baustein bestimmten Anforderungen genügen. Aufgrund des preemptiven Betriebssystemansatzes und der damit verbundenen hohen Rekonfigurationsrate des Bausteins steht die Konfigurationsschnittstelle des FPGA-Bausteins in Zentrum der Anforderungen.

Diese Konfigurationsschnittstelle hat direkten Einfluß auf den Overhead des gesamten Betriebssystems und unterliegt daher mehreren Anforderungen. Weitere Anforderungen an den Konfigurationsbitstrom entscheiden über die Effektivität bei der, für den Ansatz notwendigen, Rekonstruktion der Schaltungen. Desweiteren sind aber auch gewisse Anforderungen an die praktische Verwendbarkeit der Bausteine gefordert, die einen Einsatz überhaupt erst ermöglichen.

Alle an den FPGA-Baustein gestellte Anforderungen sind in den nachfolgenden Abschnitten näher erläutert und anhand von möglichen Alternativen begründet.

3.4.1.1 Konfigurationsschnittstelle

Über die Konfigurationsschnittstelle eines FPGA-Bausteins wird der Bitstrom geladen. Dies wird auch Konfigurieren oder Rekonfigurieren genannt. Da bei jedem Prozesswechsel auf dem FPGA-Baustein dieser Bit-

strom rekonfiguriert wird, stellt die dafür benötigte Zeit einen proportionalen Anteil an dem systembedingten Overhead dar. Somit verringert sich mit kleiner werdenden Konfigurationszeiten auch der systembedingte Overhead des Betriebssystems.

Viele FPGA-Bausteine bieten dem Benutzer bis zu drei Schnittstellen zum Konfigurieren des Bausteins an. Jede dieser Schnittstelle ist für ein bestimmtes Einsatzgebiet konzipiert und verfügt über entsprechende Vor- und Nachteile. Nachfolgend werden die drei verfügbaren Schnittstellen kurz vorgestellt und unter dem Gesichtspunkt der Geschwindigkeit beurteilt.

JTAG: Die JTAG[AK96] Schnittstelle wurde ursprünglich für den Systemtest eingebauter Bausteine entwickelt. Für FPGA-Bausteine wurde die Funktionalität dieser einfachen, schon etablierten Schnittstelle erweitert, um die FPGA-Bausteine darüber konfigurieren zu können.

Bei der bitseriellen Schnittstelle, die ein Hintereinanderschalten mehrerer Bausteine ermöglicht, wird jedes Konfigurationsbit einzeln geladen und benötigt daher relativ lange für die gesamte Konfiguration.

Bitserielle Schnittstelle: Sie ist ausschließlich zur Konfiguration des FPGA-Bausteins zuständig und kann den Konfigurationsbitstrom sowohl passiv nur empfangen als auch aktiv von einem weiteren Baustein auslesen. Die aktive Beschaltung findet bei Geräten statt, die während der Power-Up Phase den Bitstrom für einen FPGA-Baustein aus einem ROM-Baustein automatisch auslesen. Sie werden in der Regel nur einmal konfiguriert. Der Betrieb mit einer passiven Schnittstelle erfordert einen Host-Prozessor zur Konfiguration des Bitstroms.

Die Konfiguration erfolgt auch hier bitseriell über eine 1Bit breite Schnittstelle, wobei jeder Baustein separat angesprochen werden muß. Im Vergleich zu der JTAG-Schnittstelle ist die Konfiguration über diese bitserielle Konfigurationsschnittstelle nur aufgrund der höheren Frequenz schneller als die JTAG Schnittstelle.

Parallel Schnittstelle: Diese Schnittstelle wird ebenfalls zur Konfiguration des FPGA-Bausteins verwendet. Sie ist durch das parallele Anlegen der Konfigurationdaten entsprechend schneller als die beiden vorher genannten Schnittstellen. Sie wird nur passiv betrieben und erlaubt im Einzelfall auch eine selektive Adressierung der zu konfigurierenden Bereiche (partielles Rekonfigurieren).

Die Schnittstellenbreite beträgt meist 8Bit und ist an mehreren Bausteinen verfügbar. Im Vergleich zu der bitseriellen und der JTAG Schnittstelle wird eine 8fach niedrigere Konfigurationszeit erreicht.

Zum Vergleich sind in der nachfolgenden Tabelle 3.1 die theoretischen Konfigurationszeiten für eine Xilinx Virtex XV400 Baustein dargestellt.

SCHNITTSTELLE	JTAG	BITSERIELL	PARALLEL
XV400	77,2ms	50,9ms	6,4ms

Tabelle 3.1: Konfigurationszeiten für einen Xilinx Virtex XV400 Baustein unter Verwendung verschiedener Konfigurationsschnittstellen. Der Bitstrom des XV400 umfaßt ≈2.4 MBit.

Aufgrund der proportionalen Auswirkungen und der später noch näher erläuterten dominierenden Stellung der Konfigurationszeit im Vergleich zu der gesamten Prozesswechelzeit bietet nur die parallele Konfigurationsschnittstelle eine akzeptable Lösung um einen geringen Overhead zu erreichen. Daher muß sie von dem verwendeten FPGA-Baustein unterstützt werden. Aufgrund der schnell wachsenden Bausteingrößen (Logikfläche aber auch I/O Portanzahl) wird diese 8Bit parallel oder noch breitere Schnittstellen in naher Zukunft allgemein verfügbar sein.

3.4.1.2 Readbackmöglichkeit

Eine weitere, für die Umsetzung der Zustandsrekonstruktion wichtige Anforderung an die FPGA-Bausteine ist die Ermittlung der internen

Zustände aller speichernden Elemente wie Register und RAMs. Die Umsetzung der Zustandsermittlung kann auf zwei prinzipiell unterschiedliche Arten durchgeführt werden.

Einen Möglichkeit ist die Einführung einer **Kette**, angelehnt an das JTAG Testverfahren, die parallel zur Schaltung angelegt wird und alle verwendeten Register hintereinander schaltet. Diese Kette wird bei einem Prozesswechsel entsprechend zurückgelesen, um die Zustände aller internen Register zu erhalten. Über die gleiche Kette erfolgt auch die Wiederherstellung des Schaltungszustandes bei einer erneuten Ausführung der Anwendung. Obwohl diese Art des Readbacks nur die wirklich verwendeten Register sichert, hat sie entscheidende Nachteile. Zum einen bedeutet diese Lösung einen nicht unerheblichen zusätzlichen Aufwand an Logic (Multiplexer) und Netzwerkverbindungen und zum anderen sind RAM-Elemente von einer solchen Zustandssicherung ausgeschlossen.

Die andere, von fast allen FPGA-Bausteinen unterstützte Methode, ist die des **Readbacks**. Readback bedeutet in diesem Zusammenhang das Zurücklesen des gesamten Inhalts des internen Konfigurationsspeichers. Dieser Readbackbitstrom, der fast ausschließlich zur Kontrolle des Konfigurationsbitstroms eingesetzt wird, enthält zusätzlich den Zustand aller speichernden Elemente inklusive der RAMs. Der Readback erfolgt wie die Konfiguration über die ausgewählte Konfigurationsschnittstelle.

Obgleich bei einem ausgeführten Readback Kommando alle Konfigurationsbits zurückgelesen werden, stellt der Readback die einzige praktikable Möglichkeit dar, um den internen Zustand aller speichernden Elemente zu ermitteln. Darüberhinaus bedeutet die erste vorgestellte Möglichkeit der Zustandsermittlung einen erheblichen Mehraufwand bei der Anwendungsentwicklung. Aus diesen beiden Gründen muß ein verwendeter FPGA-Baustein neben einer schnellen Konfigurationsschnittstelle auch die Möglichkeit eines schnellen Readback zulassen.

3.4.1.3 Zustandsrekonstruktion

Neben der Möglichkeit, den aktuellen Zustand einer Schaltung auszulesen ist es genauso notwendig, diesen Zustand bei einer erneuten Ausführung wieder herzustellen. Zu diesem Zweck muß das Betriebssystem in der

Lage sein, jedes Register und jeden RAM-Block in den Zustand setzen zu können, in dem es bei der Ausführungsunterbrechung war.

Die Register einer Logikzelle verfügen hierzu über einen sogenannten Initialisierungs-Eingang. Bei Anlegen des aktiven Signals an diesem Eingang wechselt das Register in den zuvor definierten Anfangszustand der entweder einem *HIGH* oder einem *LOW* Signal entspricht. Dargestellt ist dies noch einmal in Abbildung 3.5 anhand der Logikzelle eines Virtex Bausteins. Aufgrund dieser standardmäßig eingesetzten Methode

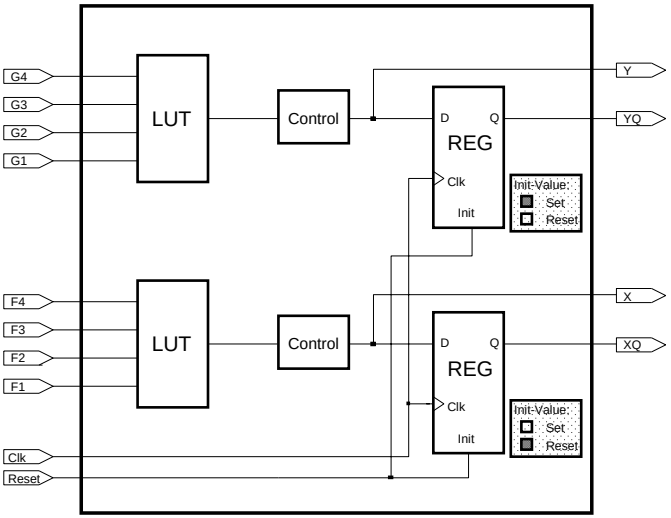


Abbildung 3.5: Statusrekonstruktion bei Register eines Virtex FPGAs. *Das obere Register ist auf HIGH und das untere auf LOW gesetzt.*

bei allen FPGA-Bausteinen ist diese Anforderung immer erfüllt.

Neben den Registern verfügen auch die FPGA internen RAM-Blöcke über eine Initialisierung, die im Unterschied zu den Registern immer mit ganzen Datenwörtern initialisiert werden. Die Datenspeicherung und Initialisierung dieser RAM-Blöcke erfolgt aufgrund der anderen Organisation direkt innerhalb des Konfigurationsspeichers . Dies bedeutet, daß

sowohl die Initialisierung der Register als auch der RAM-Blöcke standardmäßig durchgeführt werden können.

Die Festlegung des Anfangszustands erfolgt normalerweise während der Entwurfsphase. Zusammen mit der Beschreibung oder in speziellen Anforderungslisten werden die einzelnen Anfangszustände für Register und RAMs festgelegt, vom Place&Route-Werkzeug umgesetzt und in den Konfigurationsbitstrom übernommen. Siehe dazu auch Abbildung 2.5 auf Seite 23.

Das preemptive Multitasking Betriebssystem verändert jedoch die Anfangszustände mit jedem Prozesswechsel. Eine solche dynamische Veränderung des Anfangszustandes auf der Ebene der Entwurfssprache ist aufgrund der langen Laufzeiten des Place&Route-Werkzeugs (min. 1 Minute) nicht realisierbar. Vielmehr zählt die Rekonstruktion des Zustandes mit zu der Zeit, die für einen Prozesswechsel benötigt wird und muß daher so gering wir nur möglich sein. Diese zeitliche Anforderung ist aber nur durch eine direkte Manipulation des Konfigurations- und des Readbackbitstroms einzuhalten. Eine solche direkte Bitstrommanipulation erfordert weiterhin die Offenlegung von Teilen der Bitstromarchitektur, die im nächsten Anschnitt beschrieben wird.

3.4.1.4 Bitstromarchitektur

Die effektive Rekonstruktion des internen Zustandes einer FPGA-Schaltung kann nur durch eine direkte Manipulation des Konfigurationsbitstroms erreicht werden. Zur Durchführung einer solchen Manipulation muß sowohl die Bitstromarchitektur als auch die zugrundeliegende Kodierung bekannt sein.

Die Bitstromarchitektur definiert dabei die Funktionen, die von den einzelnen Bits bestimmt werden. Am Beispiel des Anfangszustands der Register (Abbildung 3.5) wird dieser Zusammenhang deutlich. Ein Bit bestimmt den Ausgangszustand des Registers nach der Konfiguration und ein weiteres Bit, das an einer anderen Position im Bitstrom liegt, beinhaltet den aktuellen Zustand des Registers. Ähnliche Zusammenhänge, wie sie in der nachfolgenden Abbildung 3.6 illustriert sind, gelten sowohl für den Konfigurations- als auch für den Readbackbitstrom.

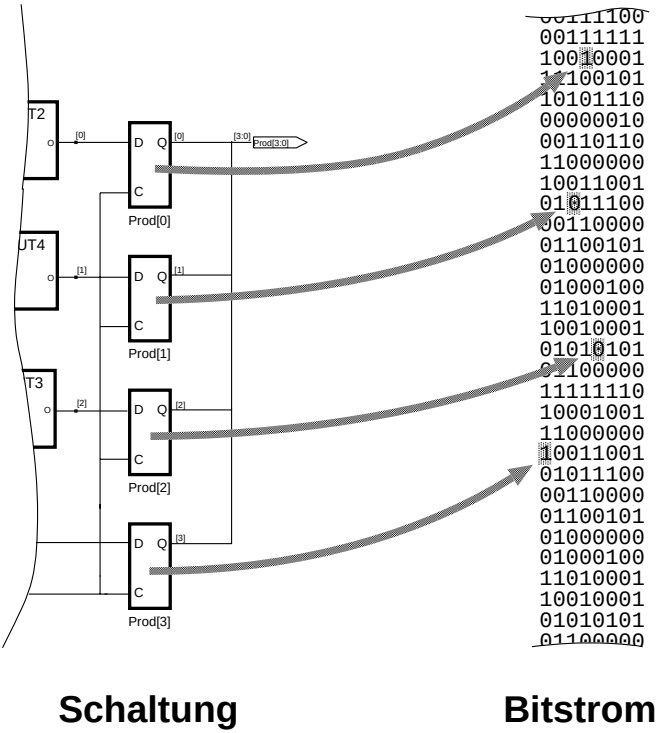


Abbildung 3.6: Zusammenhang der Anfangszustände und des Bitstroms. *Dieses illustrierte Beispiel zeigt den Zusammenhang zwischen der Netzliste und dem Konfigurationsbitstrom.*

Für die Rekonstruktion des Zustandes sind nur die Register und RAM-Inhalte interessant die im Falle des Xilinx Virtex XV400 Bausteins ca. 10% des gesamten Konfigurations- bzw. Readbackbitstroms ausmachen.

Die Kodierung der einzelnen Bitpositionen definiert die Funktion bzw. den jeweiligen Zustand des angesprochenen Elements. Z.B. kodiert eine '1' an der Stelle des Anfangszustands im Konfigurationsbitstrom ein Logikzellen-Register mit einem *HIGH*, wohingegen ein Ausgangs-Register einer I/O-Zelle zum Erreichen des gleichen Anfangszustands mit '0' kodiert werden muß.

Die direkte Bitstrommanipulation ist notwendig um bei der Umsetzung eines Betriebssystems einen geringen systematischen Overhead zu gewährleisten. Um die direkte Zustandsrekonstruktion zu ermöglichen müssen sowohl die Kodierung als auch die Architektur des Konfigurations- und des Readbackbitstroms in Teilen bekannt sein.

3.4.1.5 Konfigurationszustand

Der Ansatz des Betriebssystems umfasst neben dem FPGA-Baustein auch weitere externe Elemente wie z.B. lokale RAMs die mit in die Anwendung integriert sind. Um die Datenintegrität zu gewährleisten muß der Zustand der Signale die direkt mit den FPGA-Baustein verbunden sind während des Konfigurationsvorgangs weiteren Andorderungen genügen. Am Beispiel einer angeschlossenen Datenquelle wird dies deutlich:

Die Daten werden von der Datenquelle mit einem einfachen Steuersignal angefordert. Generiert der FPGA-Baustein während der Konfiguration ein aktives Signal an den Ausgängen, so ist dies für die externe Datenquelle gleichbedeutend mit dem Auslesen von Daten. Diese, während des Konfigurationsvorgangs übertragenen Daten sind verloren, da die FPGA-Schaltung die Daten nicht entgegennehmen kann.

Diese eher unscheinbare Anforderung ist sowohl für die Auswahl des FPGA-Bausteins aber auch für die Entwicklung von Datenquellen oder das Verbinden von RAMs wichtig.

3.4.1.6 Software-Unterstützung

Diese praktisch orientierte Anforderung soll gewährleisten, daß der ausgewählte FPGA-Baustein, der den anderen Anforderungen genügt, auch später für die Anwendungen eingesetzt werden kann.

Bei der Entwicklung einer Anwendung für einen FPGA-Baustein spielen die Entwicklungswerkzeuge eine entscheidende Rolle. Deren Qualität ist mitbestimmend über die Leistungsfähigkeit der endgültigen Schaltung die auf dem FPGA-Baustein ausgeführt wird. Im speziellen ist es das Place&Route-Werkzeug, das über die Verwendbarkeit der Bausteine mitentscheidet. Ein Beispiel, das diesen wichtigen Zusammenhang aufzeigt, ist der Xilinx Baustein XC6216. Er ist hervorragend für ein Multitasking Betriebssystem geeignet, aber aufgrund der fehlenden Unterstützung des Place&Route-Werkzeugs ist das Einsatzgebiet dieses Bausteins nur auf sehr kleine und regelmäßige Schaltungen beschränkt.

Aus Gründen der einfachen und unkomplizierten Programmierbarkeit sollte daher bei der Auswahl des eingesetzten FPGA-Bausteines auch die jeweilige Unterstützung durch die Software-Werkzeuge als Anforderung herangezogen werden. Ein weiterer Punkt bei der Bausteinauswahl ist die verfügbare Anzahl an Logikzellen. Zur Entwicklung von komplexeren Anwendungen sollten Bausteine mit einer großen Anzahl von Logikressourcen verwendet werden.

3.4.2 FPGA-Koprozessor Anforderungen

Neben den zahlreichen Anforderungen, die an den zur Verwendung kommenden FPGA-Baustein gestellt werden, bestehen weitere Anforderungen die den FPGA-Koprozessor betreffen. Nur FPGA-Koprozessoren die die folgenden Anforderungen erfüllen sind geeignet für ein preemptives Multitasking Betriebssystem.

Diese, sich teilweise mit dem FPGA-Baustein überschneidenden Anforderungen betreffen folgende Bereiche: Die Anbindung der Konfigurationsschnittstelle, die Takterzeugung auf dem FPGA-Koprozessor, die lokalen RAM-Bänke und den Zugriff auf diese RAMs vom Host-Prozessor aus. Diese einzelnen Bereiche, aus denen sich die Anforderungen ergeben, sind nachfolgend näher beschrieben.

3.4.2.1 Konfigurationsschnittstelle

Wie in dem Abschnitt 3.4.1.1 ausführlich dargestellt ist, beeinflussen die Zeiten die für die Konfiguration und den Readback des FPGA-Bausteins benötigt werden maßgeblich den systembedingten Overhead des Betriebssystems.

Zur Minimierung dieser Zeiten ist bei der Auswahl des FPGA-Koprozessors darauf zu achten, daß die entsprechend schnellste Konfigurationsschnittstelle für die Konfiguration und den Readback genutzt wird. Diese Schnittstelle muß auf dem FPGA-Koprozessor optimal umgesetzt werden, so daß das Betriebssystem auf dem Host-Prozessor einen schnellen Prozesswechsel auf dem FPGA-Koprozessor durchführen kann.

3.4.2.2 Takterzeugung

Bei der Durchführung eines Prozesswechsels wird zuerst der Zustand der gesamten ausscheidenden Anwendung durch einen Readback gesichert. Dieser Readbackvorgang benötigt bei einem modernen FPGA-Baustein in der Regel etwas länger als der eigentliche Konfigurationsvorgang und beträgt für den in der Tabelle 3.1 angegebenen Baustein XV400 ≈8ms. Bedingt durch die schnelle Schaltungstaktung im MHz Bereich werden so zwischen dem Auslesen des ersten und des letzten Registerwertes mehrere tausend Takte verarbeitet. Der in einem solchen Fall entstehende Status der Schaltung kann technisch gesehen zwar rekonstruiert werden, aber dennoch werden bedingt durch die fehlende Synchronität die einzelnen fein aufeinander angestimmten Anwendungsabläufe in zeitlich unterschiedlichen Ausführungszuständen gesichert. Die fehlende Synchronität im Anwendungsablauf, die auch nachträglich nicht wiederhergestellt werden kann, macht eine Rekonstruktion des Zustands unmöglich.

Eine Rekonstruktion des Zustands kann nur durchgeführt werden, solange sich die Schaltung während des gesamten Readbackvorgangs nicht verändert. Diese Anforderung an den FPGA-Koprozessor kann entweder durch das Einführen von zusätzlicher Logik während des Entwicklungsprozesses oder durch ein Abschalten der Taktgeneratoren verwirklicht werden. Beide Realisierungen ermöglichen ein 'Einfrieren' des Zustands durch die Abschaltung der Taktgenerierung und damit eine synchrone Sicherung des FPGA-Schaltungszustands.

Das Einführen zusätzlicher Logik ist vergleichbar mit der individuellen Readback Kette (siehe Abschnitt 3.4.1.2) und auch mit den gleichen Nachteilen verbunden.

Das Verändern der Taktgenerierung zum 'Einfrieren' einer Schaltung erfolgt hingegen sehr einfach durch die Abschaltung des Takts. Ohne den Takt behalten die Register und RAMs ihren derzeitigen Zustand und bei einem folgenden Readback des Zustands ist die Synchronität der Schaltung erhalten, da alle Register zum gleichen Zeitpunkt angehalten werden. Technisch kann diese Taktabschaltung unter Verwendung eines, in den meisten Fällen vorhandenen, 'Output-Enable'-Signals des zugehörigen Taktgenerators durchgeführt werden, das über die Schaltung in Abbildung 3.7 angesteuert wird. Diese Schaltung gewährleistet, daß selbst bei mehreren unterschiedlich schnellen Takten die Abschaltung immer zum gleichen Zeitpunkt erfolgt, ohne das einzelne Takte 'verloren' gehen.

Bedingt durch diese Abschaltung des Taktes unterliegt die weitere Taktschaltung auf dem FPGA-Koprozessor weiteren Anforderungen. So ist der Einsatz von PLL[4] (Phase Locked Loop) oder DLL (Delay Locked Loop) Schaltungen unmöglich, da diese versuchen einen Takt durch Ausgleichen von Schwankungen möglichst stabil zu halten. Eine Verringerung von z.B. 10MHz auf 0Hz ist jedoch eine sehr massive Störung, die von den PLLs oder DLLs entweder nur nach einer endlichen Zeit oder gar nicht ausgeregelt werden kann. Ein solches Verhalten führt somit zu einer nicht eindeutigen Unterbrechung der Schaltungsausführung und ist daher in jedem Fall zu vermeiden.

Eine weitere Anforderung betrifft die Trennung zwischen der Taktgenerierung für die Schaltung und für die Konfigurationsschnittstelle. Der Takt für die Konfigurationsschnittstelle wird unter anderem während des Readbacks benötigt. Im Gegensatz dazu muß der Takt für die Schaltung während des Readbacks abgeschaltet sein. Diese beiden Anforderungen lassen sich aber nur bei getrennt zu steuernden Taktsignalen erfüllen. Daher ist bei der Auswahl bzw. dem Aufbau eines FPGA-Koprozessor u.a. auch auf eine solche Takttrennung zu achten.

[4]PLL und DLLs sind Schaltungen zur Stabilisierung und Synchronisierung von Taktsignalen

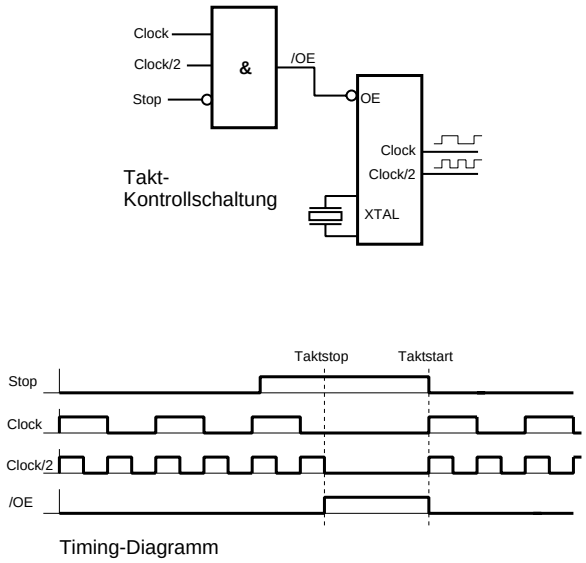


Abbildung 3.7: Schaltung die ein sicheres Abschalten des Taktgenerators ermöglicht. *In der oberen Schaltung wird über ein Stop-Signal die Taktgenerierung immer zum gleichen Zeitpunkt an- bzw. abgeschaltet, um keine Takte zu 'verlieren'. Dies ist in dem unteren Timingdiagramm noch einmal verdeutlicht.*

Ein in der Praxis häufig angewendeter Trick ist die Verwendung von mehreren unterschiedlich schnellen Taktgeneratoren. Dadurch wird die Leistungsfähigkeit einzelner Schaltungsteile gesteigert bzw. die Zugriffsrate auf ein RAM wird vergrößert. Gezeigt wird dies in der Doktorarbeit von Holger Singpiel[Sin00].

Für die Rekonstruktion einer Schaltung ist es jedoch wichtig, den Zustand einer Schaltung so zu unterbrechen, daß der Anwendungsablauf nicht verändert wird. Diese Anforderung ist bei mehreren unterschiedlich schnellen Takten jedoch nur erfüllt, wenn folgende Eigenschaften zutreffen:

Phasensynchronität: Die einzelnen Takte müssen phasensynchron zueinander schalten. Nur diese Synchronität gewährleistet, daß der Ablauf nicht durch ein veränderte Phase gestört wird. Ist die Phasensynchronität nicht gegeben kommt es durch das Anhalten und das Fortführen der Taktgenerierung automatisch zu Phasenverschiebungen, die den Ablauf stören können.

Ganzzahlige Multiplikatoren: Nur die Ableitung der höherfrequenten Takte von einem Basistakt ermöglicht eine Synchronität zueinander. Um zusätzlich eine Synchronität mit jeder Flanke des Basistakts zu erreichen, dürfen die höheren Takte nur mit einem ganzzahligen und geraden Vielfachen des Basistakt arbeiten.

Abschaltung des Basistakts: Die zuvor genannte Ableitung von dem Basistakt ist auch für die Abschaltung und den Neustart wichtig. So kann der Ablauf nur dann gewährleistet werden, wenn die Taktsignale nach dem Neustart den gleichen Signalpegel besitzen wie vor der Abschaltung der Takte. Verdeutlicht wird dies in dem Timingdiagramm in Abbildung 3.8. Der Neustart erfolgt nur wenn alle Taktsignale auf *HIGH* gesetzt sind. Dies hat zur Folge, daß auch das Anhalten des Taktes unmittelbar vor diesem Zustand erfolgen muß, um keinen Takt zu 'verlieren'.

Abbildung 3.8 stellt die drei Anforderungen bei der Anwendung mehrerer Taktsignale noch einmal zusammen.

Die oben genannten Anforderungen bei mehreren unterschiedlichen Takten werden von den meisten FPGA-Koprozessoren erfüllt, da zur Taktgenerierung ein programmierbarer Taktgenerator eingesetzt wird. Erfüllen die unterschiedlichen Takte die genannten Anforderungen nicht, so kann die Schaltung mit nur einem der zur Auswahl stehenden Takte betrieben werden. Die Möglichkeit der Taktabschaltung ist jedoch unbedingt notwenig, um die Schaltung innerhalb eines Taktes anzuhalten. Sollte eine solche Möglichkeit nicht vorhanden sein, so kann dies durch ein nachträgliches Einfügen der in Abbildung 3.7 dargestellten Schaltung, in das vorhandenen Taktsystem erfolgen. Alternativ ist auch eine Taktabschaltung innerhalb der Schaltung realisierbar, solange die Taktverteilung der Schaltung innerhalb des FPGA-Bausteins erfolgt.

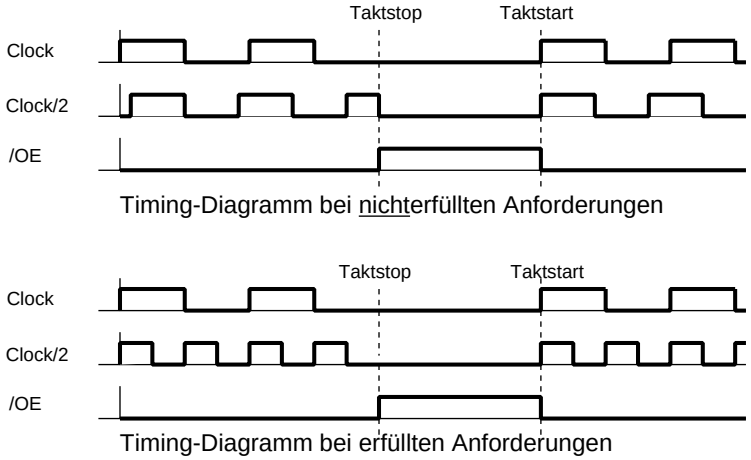


Abbildung 3.8: Darstellung der Anforderungen, die beim Einsatz von mehreren Takten notwendig sind. *Im oberen Timingdiagramm sind zwei Takte dargestellt, die den Anforderungen nicht gerecht werden. Im unten Diagramm sind die Anforderungen erfüllt.*

3.4.2.3 RAM Anbindung

Viele der verfügbaren FPGA-Koprozessoren stellen neben dem FPGA-Baustein noch weitere Elemente für die Nutzung bereit, die von existierenden Anwendungen intensiv genutzt werden. An erster Stelle ist dabei das lokale RAM zu nennen, das unter anderem zum Zwischenspeichern von Daten aber auch als große Look-Up Tabelle eingesetzt wird. Durch diese Integration der verfügbaren RAMs sind auch diese Bausteine ein Teil der Anwendung die bei einem Prozesswechel gesichert werden müssen.

Aufgrund des direkten Zugriffs der FPGA-Schaltung auf diese RAMs sind diese in der Regel fest mit dem FPGA-Baustein verbunden. Daher erfolgt die Sicherung des RAM-Inhalts entweder direkt über die derzeit geladene FPGA-Schaltung oder aber nach der Konfiguration einer speziellen FPGA-Schaltung die den Zugriff ermöglicht.

Bei der Anbindung von SDRAM bedeutet eine Sicherung des gesamten SDRAMs aufgrund der Speichergröße von meist mehr als 32MByte ein zu großer Aufwand.[5] Aus diesem Grund werden SDRAMs entweder exklusiv von einer FPGA-Schaltung verwendet oder der zur Verfügung stehende Speicher wird unter den Schaltungen in einzelne Pages aufgeteilt.

Unabhängig von der Aufteilung unter mehreren FPGA-Schaltungen muß das eingesetzte SDRAM-Modul über eine unabhängige Refresh-Schaltung verfügen. Diese automatische Refresh-Schaltung ist notwendig, da während des Readbacks und der Rekonfiguration die FPGA-Schaltung die Datenworte innerhalb des SDRAMs nicht erneuern kann [TG99]. Darüberhinaus muß ein inaktiver Zustand (mit Ausnahme der Aktivierung der Refresh-Schaltung) durch die anliegenden FPGA-Signale gewährleistet sein, um den Inhalt des SDRAMs während der Prozesswechselzeit nicht zu verändern.

3.4.2.4 Datenübertragung

Die gesamte Leistungsfähigkeit der Anwendungen und die schnellen Prozesswechsel des Betriebssystems unterliegen vor allem der Datentransferrate, die zwischen dem Host- und dem FPGA-Koprozessor erreicht werden kann. Diese beeinflußt neben der Konfiguration, dem Readback und der Sicherung des SRAMs auch den Datenaustausch zwischen der Anwendung und dem Host-Prozessor. Aus praktischen Gründen ist daher bei der Auswahl des FPGA-Koprozessor auf eine effektive Datentransferrate zu achten.

3.4.3 FPGA-Schaltungsanforderungen

Die oben genannten Anforderungen an den FPGA-Baustein und den FPGA-Koprozessor sind für die Auswahl der infragekommenden FPGA-Koprozessor erforderlich. Darüberhinaus bestehen weitere Anforderungen, die bei der Entwicklung der Anwendungen bzw. der FPGA-Schaltungen wichtig sind.

[5] Die Sicherung und Rekonstruktion von 32MByte Speicher benötigt bei einer Datenübertragungsrate von 120MByte/s ≈530ms.

Diese Schaltungsanforderungen sind speziell für den Multitaskingbetrieb der Anwendungen notwendig, um die Datenintegrität zu gewährleisten. Sie beschränken sich auf den Aufbau von speichernden Elementen und die Integration von externen Elementen wie z.B. RAMs. Darüberhinaus werden hier vor allem auch praktische Anforderungen beschrieben, die eine schnellere und einfachere Entwicklung von Anwendungen ermöglichen.

3.4.3.1 Aufbau speichernder Elemente

Wie in Abschnitt 3.4.1.3 beschrieben ist die Rekonstruktion eines Registerinhaltes nur möglich, wenn der Anfangszustand des Registers durch ein Bit im Bitstrom definiert werden kann.

Der Aufbau eines Registers aus rückgekoppelten Logikbausteinen wie NAND oder NOR – also aus kombinatorischer Logik – ist für den Multitaskingbetrieb nicht erlaubt. Dieser Aufbau von speichernden Elementen kann innerhalb des Konfigurationsbitstroms nicht wiederhergestellt werden, da der kombinatorischen Logik kein Anfangszustand zugewiesen ist.

Das Auslesen und die Rekonstruktion der Zustände ist somit auf Register bechränkt. Um eine Datenintegrität zu garantieren müssen alle speichernden Elemente einer FPGA-Schaltung entweder als Register oder als RAMs ausgeführt werden.

3.4.3.2 Externe Elemente

Eine in der Praxis häufiger vorkommende Anforderung ist verbunden mit der Integration von externen Bausteinen die auf dem FPGA-Koprozessor zur Verfügung stehen. In Nachfolgenden wird die dort entstehende Problematik anhand des Beispiel eines RAM-Bausteins erläutert.

Die zum Einsatz kommenden RAM-Bausteine benötigen für einen lesenden Zugriff meist ein oder zwei Taktzyklen zum Bereitstellen der Daten. Im oberen Teil der Abbildung 3.9 ist eine solcher lesender RAM-Zugriff mit einem Takt Wartezeit illustriert. Erfolgt der Prozesswechsel direkt nach der Adressierungsphase, so werden die Daten, die in der direkt anschließenden Datenphase auf dem Bus bereitstehen, aufgrund des angehaltenen Taktes nicht mehr in die FPGA-Schaltung übernommen.

Bei der erneuten Ausführung der Anwendung startet diese in der zuvor abgebrochenen Datenphase, aber die Daten des RAMs liegen nicht mehr an dem Datenbus an. Eine Unterbrechung zwischen der Adressierungs- und der Datenphase führt somit zu einem falsch gelesenen Datenwort und damit auch zu einem fehlerhaften Endergebnis.

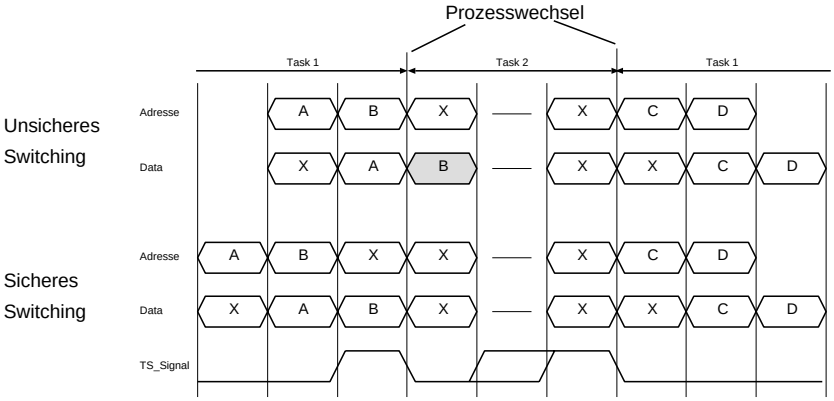


Abbildung 3.9: Timingverhalten bei einem Lesezugriff auf ein externes RAM-Element. *Im oberen Diagramm wird ein unsicherer Prozesswechsel gezeigt, bei dem Datenworte falsch gelesen werden. In unteren Beispiel signalisiert ein spezielles Signal den Zeitpunkt eines sicheren Prozesswechsels.*

Im unteren Beispiel werden alle nicht unterbrechbaren Taktzyklen eines RAM-Zugriffs durch ein spezielles 'Task-Switch' Signal markiert. Dieses Signal signalisiert der Taktabschaltung, daß der Takt angehalten werden kann ohne das angeforderte Daten verloren gehen. Aufgrund der synchronen und ohne Waitstates auskommenden internen Register und RAMs ist dieses Signal nur für externe RAM-Elemente notwenig und daher durch eine einfache Überwachung der zum RAM gehenden Kontrollsignale zu realisieren.

Weitere, in die Anwendung integrierte externe Bausteine, müssen ebenfalls mit ihren Zugriffen überwacht werden und abhängig von dem jeweiligen Zustand muß das 'Task-Switch' Signal generiert werden, um eine sichere Abschaltung des Taktes immer zu gewährleisten.

3.4.3.3 Anwendungsentwicklung

Die letzte, für eine schnelle und einfache Anwendungsentwicklung notwendige Anforderung ist die exklusive Nutzung der auf dem FPGA-Koprozessor verfügbaren Ressourcen für die jeweilige Anwendung.

Bei einer gleichzeitigen Nutzung einzelner Ressourcen, wie z.B. das lokale RAM, durch zwei konkurrierende Prozesse muß der Zugriff auf die Bausteine und die Datenrate entsprechend aufgeteilt werden. Eine solche Aufteilung kann aufgrund entstehender Wartezeiten zu einem großen Leistungsverlust führen und zudem erfordert die Aufteilung einen zusätzlichen Aufwand bei der Entwicklung der Anwendungen.

Demgegenüber stellt die exklusive Nutzung des gesamten FPGA-Koprozessors eine erhebliche Vereinfachung bei der Anwendungsentwicklung dar, die durch Ausnutzung der gesamten zur Verfügung stehenden Ressouren ein Maximum an Leistungsfähigkeit erreicht. Speziell die Simulation der entwickelten Anwendungen wird durch diese exklusive Nutzung erheblich vereinfacht und ermöglicht den Aufbau einer Simulationsumgebung die für alle Anwendungen gleich ist. Mit dieser Simulationsumgebung ist die Durchführung aller drei Anwendungssimulationen innerhalb des Entwicklungsprozesses aus nur einer Umgebung heraus möglich. In Abbildung 3.10 ist diese Simulationsumgebung vereinfacht dargestellt.

3.4.4 Betriebssystem Anforderungen

Das Betriebssystem steuert die gesamte Ausführung der unterschiedlichen Anwendungen auf dem FPGA-Koprozessor und unterliegt daher ebenfalls verschiedenen Anforderungen.

Diese Anforderungen beziehen sich weniger auf technische Einzelheiten als vielmehr auf eine Liste von Aufgaben die das Betriebssystem für eine Multitaskingverarbeitung erfüllen muß. Zu den Aufgaben gehören

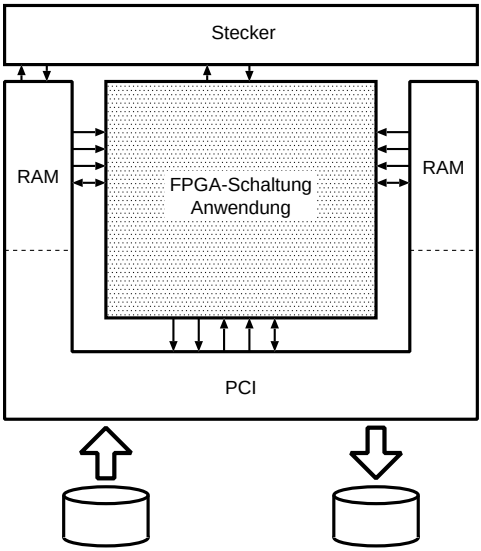


Abbildung 3.10: Dargestellt ist eine Simulationsumgebung die durch eine exklusive Nutzung der FPGA-Koprozessor Ressourcen ermöglicht wird.

die Koordinierung der einzelnen Anwendungen und deren Ausführung auf dem FPGA-Koprozessor, die Bestimmung der Ausführungsreihenfolge, aber auch die Konfiguration und der Readback des FPGA-Bausteins und die Rekonstruktion des internen Zustands. Die nachfolgende Zusammenstellung gibt einen detailieren Überblick über die einzelnen Aufgaben des Betriebssystems.

3.4.4.1 Konfiguration und Readback

Diese beiden Funktionen, die eine preemptive Ausführung der Anwendungen ermöglichen, werden bei jedem anstehenden Prozesswechsel nacheinander durchgeführt. Der Readback sichert sowohl den aktuellen Ausführungszustand des FPGA-Bausteins als auch den Inhalt der verwendeten RAMs. Bei der anschließenden Konfiguration werden diese Zustände

entsprechend wieder hergestellt, um die Anwendung weiter auszuführen.

Aufgabe des Betriebssystems ist es, diese beiden Schritte koordiniert nacheinander auszuführen. Aufgrund des Einflusses der Readback- und der Konfigurationszeiten ist hier auf eine möglichst gute Implementierung zu achten, die die Möglichkeiten des FPGA-Bausteins und des FPGA-Koprozessor optimal nutzen.

3.4.4.2 Zustandsrekonstruktion

Eine weitere Funktion die zur Durchführung des preemptiven Multitaskings notwendig ist analysiert den unterbrochenen Zustand der Schaltung und stellt diesen Zustand, unter Verwendung der direkten Bitstrommanipulation, in dem neuen Konfigurationsdatensatz wieder her. Unter Berücksichtigung einiger Randbedingungen ist somit die Fortführung der Schaltung an dem unterbrochenen Zustand gewährleistet.

Die Funktionsweise der Zustandsrekonstruktion beruht auf dem in Abschnitt 3.3.1 beschriebenen Prinzip und ist in dem folgenden Abschnitt 3.5 detailiert beschrieben. Einen direkten Einfluß auf den systematischen Overhead hat die Zustandsrekonstruktion nicht, da sie auf dem Host-Prozessor parallel zu der Anwendung auf dem FPGA-Koprozessor durchgeführt wird.

3.4.4.3 Verwaltungsaufgaben

Einer der zentralen Aufgaben übernimmt das Betriebssystem mit der Verwaltung der einzelnen voneinander unabhängigen Anwendungen, die parallel auf dem FPGA-Koprozessor ausgeführt werden. Neben der Speicherung des Konfigurationsbitstroms, des Readbackbitstoms und der Speicherinhalte sind weitere anwendungspezifische Informationen wie z.B. die Priorität, das Prozessalter oder der Prozesszustand für die Verwaltung der Anwendungen notwendig.

Basierend auf diesen Informationen und Zuständen werden die einzelnen Anwendungen vom Betriebssystem nach einem dynamischen Zeitplan erst für die Ausführung vorbereitet und anschließend zur Ausführung gebracht. Die Verwaltungsaufgaben und das Erstellen des Zeitplans erfolgt bei jedem Betriebssystemaufruf oder nach einer bestimmten Zeit,

die dem Prozess für die Verarbeitung zur Verfügung steht. Somit zählt die benötigte Zeit zur Ausführung der Verwaltungstätigkeit ebenfalls zu dem Overhead und unterliegt der Randbedingung möglichst schnell zu arbeiten.

3.4.4.4 Anwendungsausführung

Jede Anwendung besitzt einen exklusiven Zugriff auf den FPGA-Koprozessor innerhalb eines Zeitfensters, daß ihr für die Verarbeitung zur Verfügung steht. Diese exklusive Nutzung stellt zum einen die Grundlage für eine problemlose Verarbeitung der Anwendung dar und zum anderen ist die Datensicherheit über die Anwendung hinaus sichergestellt.

Diese Bündelung der Zugriffe auf den FPGA-Koprozessor erfolgt über das Betriebssystem. Somit liegt neben der Verwaltung der Daten auch die gesamte Anwendungsabwicklung vom Anmelden einer neuen Anwendung, über das Laden von Konfigurationen bis zum Ausführen der Datentransfers in der Hand des Betriebssystems.

3.4.5 Zusammenfassung

Alle Anforderungen die den Betrieb eines preemptiven Multitasking Betriebssystems ermöglichen bzw. in einer besonderen Weise unterstützen sind in der nachfolgenden Abbildung 3.11 noch einmal übersichtlich zusammengefaßt. Unterschieden wird grundsätzlich in Anforderungen die an die Hardware, also die verwendeten FPGA-Bausteine und den FPGA-Koprozessor, und die an die Software, einschließlich Betriebssystem und FPGA-Schaltung, gestellt werden.

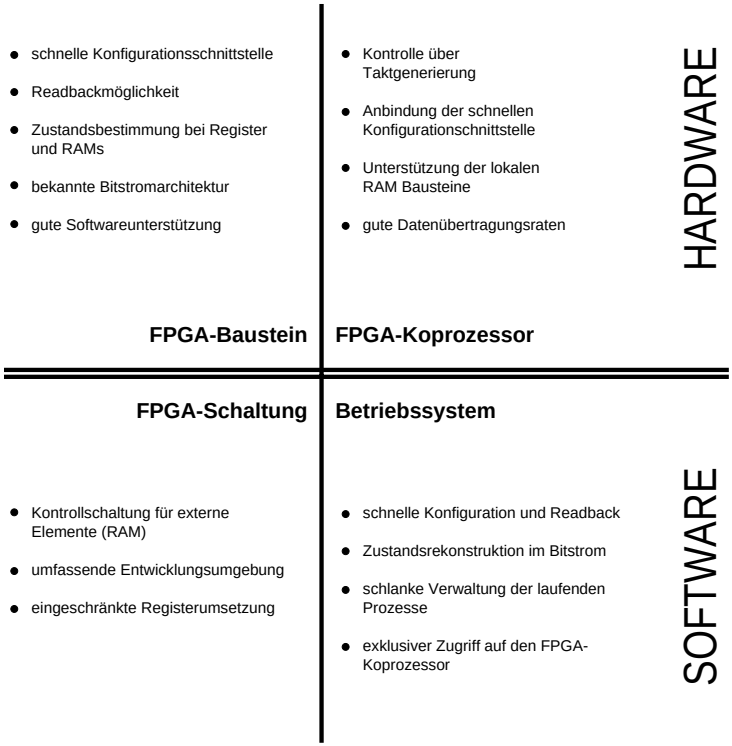


Abbildung 3.11: Auflistung der Anforderungen, die ein preemptives Multitasking Betriebssystem ermöglichen bzw. in einer besonderen Weise unterstützen sollte.

3.4.6 Baustein und FPGA-Koprozessor Auswahl

Unter Zugrundelegung der zuvor beschriebenen Anforderungen kann eine Bewertung der derzeit verfügbaren FPGA-Bausteine und FPGA-Koprozessor durchgeführt werden, die eine Einschätzung zur späteren Verwendbarkeit des Multitasking Betriebssystems ermöglicht.

Im der nachfolgenden Tabelle 3.2 sind die am Markt verfügbaren FPGA-Bausteine aufgelistet. Für den Vergleich wurden die FPGA-Bausteine von verschiedenen Herstellern untersucht, wobei jeweils ein Baustein einer FPGA-Serie für die Bewertung herangezogen wurde. Die Bewertungskriterien für den Vergleich leiten sich dabei direkt aus den zuvor beschriebenen Anforderungen ab und betreffen: die Geschwindigkeit der Konfigurationsschnittstelle und die Bereitstellung eines Readbacks zur Zustandsbestimmung der einzelnen Register und internen RAMs.

Die Untersuchung dieser Kriterien ist wichtig für die prinzipielle Eignung der Bausteine in einem Multitasking Betriebssystems. Darüberhinaus wurden weitere Kriterien untersucht, die für einen späteren praktischen Einsatz der Bausteine wichtig sind. Dazu zählen die Anzahl der verfügbaren Logikzellen und Qualität der Place&Route-Werkzeuge. Aufgrund der unterschiedlichen Ausprägung der Logikzellen wurden diese auf eine normierte Grundzelle mit einem Register und einer 4-Input Look-Up Table abgebildet. Erst diese Abbildung ermöglicht einen fairen Vergleich der FPGA-Bausteinserien untereinander. Das Verhältnis der normierten Zellen zu der für den Baustein benötigten Konfigurationszeit erlaubt dann einen Vergleich der verschiedenartigen Bausteine bezüglich ihrer Konfigurationsschnittstellen.

Zum Vergleich wurde auch der FPGA-Baustein der Xilinx XC6216 aufgenommen, der bei den schon existierenden Betriebssystemen zum Einsatz kommt, aber nicht mehr vertrieben wird.

Wie aus der Tabelle 3.2 ersichtlich ist, sind nur zwei der angegebenen Bausteine vollkommen ungeeignet für den Einsatz in einem Multitasking Betriebssystem. Diese beiden Bausteine von Altera und von Actel verfügen über keine Readbackmöglichkeit und besitzen zudem eine vergleichsweise langsame Konfigurationsschnittstelle. Alle weiteren aufgeführten Bausteine können in einem Multitasking Betriebssystem eingesetzt werden.

Bei der Auswahl der verwendbaren FPGA-Koprozessoren spielt maßgeblich die Datenaustauschrate zwischen dem Host- und dem FPGA-Koprozessor eine Rolle. Diese limitiert unter anderem die Geschwindigkeit der Konfigurationsschnittstelle und hat somit einen großen Einfluß auf den systematischen Overhead des Betriebssystems.

Weitere Kriterien für die Auswahl der FPGA-Koprozessoren ist der zum Einsatz kommende FPGA-Baustein und die Verfügbarkeit von lokalen RAM-Bänken. Eine vollständige Liste der verfügbaren FPGA-Prozessoren ist unter [Opt] zu finden.

Die gesamte Gruppe der 'Stand-Alone' FPGA-Prozessoren ist unter Berücksichtigung der Kriterien ungeeignet, da diese meist nur einen sehr langsamen Datenaustausch bereitstellen. Darüberhinaus sind sie zum Zweck der Bausteinevaluation oder zur Verarbeitung eines speziellen Anwendung erstellt und somit für den allgemeinen Einsatz ungeeignet. Sie werden im weiteren Verlauf der Arbeit nicht mehr berücksichtigt.

Die zweite Gruppe der FPGA-Koprozessoren ist als Hardwareplattform für das Multitasking Betriebssystem geeignet, da sie in der Regel über eine hohe Datenaustauschrate verfügen. Die Mehrzahl der insgesamt 43 FPGA-Koprozessoren [Opt] verwenden entweder den PCI-Bus (51%) oder die industrielle Ausführung CPCI-Bus (7%). Diese Verbindung ist mit einer theoretischen Datenrate von 132MByte/s sehr gut für den Multitaskingbetrieb geeignet, wenngleich nicht alle FPGA-Koprozessoren diese Datenrate auch umsetzen. Weitere Schnittstellen wie SBUS, PMC-PCI, VME oder ISA verfügen mit Ausnahme der ISA Schnittstelle ebenfalls über gute Datenraten und sind daher genauso geeignet.

Unter der Berücksichtigung der beiden weiteren Auswahlkriterien – FPGA-Baustein und lokale RAMs – sind es insgesamt 15 FPGA-Koprozessoren die für einen Einsatz verwendet werden können.

Die Anforderungen, die sowohl an die FPGA-Schaltung als auch an das Betriebssystem selbst gestellt werden, sind entsprechend durch den Schaltungs-Entwickler bzw. innerhalb der Umsetzung des Betriebssystems durchzuführen. Sie stellen nur bedingt ein Auswahlkriterium für die Wahl des FPGA-Bausteins und des FPGA-Koprozessor dar.

Baustein Name	Konfigurations-schnittstelle	Read-back	Zustands-rekonst.	normierte Logikzellen	Progammier-Werkzeuge	Partielle-Rekonfig.	Logikzellen / Konfig.Zeit
Xilinx XL4085	10MHz / 8Bit 23,8ms	Ja	Ja	6.272	++	Nein	263 Zellen/ms
Xilinx XV400	50MHz / 8Bit 6,4ms	Ja	Ja	10.800	++	Ja	1688 Zellen/ms
Lucent 3T125	25MHz / 8Bit 4,3ms	Ja	Ja	7.056	+	Ja	1640 Zellen/ms
Altera 10K100	10MHz / 1Bit 117ms	Nein	Ja	4.992	—	Nein	42 Zellen/ms
Atmel AT40K05	10MHz / 8Bit 0.25ms	Ja	Ja	256	- -	Nein	1024 Zellen/ms
Actel A65ES100	10MHz / 1 Bit 300ms	Nein	Ja	4096	—	Nein	14 Zellen/ms
Xilinx XV6216	25MHz / 32Bit 0,2ms	Ja	Ja	16.382[a]	- -	Ja	11520 Zellen/ms

Tabelle 3.2: Vergleich der FPGA-Bausteine im Hinblick auf deren Verwendbarkeit für ein preemptives Multitasking Betriebssystem.

[a] Logikzellen des XC6216 entsprechen nicht den normierten Grundzellen, da sie aus 5 Multiplexern und einem Register besteht. Angegeben ist die Anzahl an verfügbaren Zellen.

3.5 FPGA Anwendungsrekonstruktion

Für die Umsetzung der in Abschnitt 3.3.1 beschriebenen Zustandsrekonstruktion einer Anwendung müssen spezielle Kenntnisse über den Baustein und dessen Verhalten bekannt sein, die auch als Anforderungen an den FPGA-Baustein in Abschnitt 3.4.1 formuliert sind.

Relevant ist zum einen der Aufbau des Bitstroms und die Kenntnis über die Zuordnung der einzelnen Bits zu den damit verbundenen Funktionen. Detaillierte Kenntnis über die Positionen zum Auslesen der aktuellen Zustände und zum Definieren der Anfangszustände von Registeren und RAMs sind für eine Zustandsrekonstruktion unerlässlich. Unterschieden wird dabei zwischen dem Konfigurationsstrom und dem Readbackstrom, da der Aufbau sich ein wenig voneinander unterscheidet. Zum anderen ist es ebenfalls wichtig das Verhalten des Bausteins zu kennen, um die Umsetzung dieser Zustandsrekonstruktion zuverlässig zu gewährleisten.

In diesem Abschnitt werden die Grundlagen in Form des Bitstromaufbaus, das Vorgehen bei der Zustandsermittlung und der -rekonstruktion und die Abfolge der Rekonstruktion auf dem FPGA-Baustein im Detail erläutert. Aufgrund der unterschiedlichen internen FPGA-Architekturen und der darauf aufbauenden Bitstromarchitekturen wird die Zustandsrekonstruktion anhand eines konkreten Beispiels, dem Xilinx Virtex XV400, erläutert.[6] Diese Bausteinserie besitzt die derzeit am besten dokumentierte Bitstromarchitektur.

3.5.1 Bitstromarchitektur des Virtex FPGAs

Der Aufbau des Konfigurations- bzw. des Readbackbitstroms ist bei dem Virtex Baustein bis auf verschiedene Offsetwerte identisch. Die Architektur selbst ist stark abhängig von dem Aufbau und der Anordnung der Logikzellen und des Verbindungsnetzwerks, da die Speicherbits der Konfigurationsebene in unmittelbarer Nähe zu den entsprechenden Blöcken auf der Funktionsebene angeordnet sind. Bedingt durch diesen Zusam-

[6] Die Zustandsrekonstruktion ist übertragbar auf alle anderen FPGA-Bausteine und konnte auch in Teilen für die Bausteinreihe Xilinx XC4000EX gezeigt werden.

menhang und durch den regelmäßigen Aufbau des Zellen-Arrays erfolgt der Aufbau des Bitstroms in mehreren hierarchischen Stufen.

Im Fall der Virtex Bausteinfamilie wurde für die oberste Hierarchieebene eine spaltenorientierte Unterteilung des Zellenarrays gewählt, die in Abbildung 3.12 illustriert ist. Wie in der Abbildung dargestellt, wer-

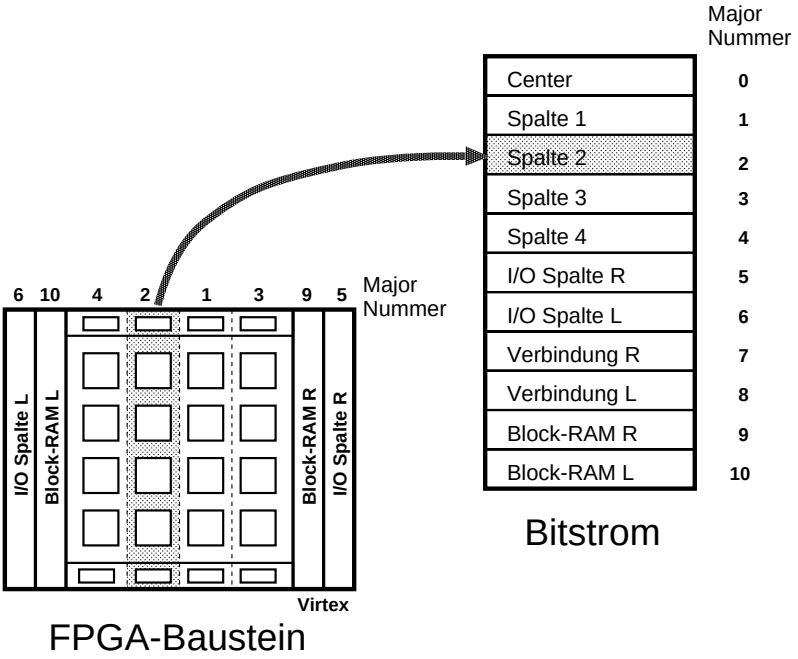


Abbildung 3.12: Vereinfachte Abbildung der einzelnen Spalten eines Virtex Bausteins auf die oberste Hierarchieebene des Bitstroms.

den nach der Definition von Xilinx anfangend mit den innersten Netzwerkverbindungen die einzelnen Spalten nacheinander zu einem vollständigen Bitstrom zusammengefügt. Die Spalten des Zellenarrays werden dabei alternierend von innen nach außen an den Bitstrom angehängt. Mit dem Abschluß des Zellenarrays folgen die Verbindungen und Einstel-

lungen für die I/O-Zellen auf der rechten und linken Seite und vier weitere Blöcke für das in den Baustein integrierte RAM. Der gesamte Bitstrom wird somit von fünf unterschiedlichen Blöcken beschrieben, die in Größe und Aufbau variieren und unterschiedliche Bereiche im FPGA-Baustein konfigurieren. Diese sind: Zentrale Verbindungsnetzwerk, Logikzellenspalte, I/O-Zellenspalte, RAM-Verbindungen und RAMs. Bezeichnet werden die einzelnen Spaltenblöcke mit einer fortlaufenden 'Major'-Nummer, um diese eindeutig adressieren zu können.

Die zur Zustandsrekonstruktion wichtigen Spalten sind die Logikzellenspalten, die I/O-Zellenspalten und die RAM-Spalten. Anhand einer Logikzellenspalte wird der weitere Aufbau des Bitstroms erläutert. Näheres zu den anderen Blockdefinitionen sind in [Kel00] zu finden.

Der Aufbau einer Logikzellenspalte bildet die zweite Hierarchieebene des Bitstroms und definiert die Einstellung der gesamten Spalte. Dazu gehören die jeweils oben und unten angeschlossenen I/O-Zellen, die Logikzellen innerhalb der Spalte und alle Verbindungen des Verbindungsnetzwerks, die sich innerhalb der Spalte befinden. Aufgebaut ist dieser Block aus 48 einzelnen 'Frames' die jeweils mit einer 'Minor'-Nummer adressiert werden. Der gesamte Aufbau einer Logikzellenspalte mit allen Frames ist in Abbildung 3.13 dargestellt.

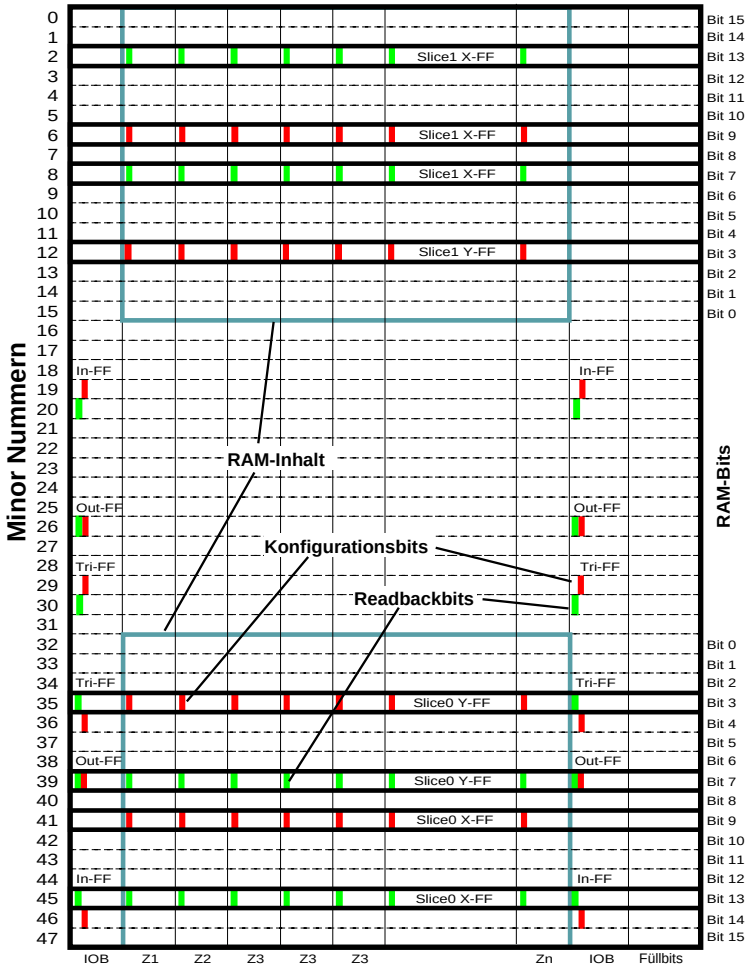


Abbildung 3.13: Abbildung des Aufbaus eines Spaltenblocks mit seinen 48 'Minor' Frames. *Die roten Bits markieren die Positionen der zustandsbestimmenden Konfigurationsbits und auf den grünen Positionen im Readbackbitstream kann der Zustand ausgelesen werden.*

In der dritten Hierarchieebene ist nun noch jede dieser 'Minor'-Frames unterteilt in jeweils 18 Bit breite Zellen. Jede dieser Zellen ist einer Logikzelle oder I/O-Zelle innerhalb der Spalte auf dem FPGA-Baustein zugeordnet und ermöglicht somit eine exakte Adressierung jeder einzelnen Logikzellen. Abbildung 3.14 zeigt diesen Zusammenhang graphisch da.

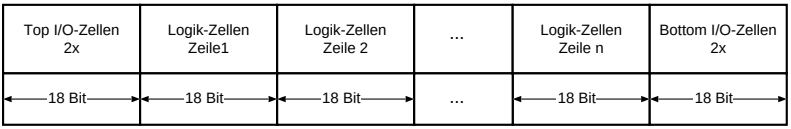


Abbildung 3.14: Abbildung des Aufbaus eines 'Minor'-Frames.

Innerhalb der zweiten Hierarchieebene bestimmt jeder 'Minor'-Frame eine Vielzahl von Einstellungen an den Logikzellen, den I/O-Zellen und den Verbindungsnetzwerken der jeweiligen Array-Reihe.

In Abbildung 3.13 sind die Bits rot markiert, die den jeweils eingestellten Anfangszustand der Register definieren. Bei der Virtex Serie sind es vier Register innerhalb einer Logikzelle, die auf vier 'Minor'-Frames (6, 12, 35 und 41) verteilt initialisiert werden. Innerhalb jedes dieser 'Minor'-Frames werden durch den 'Minor'-Frame Aufbau alle untereinanderliegenden Logikzellen, anfangend mit der Zelle in Zeile 1, definiert. Zusätzlich sind noch die Zustandsbits für die I/O-Zellen, die oben und unten an die Logikzellenspalte angrenzen, angegeben. Sie werden durch Bits in sechs 'Minor'-Frames definiert (19, 26, 29, 36, 39 und 46). Die Zustandsbits für die RAMs[7] verteilen sich über 32 'Minor'-Frames (0–15 und 32–47) sind aus Gründen der Übersichtlichkeit in der Abbildung nur angedeutet.

Wie schon in Abschnitt 3.4.1.4 erwähnt stimmen die Positionen, die im Konfigurationsstrom den Anfangszustand definieren nicht mit den Positionen des Registerzustands im Readbackstrom überein. Dies gilt für alle Register innerhalb des FPGA-Bausteins jedoch nicht für die ein-

[7] Bei diesen RAMs handelt es sich um Logikzellen, welche die Look-Up Tabelle zum Speichern von Daten verwenden. Das so konfigurierte RAM ist 16x1Bit groß.

zelnen Bits der RAMs. Der Zustand der jeweiligen Logikzellenregister im Readbackbitstrom ist ebenfalls definiert in vier 'Minor'-Frames (2, 8, 38 und 45), die in Abbildung 3.13 zur besseren Unterscheidung in Grün angezeichet sind. Ergänzend sind auch die Positionen der Register der angrenzenden oberen und unteren I/O-Zellen mit in die Abbildung aufgenommen.

Zusätzlich zu der Position der einzelnen Register oder RAM-Zellen ist auch die Kodierung innerhalb des Bitstroms ein wichtiger Punkt um den jeweils gewünschten Zustand einzustellen. Die nachfolgende Tabelle 3.3 zeigt einen Überblick über die einzelnen Registertypen und deren Kodierungen im Konfigurations- und Readbackbitstrom.

	Konfiguration		Readback	
Register	HIGH	LOW	HIGH	LOW
Logik	*1*	*0*	*1*	*0*
I/O-Output	*1*	*0*	*1*	*0*
I/O-Input	*1*	*0*	*1*	*0*
I/O-Tristate	*0*	*1*	*1*	*0*

Tabelle 3.3: Kodierung der Registerzustände von ausgewählten Registern im Konfigurations- und Readbackbitstrom.

3.5.2 Zustandsermittlung

Bei der Zustandsermittlung ist es wichtig, alle zustandsbeschreibenden Signale entsprechend ihres aktuellen Zustandes zu sichern, d.h. die Zustände aller speichernden Elemente einer Schaltung müssen ermittelt werden.

Die einheitliche Vorgehensweise ist unterteilt in zwei aufeinanderfolgende Schritte: Readback und Analyse. Mit dem Readback wird der aktuelle Zustand aus dem FPGA-Baustein ausgelesen. Dieser Readbackbitstrom enthält alle notwendigen Informationen bzw. Zustände, die für die Zustandsermittlung notwendig sind. Dies sind insgesamt nur $\approx$10% des gesamten Readbackbitstroms. Der größere restliche Teil besteht aus Konfigurationsdaten die z.B. die Verbindungen des Netzwerks innerhalb

des FPGA-Bausteins bestimmen. Aus diesem Grund und aufgrund der unterschiedlichen Positionen zwischen dem Readback- und dem Konfigurationsbitstrom, der in Abbildung 3.13 graphisch dargestellt ist, muß der Readbackbitstrom nach dem Auslesen in einem zweiten Schritt vom Betriebssystem, das auf dem Host-Prozessor läuft, analysiert werden. Diese Analyse ermittelt die Zustände aller speichernden Elemente indem die verteilten Bits im Readbackbitstrom einzeln adressiert und ausgelesen werden. Für den ausgewählten FPGA-Baustein der Virtex Serie speichern folgende Elemente Schaltungsdaten: Logikzellen Register, I/O-Zellen Register, Look-up Table RAMs und Block-RAMs.

In der nachfolgenden Liste sind die einzelnen Elemente, die alle oder nur zum Teil auch in anderen FPGA-Bausteinen vorhanden sind, näher erläutert. Die Adressierung der einzelnen Zustandsbits ist aufgrund des hierarchischen Aufbaus des Bitstroms ähnlich und wird daher nur exemplarisch für die Logikzellen Register des XV400 beschrieben.

Logikzellen Register: Bei dem XV400 Baustein sind jeweils vier normierte Grundzellen in einer Logikzelle vereint, so das auch vier Registerbits analysiert werden müssen. Insgesamt müssen 9600 Registerbits für den gesamten XV400 analysiert werden. Eine Logikzelle ist dabei unterteilt in zwei 'Slices' mit jeweils einem X- und einem Y-Register (siehe auch Abbildung 3.13 auf Seite 82 und Abbildung 3.5 auf Seite 58). Für die Ermittlung der jeweiligen Bitadresse innerhalb des Bitstroms kommen folgende Formeln zum Einsatz, die sowohl die Byte- als auch den Bitoffset innerhalb des adressierten Bytes bestimmen:

$$Major = \begin{cases} Spalte \leq \frac{Spaltenzahl}{2} & : \quad Spaltenzahl - Spalte * 2 + 2 \\ Spalte > \frac{Spaltenzahl}{2} & : \quad Spalte * 2 - Spaltenzahl - 1 \end{cases} \tag{3.1}$$

$$Minor = \begin{cases} Slice = 0 \text{ und } Reg. = X & : \quad 45 \\ Slice = 0 \text{ und } Reg. = Y & : \quad 39 \\ Slice = 1 \text{ und } Reg. = X & : \quad 2 \\ Slice = 1 \text{ und } Reg. = Y & : \quad 8 \end{cases} \tag{3.2}$$

$$FrameAdresse = (8 + (Major - 1) * 48 + Minor) \tag{3.3}$$

$$ByteAdresse = (FrameAdresse * Framebits + (18 * Zeile + 1))\%8 \quad (3.4)$$

$$BitAdresse = (18 * Zeile + 1) \gg 8 \quad (3.5)$$

Mit diesen Formeln ist die Berechnung der Adresse jedes einzelnen Bits garantiert und der Zustand der gesamten Logikzellen Register ist somit analysierbar.

I/O-Zellen Register: Die I/O-Zellen des XV400 Bausteins unterscheiden sich von den Logikzellen und verfügen über drei individuelle Register pro Baustein-Pin. Die drei Register speichern jeweils das Eingangssignal, das Ausgangssignal und das Steuersignal des Tristate-Treibers. Ein kombinatorisches Funktionselement ist nicht vorhanden. Insgesamt verfügt der XV400 Baustein über 1440 dieser I/O-Register die es ebenfalls zu sichern gilt.

Die Bitpositionen dieser Register verteilen sich im Gegensatz zu den Registern der Logikzellen sowohl auf die Logikzellenspalten aber auch auf die I/O-Zellenspalte. Alle Pins am oberen und unteren Ende einer Logikzellen Spalte sind jeweils paarweise benachbart in den Logikzellenspalten zusammengefaßt. Zu sehen ist das auch an der ersten und der vorletzten Spalte der 'Minor'-Framebeschreibung in Abbildung 3.13. Die weiteren Register der I/O-Zellen auf der rechten und linken Seite des FPGA-Bausteins sind in eigenen I/O-Zellenspalten zusammengefaßt.

Look-Up Table RAMs: Bei dem behandelten Baustein besteht die Möglichkeit, die für kombinatorische Logik vorgesehene Look-Up Tabelle auch als ein 16x1Bit RAM-Speicher zu betreiben, um kleine Datenmengen zwischenzuspeichern. Aufgrund der Logikzellenaufteilung wird auch hier in zwei 'Slices' und jeweils eine G- und eine F-LUT unterschieden. Für eine vollständige Rekonstruktion der Schaltung müssen auch diese Elemente entsprechend analysiert werden, solange sie als RAMs genutzt werden. Diese Einstellungen der Look-Up Tabellen kann dabei leicht aus den Konfigurationsdaten, die zusammen mit dem Readbackbitstrom vorliegen, ausgelesen werden.

Bei der Analyse müssen für jedes einzelne Bit (siehe auch Abbildung 3.13) der 16 RAM-Adressen jeweils die Bitstromadresse ausgerechnet werden und der Zustand aus dem Readbackbitstrom ausgelesen werden.

Die Anzahl der zu sichernden RAMs ist aufgrund der Unterscheidung des Verwendungszwecks der Look-Up Tabellen abhängig von der zu sichernden FPGA-Schaltung. Eine Sicherung und Rekonstruktion von Look-Up Tabellen die kombinatorische Funktionen ausführen ist nicht notwendig.

Block-RAMs: Zusätzlich zu den umgewandelten Look-Up Tabellen verfügen moderne FPGA-Bausteine wie der XV400 über weitere feste RAM-Blöcke die zur Sicherung von größeren Datenmengen (4kBit) verwendet werden können. Auch diese Blöcke gehören zu den speichernden Elementen einer FPGA-Schaltung und müssen daher ebenfalls gesichert werden. Dem XV400 Baustein stehen insgesamt 20 dieser RAM-Blöcke zur Verfügung die insgesamt 80KBit umfassen. Der Inhalt dieser verteilten RAM-Blöcke ist in separaten 'Major'-Blöcken kodiert (siehe auch Abbildung 3.12) und kann somit als ein zusammenhängender Speicherblock aus dem Readbackbitstrom extrahiert werden. Aus diesem Grund müssen bei der Analyse dieser RAM-Daten nur zwei Anfangsadressen berechnet werden, wodurch eine komplexe Ermittlung der einzelnen Bitpositionen entfällt.

3.5.3 Zustandsrekonstruktion

Ziel der Rekonstruktion ist es nach dem Konfigurieren des Bitstroms und dem Start des FPGA-Bausteins den gleichen Zustand in allen speichernden Elementen der Schaltung wieder herzustellen und somit die unterbrochene und verdrängte Anwendung an der gleichen Stelle weiter fortzuführen.

Die Zustandsrekonstruktion erfolgt dabei in umgekehrter Reihenfolge zu der Zustandsermittlung ebenfalls in zwei Schritten: Zustandsrekonstruktion und Konfiguration. In dem ersten Schritt wird der Zustand, wie er vor der Unterbrechnung im FPGA-Baustein angehalten

wurde, entsprechend rekonstruiert. Diese Aufgabe übernimmt ebenfalls das Betriebssystem das auf dem Host-Prozessor ausgeführt wird. Die Rekonstruktion selbst erfolgt durch die direkte Manipulation des Konfigurationsbitstroms, da nur so eine schnelle Zustandsrekonstruktion in wenigen Millisekunden durchgeführt werden kann. Zwischen der Analyse und der Rekonstruktion besteht eine 1:1 Beziehung in der jedes Register und jedes RAM-Bit in den Zustand versetzt wird, das zuvor aus dem Readbackbitstrom analysiert wurde. Zu diesem Zweck werden, wie zuvor bei der Zustandsanalyse, nur einzelne zustandsdefinierende Konfigurationsbits verändert die weder Einfluß auf die Schaltungsfunktion noch auf die Schaltungsverbindungen haben. Diese einzelnen Bits im Konfigurationsbitstrom und die unterschiedlichen Bitpositionen erfordern auch hier eine genaue Berechnung der jeweiligen Adresse innerhalb des Bitstroms. Sie sind für die daran anschließende Manipulation notwendig. In dem zweiten Schritt wird der FPGA-Baustein mit dem entsprechend angepaßten Bitstrom wieder konfiguriert und erneut gestartet, wobei der intere Startzustand dem zuvor abgebrochenen Zustand entspricht.

In der nachfolgenden Liste sind noch einmal die vier speichernden Grundelemente der Virtex Bausteine aufgeführt. Erklärt wird, wieder anhand der Logikzellen Register, wie die Adressierung der einzelen Zustandsbits in dem Konfigurationsbitstrom berechnet wird und welche Besonderheiten bei der Registerekonstruktion der einzelnen Elemente zu beachten ist.

Logikzellen Register: Die Rekonstruktion des Zustands in diesen Logikzellen Registern erfolgt nach der in Abschnitt 3.4.1.3 beschriebenen Methode. Die Anfangszustände von jedem der vier Register einer Logikzelle werden entsprechend der zuvor analysierten Zustände und der Kodierungstabelle entweder auf *HIGH* oder *LOW* gesetzt. Diese Veränderung bewirkt, daß nach der Konfiguration des FPGA-Bausteins der Zustand aller Register den jeweils eingestellten Wert annehmen und die Anwendung in den Zustand setzt, der zum Zeitpunkt der Unterbrechung bestanden hat. Wie in Abbildung 3.13 dargestellt, sind die Positionen der Konfigurationsbits, die den Anfangszustand definieren, bis auf den 'Minor' Frame identisch. Aus diesem Grunde unterscheiden sich die Formeln zur

Berechnung der jeweiligen Bitposition nur in der Berechnung des 'Minor' Wertes:

$$Major = \begin{cases} Spalte \leq \frac{Spaltenzahl}{2} & : \quad Spaltenzahl - Spalte * 2 + 2 \\ Spalte > \frac{Spaltenzahl}{2} & : \quad Spalte * 2 - Spaltenzahl - 1 \end{cases} \tag{3.6}$$

$$Minor = \begin{cases} Slice = 0 \text{ und } Reg. = X & : \quad 41 \\ Slice = 0 \text{ und } Reg. = Y & : \quad 35 \\ Slice = 1 \text{ und } Reg. = X & : \quad 6 \\ Slice = 1 \text{ und } Reg. = Y & : \quad 12 \end{cases} \tag{3.7}$$

$$FrameAdresse = (8 + (Major - 1) * 48 + Minor) \tag{3.8}$$

$$ByteAdresse = (FrameAdresse * Framebits + (18 * Zeile + 1))\%8 \tag{3.9}$$

$$BitAdresse = (18 * Zeile + 1) \gg 8 \tag{3.10}$$

I/O-Zellen Register: Genauso wie bei den zuvor beschriebenen Logikzellen Register erfolgt auch bei den Registern der I/O-Zellen die Zustandsrekonstruktion durch das Setzen der entsprechenden Anfangszustände die die Register nach der Konfiguration annehmen. Für die jeweiligen Input-, Output- und Tristateregister einer I/O-Zelle, die an der oberen und der unteren Seite des XV400 Bausteins angeschlossen sind, werden die Anfangszustände ebenfalls in der Logikzellenspalte definiert was auch in der Abbildung 3.13 zu sehen ist. Die 'Minor'-Frames 19, 26, 30, 36, 39 und 46 beinhalten die jeweiligen Bitpositionen für die Register der oberen und der unteren I/O-Zelle. Die I/O-Zellen, die auf der rechten und linken Seite des Bausteins angeschlossen sind, werden innerhalb der entsprechenden I/O-Zellenspalten gesetzt. Dort werden jeweils drei benachbarte I/O-Zellen zusammengefaßt. Die Position dieser I/O-Zellenspalten innerhalb des gesamten FPGA-Bitstroms ist in Abbildung 3.12 zu sehen.

Look-Up Tabellen RAMs: Bei der Rekonstruktion des RAM-Zustandes sind die entsprechenden Bitpositionen im Readback- und

Konfigurationsbitstrom übereinstimmend, so daß die berechneten Bitpositionen, die zum Analysieren der Zustände verwendet wurden, auch für die Rekonstruktion eingesetzt werden können. Wie zuvor bei der Analyse der einzelnen RAM-Bits werden auch bei der Rekonstruktion die Bits der RAM-Adressen einzeln in den Konfigurationsbitstrom geschrieben. Bei dieser Rekonstruktion ist ebenfalls zu analysieren, ob die adressierte Look-Up Tabelle in ihrer Funktion ein speicherndes oder ein kombinatorisches Element darstellt. Diese Überprüfung erfolgt wie schon zuvor mit einer Analyse einzelner Bits des übrigen Konfigurationsbitsstroms.

Block-RAMs: Die letzten, zu rekonstruierenden Elemente eines Virtex Bausteins sind die Block-RAMs. Wie zuvor bei dem Look-Up Tabellen RAMs sind auch hier die Bitpositionen der einzelnen RAM-Bits identisch, so daß für die Rekonstruktion keine neuen Adressen errechnet werden müssen. Zusätzlich kann bei den Block-RAMs die Rekonstruktion mit nur wenigen Befehlen durchgeführt werden, da die analysierten Daten zusammenhängend in den Konfigurationsbitstrom kopiert werden können.

Nach der Rekonstruktion des Konfigurationsbitstroms ist der zuvor unterbroche Zustand der FPGA-Schaltung im Bitstrom wieder hergestellt und kann bei einer erneuten Konfiguration des FPGA-Bausteins diesen initialisierten Zustand wieder hergestellen. Diese Rekonstruktion erfolgt bei den Registern durch die Initialisierung des Anfangszustands und bei den RAM-Elementen durch das Verändern der Initialisierung selbst. Bei einer erneuten Prozesszuteilung durch das Betriebssystem wird der FPGA-Bausteins mit diesem modifizierten Bitstrom konfiguriert und gestartet. Der eigentliche Start der Schaltung erfolgt automatisch durch ein spezielles Startkommando, das am Ende des Konfigurationsbitstroms mit übertragen wird. Dieses Startkommando initiiert eine Start-up Sequenz, die den FPGA-Baustein und die konfigurierte Schaltung innerhalb von mehreren Schritten einschaltet. Die nachfolgende Darstellung 3.15 zeigt die einzelnen Schritte die bei dieser Start-up Sequenz ausgeführt werden.

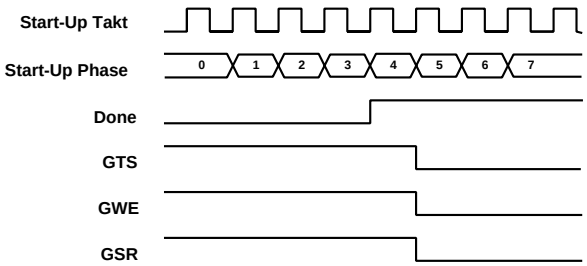


Abbildung 3.15: Start-Up Sequenz nach der Konfiguration des Virtex Bausteins.

Die Reihenfolge der einzelnen, in der nachfolgenden Auflistung beschriebenen Schritte kann durch den Entwickler eingestellt werden.

DONE Pin: Das DONE Signal wird als einziges der erzeugten Signale über den DONE Pin nach aussen geführt. Es signalisiert dem Betriebssystem, das die Konfiguration ohne Fehler durchgeführt wurde und der Baustein normal startet.

GTS Signal: Das Global Tristate Signal schaltet alle Tristate-Treiber der I/O-Zellen die von der Schaltung verwendet werden zur gleichen Zeit ein. Dies verhindert ein zeitlich nicht synchrones Einschalten der Ausganssignale, das zu Datenverlusten oder zu gegeneinander treibenden Bausteinen führen kann.

GWE Signal: Das Global Write Enable Signals aktiviert die RAMs und die Register der FPGA-Schaltung. Das Setzen dieses Signals aktiviert somit die gesamte Schaltung.

GSR Signal: Das aktivierte Global Set-Reset Signal versetzt alle Register einer Schaltung in den eingestellten Initialisierungszustand. Es sollte zusammen mit den GWE Signal geschaltet werden um die Schaltung zu starten.

Ein aktiviertes GSR Signal veranlasst den Wechsel der einzelnen Register in den im Konfigurationsbitstrom angegeben Anfangszustand. Die

RAM-Bits werden im Unterschied zu den Registern ohne das GSR Signal gesetzt, da die Positionen im Konfigurationsbitstrom direkt in den Speicher übernommen werden. Das gleichzeitige Freigeben der Schaltung erfolgt durch das Aktivieren des GWE Signals. Somit ermöglicht die Abschaltung des GSR Signals und die gleichzeitige Aktivierung des GWE Signals einen synchronen Start der Schaltung.

Die Zustandsrekonstruktion des gesamten FPGA-Bausteins über die Initalisierung verhindert jedoch eine allgemeine Verwendung des GSR Signals als Reset für die Schaltung, denn eine Aktivierung des GSR Signals veranlasst die Schaltung in den zuvor restaurierten Zustand und nicht in den ursprünglichen Reset Zustand der Anwendung zurückzukehren. Ein eventuell notwendige Resetschaltung ist daher getrennt von dem GSR Signal aufzubauen.

3.5.4 CRC Kontrolle

Eine weitere Komponente des Konfigurationsbitstrom ist der CRC (Cyclic Redundancy Check), der eine Kontrolle des übertragenen Bitstroms ermöglicht. Die CRC Kontrolle verhindert die Ausführung einer durch einen Bitstrom- oder einen Übertragungsfehler veränderten Konfiguration auf dem FPGA-Baustein. Solche Fehler können zu einer veränderten Schaltungsfunktionalität bzw. im ungünstigen Fall zu einer teilweisen Zerstörung des Bausteins führen.

Die Zustandsrekonstruktion manipuliert $\approx$10% des gesamten Konfigurationsbitstrom, so daß der ursprünglich errechnete CRC-Vergleichswert nicht mehr übereinstimmt und es daher zu einem Abbruch der Start-Up Sequenz kommt. Um diesen Startabbruch zu vermeiden, ist es notwendig, entweder den CRC-Vergleichswert basierend auf dem manipulierten Bitstrom neu zu errechnen oder die CRC Kontrolle im Baustein abzuschalten. Aufgrund der, für eine Neuberechnung benötigen Zeit, wird für diesen Ansatz die Abschaltung der CRC Kontrolle angewendet. Möglich ist dies da in dem Konfigurationsbitstrom nur Bits verändert werden, die den Initialisierungszustand von Registern und RAMs beeinflussen. Die Schaltungsfunktion und – noch wichtiger – die Verbindungen innerhalb der Schaltung werden zu keiner Zeit verändert, so das es auch zu keiner Zerstörung des Bausteins kommen kann.

3.6 Funktionsaufbau und Messungen

Die praktische Umsetzung der beiden zuvor beschriebenen Funktionen für die Ausführung im Betriebssystem wird in diesem Abschnitt erläutert. Basierend auf der Bitstromarchitektur und der Bitkodierung wurden vier Funktionen entwickelt die eine schnelle und direkte Rekonstruktion ermöglichen. Im Hinblick auf eine optimale und damit schnelle Ausführung der Rekonstruktion wird der Aufbau der Funktionen näher erläutert. Anschließend durchgeführte Messungen und Vergleiche zeigen deren Leistungsfähigkeit und dienen gleichzeitig zur Abschätzung der später erreichbaren Zeit für einen Prozesswechsel.

3.6.1 Aufbau der Rekonstruktionsfunktionen

Ziel der Funktionen ist die Analyse und die Rekonstruktion des Zustands aller Register und RAMs einer Schaltung. Diese Rekonstruktion, die einen Teil des Prozesswechsels darstellt, wird von dem Betriebssystem durchgeführt, das auf dem Host-Prozessor neben den anderen dort ablaufenden Anwendungen verarbeitet wird. Zur Vermeidung eines zu großen Einflußes auf diese Anwendung wird zusätzlich ein weiteres Ziel verfolgt: Die Ausführungszeit der Funktionen ist durch eine geeignete Verarbeitungsreihenfolge auf ein Minimum zu beschränken.

Insgesamt ist die Analyse und Rekonstruktion auf vier Funktionen verteilt, die jeweils einzeln die Logikzellen Register, die I/O-Zelle Register, das Look-Up Tabellen RAM und die Block-RAMs bearbeiten. Diese funktionale Aufteilung ist vor allen bei den Funktionen, die die RAMs rekonstruieren von Vorteil, da die berechneten Bitadressen sowohl für den Readback als auch für den Konfigurationsbitstrom verwendet werden können. Ein weiterer Vorteil dieser Aufteilung liegt in der Vermeidung einer Zwischenspeicherung der einzelnen Statuswerte, da diese direkt im Readbackbitstrom ausgelesen und dann sofort anschließend im Konfigurationsbitstrom angepaßt werden können.

Diese Vorgehensweise, bei der jedes Registerbit einzeln und nacheinander verarbeitet wird ist dennoch geprägt von sehr langen Ausführungszeiten, die in den häufigen Berechnungen der Bitpositionen begründet sind. Bei der Betrachtung des hierarchischen Bitstromaufbaus, wie er

in Abbildung 3.13 abgebildet ist, fällt eine immer wiederkehrende Regelmäßigkeit der einzelnen Bits auf. Dies gilt sowohl für die Register als auch für die Look-Up Tabellen RAMs. Unter Ausnutzung dieser Regelmäßigkeit erfolgt die Berechnung der einzelnen Bitpositionen nicht mehr absolut zum Anfang des Bitstroms sondern relativ zu dem jeweils ersten Bit in einem Frame. Unter Verwendung dieser relativen Adressierung werden die dafür notwendigen Rechenoperationen auf ein Minimum reduziert und dadurch auch die Ausführungszeit verringert. Die bei dieser relativen Adressierung nicht vorhandene streng monotone Reihenfolge der Register ist unerheblich solange die 1:1 Beziehung zwischen den beiden Bitposition im Readback- und Konfigurationsbitstrom erhalten bleibt.

Anhand des Beispiels der Rekonstruktion aller Logikzellen Register wird die Vorgehensweise verdeutlicht. Dargestellt in Abbildung 3.16 werden am Anfang der Funktion die Offsetwerte der vier Registerpositionen innerhalb des ersten 'Major'-Frames mit den Formeln (3.1 - 3.5) berechnet. Abgelegt in jeweils vier unabhängigen Variablen zeigen sie direkt auf die Bitpositionen (Readback) der vier Register (S0X, S0Y, S1X und S1Y) aus der Logikzelle in der ersten Zeile der Arrayspalte. Entsprechende Variablen werden auch für die Adressierung in dem Konfigurationsbitstrom angelegt, um eine direkte Zustandsübertragung durchführen zu können.

Die Rekonstruktion erfolgt nun innerhalb der ausgewählten Spalte durch die Verarbeitung aller Registerbits, die sich in dem adressierten 'Minor'-Frame befinden. Aufgrund des Aufbaus eines 'Minor'-Frames (siehe Abbildung 3.14) wird die Bitposition jeweils um 18 Bit erhöht, wodurch die Register in der nächsten Zeile adressiert werden. Dieses Durchlaufen der Spalte erfolgt mit eigenen Laufvariablen. Nach Durchlauf aller vier Register bzw. deren 'Minor'-Frames werden die vier Variablen, die auf die Logikzelle in der ersten Zeile zeigen, um einen 'Major'-Frame weiter gesetzt, um die Rekonstruktion der nächsten Logikzellenspalte durchführen zu können. Diese spaltenweise Rekonstruktion wird solange fortgesetzt, bis die letzte Spalte innerhalb des Logikzellen Arrays vollständig verarbeitet wurde. Der gesamte Ablauf der Funktion, der bei den anderen zu rekonstruierenden Elementen ähnlich ist, ist noch einmal in dem Ablaufdiagramm 3.17 dargestellt.

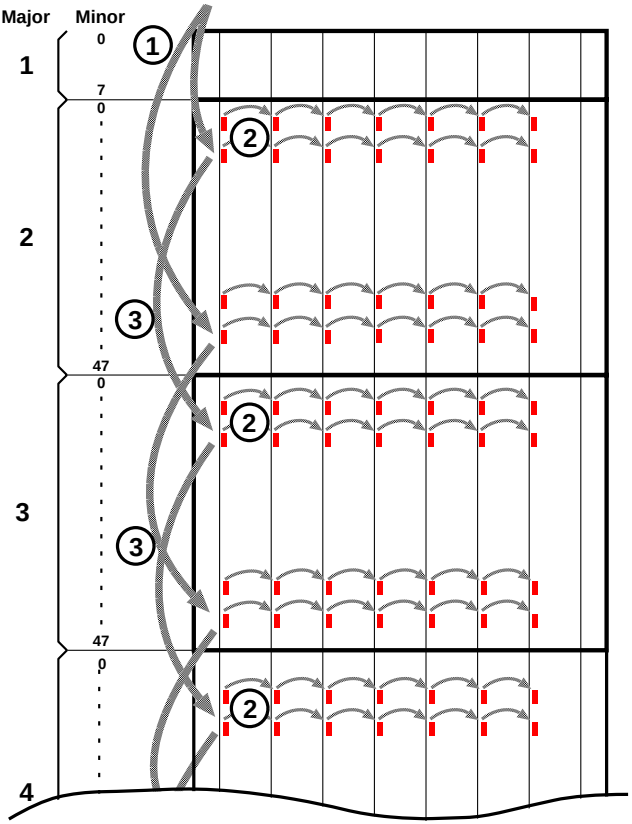


Abbildung 3.16: Relative Adressierung der Bitpositionen innerhalb des Readback und Konfigurationsbitstroms am Beispiel der Logikzellenrekonstruktion.

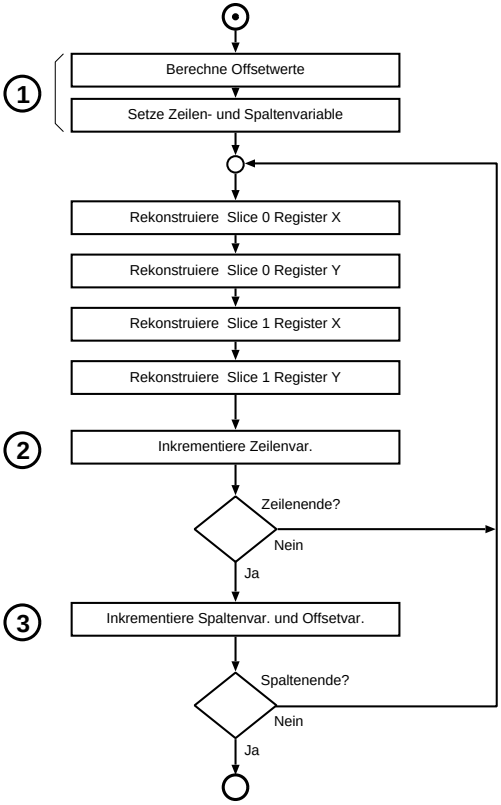


Abbildung 3.17: Ablaufdiagramm der Funktion die den Zustand alle Register in den Logikzellen rekonstruiert.

Die vier Funktionen zur Rekonstruktion des gesamten FPGA-Bausteins basieren bis auf die Block-RAM Funktion auf dem oben angegebenen Prinzip. Mit diesen Funktionen ist das Betriebssystem in der Lage einen beliebigen Baustein der Xilinx Virtex Serie[8] mit all seinen speichernden Elementen vollständig zu rekonstruieren. Diese Rekonstruktion basiert dabei nur auf den im Readbackbitstrom enthaltenen Zustandsdaten und stellt diesen Zustand in dem dazugehörigen Konfigurationsbitstrom zu 100% wieder her.

3.6.2 Messungen

Um die Leistungsfähigkeit der einzelnen Funktionen zu evaluieren, werden die oben beschriebenen vier Funktionen mit weiteren Funktionen, die jeweils die gleiche Funktion ausführen, verglichen. Ziel dieser Messungen ist es, den Funktionsaufbau in Hinblick auf das zweite Ziel – eine minimale Ausführungszeit – hin zu überprüfen.

Die beiden anderen Funktionen unterscheiden sich in deren Aufbau und der Durchführung der Rekonstruktionen. Die Unterschiede werden im nachfolgenden anhand der Logikzellen Register verdeutlicht:

Die erste Funktionsgruppe rekonstruiert alle einzelnen Register nacheinander und errechnet für jede Bitposition die Adresse neu. Die zweite Funktionsgruppe ist stark angelehnt an die Ausführungsreihenfolge der oben beschriebenen Funktionen mit dem Unterschied, daß die einzelnen Registergruppen (S0X, S0Y, usw.) in jeweils einer eigenen Schleife rekonstruiert werden. Die dritte, in Abbildung 3.17 dargestellte, Gruppe verarbeitet alle vier Registergruppen hintereinander, gesteuert von nur zwei ineinander verschachtelten Schleifen.

Die Funktion zur Rekonstruktion der Logikzellen Register wurde für die Messungen in allen drei unterschiedlichen Ausführungen programmiert. Die Rekonstruktion des Block-RAMs hingegen wurde nur in der ersten und der dritten Ausführungsmethode umgesetzt. Alle Messungen erfolgen auf einem Pentium I Rechner mit 166MHz Taktfrequenz und 128MByte RAM. Die Ergebnisse sind in der folgenden Tabelle 3.4 zusammengefaßt.

[8] Dies gilt nicht für die Bausteine der Virtex-E Serie.

Rekonstruktionsfunktion	Gruppe 1	Gruppe 2	Gruppe 3
Logikzellen Register	17.4ms	5.3ms	4.8ms
Block-RAM	136ms	—	0.2ms

Tabelle 3.4: Ausführungszeiten für die unterschiedlich aufgebauten Rekonstruktionsfunktionen für die Logikzellen Register und das Block-RAM.

Die durchgeführten Messungen zeigen deutlich, daß der gewählte Funktionsaufbau und der Rekonstruktionsablauf am besten geeignet ist, um das zweite Ziel einer minimalen Ausführungszeit zu erreichen. Im Vergleich zu dem ersten naiven Ansatz werden bei der Rekonstruktion der Logikzellen Register ≈75% der ursprünglich benötigen Zeit eingespart. Die Rekonstruktion des Block-RAMs erfolgt in weniger als 1% der ursprünglichen Zeit, was auf die Rekonstruktion als zusammenhängender Block zurückzuführen ist. Die minimalen Unterschiede zu der zweiten Funktionsgruppe ergeben sich durch die zusätzlich entstehenden Kontrollstrukturen für die Schleifen.

Diese Ergebnisse sind entsprechend miteingeflossen in die Entwicklung der vier Funktionen zur Rekonstruktion von Logikzellen Register, I/O-Zellen Register, Look-Up Tabellen RAM und Block-RAM. Alle vier Funktionen rekonstruieren den Zustand aus einem gegebenen Readbackbitstrom heraus direkt in den dazu passenden Konfigurationsbitstrom. Die Rekonstruktion umfasst *alle* speichernden Elemente des FPGA-Bausteins wodurch eine 100% Rekonstruktion des Zustands erreicht wird. Die nachfolgende Tabelle 3.5 listet die einzeln benötigten Rekonstruktionszeiten und die insgesamt benötigte Zeit für die Rekonstruktion eines XV400 Bausteins auf.

Alle oben aufgeführten Messungen wurden auf einem PC System mit einem Pentium I Prozessor und 166MHz Systemtakt durchgeführt.

[9] Zeit gemessen für die Sicherung von 8 als RAM verwendete Look-Up Tabellen. Die Ausführungszeit ohne ein RAM beträgt 1.35ms.

ELEMENTE	REKONSTRUKTIONSZEIT
Logikzellen Register	4.8ms
I/O-Zellen Register	0.5ms
Look-Up Tabellen RAMs	1.9ms[9]
Block-RAM	0.2ms
$\sum$	7.4ms

Tabelle 3.5: Ausführungszeiten aller Funktionen zur vollständigen Rekonstruktion des Schaltungszustands.

3.7 Ablauf eines Prozesswechels

Der gesamte Prozesswechsel auf einem FPGA-Koprozessor erfordert neben der zuvor beschriebenen Rekonstruktion des Schaltungszustandes noch weitere Schritte, die ausgeführt werden müssen. Im Betriebssystem, das auf dem Host-Prozessor läuft, wird der Prozesswechsel in vier nacheinanderfolgende Aufgaben unterteilt, die bei jedem Prozesswechsel durchgeführt werden müssen. Diese zum Teil voneinander abhängigen Aufgaben sind:

Kontrolle der Takterzeugung: Das Anhalten des Taktes versetzt die derzeit laufende Schaltung in einen konstanten, nicht weiter veränderten Zustand, der für die nachfolgende Analyse und Rekonstruktion des Zustands notwendig ist. Abgeschaltet wird die Schaltung über die in Abbildung 3.7 dargestellte Taktkontrollschaltung, um zu gewährleisten, daß der FPGA-Koprozessor ohne einen Datenverlust angehalten wird. Jeder Prozesswechsel startet mit dieser Taktabschaltung, um den derzeitigen Zustand zu einem Taktschaltpunkt 'einzufrieren'.

Nach der erfolgten Rekonstruktion und Weiterführung der Anwendung durch das Betriebssystem erfolgt der erneute Start der Taktgenerierung als letzte Aufgabe des Betriebssystems. Durch diese Reaktivierung des Taktes ist der FPGA-Koprozessor in der Lage den Prozess an der zuvor unterbrochenen Stelle des Ablaufs weiter fortzuführen.

Sowohl das Ab- als auch das Einschalten des Taktes erfordern nur wenige einzelne Zugriffe auf den FPGA-Koprozessor und wird in wenigen μs ausgeführt.

Readback und Konfiguration des FPGAs: Nach der erfolgten Abschaltung des Taktgenerators muß der angehaltene Zustand der Schaltung von dem Betriebssystem ausgelesen werden. Zu diesem Zweck wird ein Readback des FPGA-Bausteins initiiert und die gesamte Konfiguration inklusive des derzeitigen Zustands wird in den Speicher des Host-Prozessors übertragen. Dieser zweite Schritt innerhalb eines Prozesswechsel ist notwenig, um die Zustandsdaten für die Rekonstruktion auf dem Host-Prozessor bereitzustellen.

Im umgekehrten Fall – bei der Fortsetzung einer Anwendung – wird der FPGA-Baustein mit dem zuvor rekonstruierten Konfigurationsbitstrom aus dem Speicher des Host-Prozessors konfiguriert und anschließend durch das Einschalten des Taktes wieder gestartet. Auch hier ist eine Übertragung des Konfigurationsbitstroms notwendig, da die Rekonstruktion durch das Betriebssystem auf dem Host-Prozessor erfolgt. Unmittelbar vor der Konfiguration werden die RAM-Inhalte wiederhergestellt, so daß die FPGA-Schaltung durch den erneuten Start des Taktgenerators die Anwendung weiter verarbeitet kann.

Bedingt durch die Größe des jeweiligen Konfigurations- und Readbackbitstroms, die bei dem Xilinx XV400 Baustein aus rund 2.4 MBit besteht, bestimmen diese beiden Aufgaben maßgeblich die gesamte Zeit die zur Durchführung eines Prozesswechsels benötigt wird. Die Konfiguration des XV400 Bausteins erfolgt dabei in 12ms und der Readback in 14ms (siehe Abschnitt 5.6).

Sicherung des RAM Inhalts: Zusätzlich zu dem Speichern in dem FPGA-Baustein verwendet eine Vielzahl von Anwendungen die lokalen RAMs auf den FPGA-Koprozessoren. Bei der Zustandssicherung der Anwendung muß somit neben dem Zustand des FPGA-Bausteins auch der Inhalt dieses RAMs gesichert und entsprechend rekonstruiert werden.

Die Sicherung des jeweiligen Inhalts erfolgt in einem dritten Schritt nach dem Readback des FPGA-Bausteins. Gesichert wird der Inhalt entweder über die noch konfigurierte FPGA-Schaltung oder nach der Konfiguration einer speziellen Schaltung, die einen Zugriff auf das RAM erlaubt. Die bei dieser Sicherung gelesenen Daten werden, wie zuvor der Readbackbitstrom, zum Host-Prozessor übertragen und dort zusammen mit den anderen Prozessinformationen gespeichert.

Die Wiederherstellung der Daten bei einer erneuten Ausführung des Prozesses erfolgt abhängig von der Zugriffsmöglichkeit der auszuführenden FPGA-Schaltung. Besteht eine direkte Zugriffsmöglichkeit des Betriebssystem auf das lokale RAM, so erfolgt die Wiederherstellung des RAM-Inhalts nach der Konfiguration des FPGA-Bausteins, was nur eine Konfiguration des FPGA-Bausteins erfordert. Besteht dieser direkte Zugriff nicht, so muß durch die Konfiguration einer speziellen Schaltung der RAM-Inhalt wiederhergestellt werden, bevor die eigentlich auszuführende FPGA-Schaltung geladen werden kann.

Die zur Sicherung und Wiederherstellung des RAM-Inhalts benötigt Zeit ist von zwei Faktoren abhängig: Der Datenaustauschrate zwischen dem Host- und dem FPGA-Prozessor und der Größe des RAM-Blocks. Auf den FPGA-Koprozessor werden in der Regel Speicherblöcke mit einer Größe von 512KByte bis 8MByte eingesetzt. Bei einer angenommenen Datenübertragungsrate von 120 MByte/s zwischen dem Host- und dem FPGA-Koprozessor werden somit weitere 33ms für Sicherung und Wiederherstellung eines 2MByte großen RAMs benötigt. Diese Zeit erhöht sich auf $\approx$45ms sobald eine spezielle FPGA-Schaltung konfiguriert werden muß um den Zugriff auf das RAM zu ermöglichen.

Rekonstruktion des Prozesszustands: Das Unterbrechen einer laufenden Anwendung wird mit einem vierten und letzten Schritt beendet. In diesem wird unter Verwendung der in Abschnitt 3.6 beschriebenen Rekonstruktionsfunktionen eine Analyse des unterbrochenen Schaltungszustands durchgeführt.

Mit Abschluß dieses vierten Schrittes ist die Verarbeitung der entfernten Anwendung abgeschlossen und das Betriebssystem bereitet die neu zu startende Anwendung auf dessen Ausführung vor. Der erste Schritt dieser Vorbereitung ist die Rekonstruktion des Zustands in dem entsprechenden Konfigurationsbitstrom, bevor dieser im nachfolgenden auf den FPGA-Baustein konfiguriert wird. Bedingt durch den effektiven Aufbau der Rekonstruktionsfunktionen wird diese Vorbereitung zusammen mit der Analyse durchgeführt. Somit steht der modifizierte Konfigurationsbitstrom der neuen Anwendung schon zur Verfügung und muß beim nächsten Prozesswechsel nur noch konfiguriert werden.

Die Rekonstruktion des Schaltungszustands wird für den XV400 Baustein in ≈5ms durchgeführt. Dennoch wird die Zeit für einen Prozesswechsel dadurch nicht erhöht, da diese Rekonstruktion parallel zu der Konfiguration des FPGA-Bausteins erfolgt. Ermöglicht wird dies durch den Einsatz einer DMA-Übertragung der Konfigurationsdaten auf dem FPGA-Koprozessor.

Eine Aufgabe des Betriebssystem ist es, den Prozesswechsel auf dem FPGA-Koprozessor zu steuern und die oben genannten Aufgaben nacheinander durchzuführen. Der gesamte Ablauf des Prozesswechsels auf dem FPGA-Koprozessor, dargestellt in Abbildung 3.18, ist ähnlich zu dem Prozesswechsel eines Intel Pentium Prozessors.

Der erste Schritt schaltet die Taktgenerierung ab, um den Zustand der Anwendung 'einzufrieren'. In dem zweiten Schritt wird dann der interne Zustand des FPGA-Bausteins per Readback ausgelesen und zum Host-Prozessor übertragen. Nachfolgend wird auch der Inhalt der lokalen RAMs auf dem FPGA-Koprozessor vom Betriebssystem gesichert. Der vierte Schritt analysiert den Schaltungszustand und rekonstruiert diesen im entsprechenden Konfigurationsbitstrom.

Nach der abgeschlossenen Unterbrechung der einen Anwendung wird die neu auszuführende Anwendung vorbereitet. Dazu erfolgt in Schritt fünf die Rekonstruktion des RAM-Inhalts. Diese Rekonstruktion erfolgt entweder vor oder nach der Konfiguration des FPGA-Bausteins. Die Konfiguration des FPGAs ist der sechste Schritt. Der siebte Schritt bei

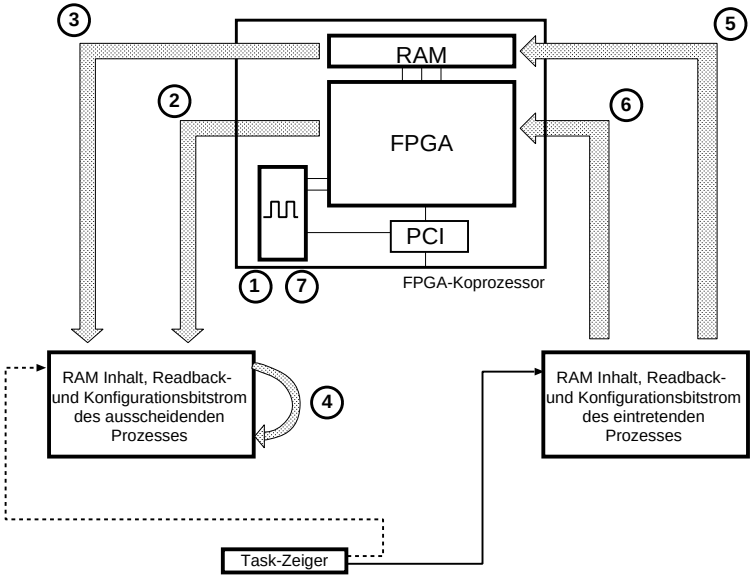


Abbildung 3.18: Darstellung der einzelnen Schritte des Prozesswechsels auf einem FPGA-Koprozessor.

einem Prozesswechel ist die Reaktivierung des Taktes die zur weiteren Abarbeitung der Schaltung startet.

Dieser gesamte Ablauf eines Prozesswechsels erfolgt innerhalb von ≈80ms auf einem Pentium I 166MHz unter Verwendung eines FPGA-Koprozessor mit einem XV400 Baustein (Details der Messungen können in Abschnitt 5.6.3.2 auf Seite 173 nachgelesen werden). Bei dieser gemessen Zeit für einen Prozesswechsel erfolgt *keine* Sicherung des lokale RAMs auf dem FPGA-Koprozessor. Eine solche Sicherung würde die Prozesswechselzeit in besten Fall um weitere 33ms erhöhen.

3.8 Zusammenfassung

In Kapitel 3 wurden die existierenden Betriebssysteme für FPGA-Koprozessor im Detail beschrieben. Es hat sich gezeigt, daß diese Entwicklungen eine Vielzahl von Einschränkungen hinsichtlich der zum Einsatz kommenden FPGA-Baustein, der möglichen Anwendungen und deren Ausführung besitzen. Fast allen Ansätzen gemeinsam ist die Verwendung der sog. 'Overlay'-Technik die nur eine sequenzielle Abarbeitung der einzelnen Anwendungen zulässt. In Unterschied dazu untersucht der neue, in dieser Arbeit beschriebene Ansatz einen preemptiven Prozessablauf zum Aufbau eines Multitasking Betriebssystem bei dem mehrere Prozesse gleichzeitig verarbeitet werden.

Wichtigste Eigenschaft eines solchen preemptiven Multitasking Betriebssystemes ist die Möglichkeit, einen laufenden Prozess in dessen Ausführung zu unterbrechen, ihn zu sichern und zu einem späteren Zeitpunkt wieder auf dem FPGA-Koprozessor weiter auszuführen. Diese preemptiven Prozesswechsel erfordern unterschiedliche Anforderungen sowohl an den FPGA-Baustein und den FPGA-Koprozessor, aber auch an die Entwicklung der Schaltungen und das Betriebssystem. Diese geforderten Anforderungen sind in dem Kapitel detailliert aufgelistet und eine Vorauswahl der in Frage kommenden FPGA-Bausteine und FPGA-Koprozessoren wurde vorgenommen.

Die zentralen Eigenschaften, die ein gefordertes preemptives Multitasking ermöglicht, ist die Analyse und die Rekonstruktion des Schaltungszustandes. Diese Schaltungsrekonstruktion wird anhand des Beispielbausteins Xilinx Virtex XV400 in der grundsätzlichen Wirkungsweise, dem Ablauf und der Umsetzung näher erläutert. Zur Evaluation der zu erwartenden Zeit für einen Prozesswechsel wurden verschiedene Messungen durchgeführt.

Zusammenfassend wurde in diesem Kapitel dargestellt, daß das Unterbrechen und Wiederherstellen einer FPGA-Schaltung unter bestimmten Voraussetzungen möglich ist und durchgeführt werden kann. Daraus entstanden sind vier Funktionen, die diese Rekonstruktionsmöglichkeit für eine gesamte Bausteinfamilie von modernen FPGA-Bausteinen umsetzen. Basierend auf diesen Rekonstruktionsfunktionen wird in den bei-

den nachfolgenden Kapiteln die Konzeption und der Aufbau des Multitasking Betriebssystem dargestellt.

Kapitel 4

Betriebssystem Architektur

Das vorherige Kapitel hat gezeigt, daß die Durchführung eines preemptiven Multitasking auf FPGA-Koprozessoren prinzipiell durchführbar ist. Neben der Frage der generellen Machbarkeit muß aber auch der Nutzen, der durch ein solches Multitasking Betriebssystem erreicht werden soll, im Verhältnis zu dem zusätzlich entstehenden Overhead beim Einsatz dieses Betriessystems stehen.

Zur Bestimmung des Nutzens werden in diesem Kapitel verschiedene Anwendungen aus einzelnen wissenschaftlichen Bereichen aber auch allgemeine Anwendungen, die durch einen FPGA-Koprozessor beschleunigt ausgeführt werden, analysiert. Basierend auf den Aufgaben eines Betriebssystems, den ermittelten Ergebnissen und den sich daraus ergebenden Anforderungen werden anschließend drei grundlegende Konzepte für den Aufbau eines Multitasking Betriebssystem erörtert und diskutiert.

4.1 Anwendungsanalyse

Der Konzeption eines geeigneten Betriebssystems geht eine Analyse der derzeitig und zukünftig durch das Betriebssystem ausgeführten Anwendungen voraus. Ziel der Analyse ist zum einen die Erstellung einer allgemeinen Programmierschnittstelle (API) für die Anwendungen und zum anderen die Bestimmung des Nutzens eines preemptiven Multitasking Betriebssystem für diese Anwendungen im Speziellen und für FPGA-Koprozessor im allgemeinen.

Zur Erstellung einer allgemeinen API werden die einzelnen Host-Programme der Anwendungen untersucht. Jede Anwendung auf einem FPGA-Koprozessor wird durch ein entsprechendes Host-Programm gesteuert. Dieses übernimmt Steuerungsaufgaben wie z.B. das Konfigurieren des FPGA-Bausteins oder die Einstellung der Taktraten, aber auch den Datentransfer zwischen dem FPGA-Koprozessor und dem Host-Speicher. Aus allen untersuchten Programmen werden die für die Ausführung notwendigen Funktionen in funktionale Gruppen eingeteilt und auf eine allgemeine API-Schnittstelle abgebildet. Dieses API bildet dann die Schnittstelle zwischen dem Host-Steuerungsprogramm und den von den jeweiligen Anwendungen eingesetzten FPGA-Koprozessoren und ermöglichen eine einheitliche Kontrolle der verschiedensten FPGA-Koprozessor über die gleichen Funktionsaufrufe. Aufgrund des geringen Einfluß dieser API-Schnittstelle auf die Konzeption des Betriebssystems wird diese nicht weiter erläutert.

Das zweite Ziel der Anwendungsanalyse – die Untersuchung des Nutzens eines preemptiven Betriebssystems – ist für diese Arbeit weitaus wichtiger, denn nur wenn der Einfluß des Betriebssystem auf die auszuführende Anwendungen klein ist und die Vorteile durch eine einfachere Ausführung und Entwicklung von Anwendungen für den FPGA-Koprozessor erreicht wird, ist ein Einsatz eines solchen Betriebssystem zu rechtfertigen. Zur Untersuchung dieses zweiten Ziels werden die Anwendungen anhand der nachfolgenden Kriterien analysiert und bewertet:

Ausführungszeit auf dem FPGA-Koprozessor: Die Ausführungszeit ist die Zeit, in der die Anwendung Daten auf dem FPGA-Koprozessor verarbeitet um das Ergebnis zu ermitteln. Diese Aus-

führungszeit ist beim Einsatz eines Multitasking Betriebssystems im besonderen von der Wahl der Zeitscheibenlänge, in der die Anwendungen auf dem FPGA-Koprozessor ausgeführt werden, abhängig. Abbildung 4.1 illustriert diesen Zusammenhang graphisch. Bei einer zu kurzen Zeitscheibenlänge entstehen viele Unterbre-

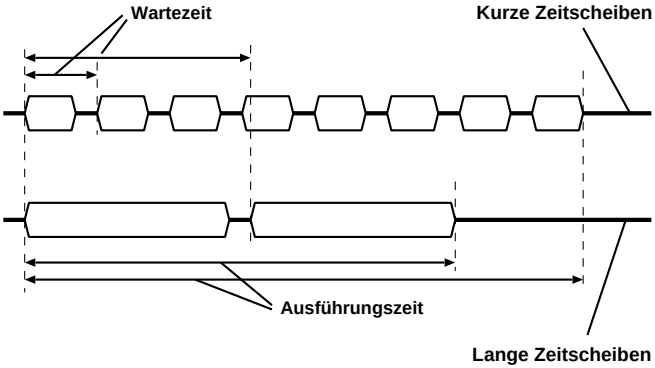


Abbildung 4.1: Warte- und Ausführungszeit in Abhängigkeit von der Zeitscheibenlänge. *Das obere Bild zeigt die Verarbeitung bei kurzen Zeitscheiben und das untere bei langen Zeitscheiben.*

chungen, die die gesamte Ausführungszeit erhöht. Die Wahl einer zu langen Zeitscheibe erhöht jedoch die Wartezeit auf den FPGA-Koprozessor, was im ungünstigsten Fall ebenfalls zur Verlängerung der Ausführungszeiten führen kann.

Darüberhinaus bestimmt das Verhältnis der Ausführungszeiten zu der Zeitscheibenlänge über die Notwendigkeit einer Zustandsrekonstruktion. Ist dieses Verhältniss stets kleiner als eins, dann ist die Rekonstruktion nicht notwendig denn das Ergebnis liegt immer vor Ablauf der Zeitscheibe vor. In diesem Fall muß der Zustand nicht gesichert werden, denn die folgende Berechnung erfolgt mit einem neuen Datensatz und einer FPGA-Schaltung die im Grundzustand gestartet wird. Ist das Verhältniss jedoch größer als eins so überschreitet die Ausführungszeit die zur Verfügung stehende

Zeitscheibe und der Zustand muß am Ende der Zeitscheibe gesichert werden, damit die Anwendung zu einem späteren Zeitpunkte weiter verarbeitet werden kann.

Wie oben gezeigt wurde, beeinflußt die Ausführungszeit der Anwendungen die Länge der gewählten Zeitscheibe und die Notwendigkeit der Zustandsrekonstuktion. Umgekehrt hat aber auch die Zeitscheibenlänge zusammen mit dem systematischen Overhead des Betriebssystems einen Einfluß auf die Ausführungszeit. Zur Bestimmung der Zeitscheibenlänge ist es daher sehr wichtig die Ausführungszeiten der einzelnen Anwendungen zu kennen.

Datenaustausch zwischen Host- und FPGA-Koprozessor: Der Datenaustausch ist ein weiterer Punkt, der bei der Analyse und der Beurteilung hinsichtlich eines Multitasking Betriebssystem eine wichtige Rolle spielt. Bei der Betrachtung des Datenaustausches wird grundsätzlich in zwei Gruppen von Anwendungen unterschieden: Datenflußanwendungen und rechenintensive Anwendungen. Bei den Datenflußanwendungen werden die Daten direkt während der Übertragung verarbeitet wodurch die übertragene Datenmenge die Ausführungszeit bestimmt. Die rechenintensiven Anwendungen hingegen übertragen nur wenig Daten, wobei dort die Ausführungszeit nicht von der Übertragung sondern von den Berechnungen bestimmt ist.

Gerade bei den Datenflußanwendungen ist aufgrund der direkten Datenverarbeitung die Ausführungszeit abhängig von der zu erreichenden Datentransferrate und der Datenmenge. Wie die Messungen in Abschnitt 5.6.2 noch zeigen, ist die Datenrate ebenfalls abhängig von der Datenmenge. Somit ist für Datenflußanwendungen die zu übertragende Datenmenge bzw. Datentransferzeit ebenfalls wichtig zur Bestimmung der Zeitscheibenlängen. Die Datenübertragung bei rechenintensiven Anwendungen ist nur sekundär interessant, da die Ausführungszeit durch die Rechenzeit bestimmt wird.

Der Datenaustausch und sein Einfluß auf die Wahl der Zeitscheiben ist somit ein weiteres Kriterium bei der Analyse der Anwendungen.

Darüberhinaus ermöglicht die Betrachtung des Datentransfers den Aufbau eines neuen Konzepts zur effektiveren überschneidenen Anwendungsverarbeitung.

Abstraktionsebene der FPGA-Koprozessor Anwendung: Die Angabe der verwendeten Abstraktionsebene ist das dritte wichtige Kriterium für die Notwendigkeit eines preemptiven Multitasking Betriebssystems. Anwendungen aus der Gruppe der Maschinenbefehle setzen voraus, daß die Funktionalität bei Ausführung der Anwendung auf dem Host-Prozessor verfügbar ist, um den Befehl ohne große Verzögerungen ausführen zu können. Dies erfordert jedoch den Wechsel der FPGA-Schaltung mit jedem Prozesswechsel auf dem Host-Prozessor, was zu einem massiven Overhead durch das Betriebssystem führt. Anwendungen die auf funktionaler- bzw. Programmebene arbeiten, können im Vergleich dazu unabhängig von Host-Prozessor verarbeitet werden, wodurch eine Prozessverarbeitung mit getrennten Prozessorzeitscheiben ermöglicht wird.

Unter Berücksichtigung des entstehenden Overheads bei häufigen Prozesswechseln ist die Abstraktionsebene ein weiteres wichtiges Kriterium bei der Analyse der Anwendungen. Zusätzlich erhält man über diese Analyse einen Eindruck über die Einsatzfähigkeit der bisher entwickelten Betriebssysteme.

4.1.1 Analyse

Unter Berücksichtigung der Ziele und der oben genannten Kriterien wurden unterschiedliche FPGA-Koprozessor Anwendungen untersucht. Die ausgewählten Anwendungen stammen dabei aus den verschiedensten Gebieten, in denen die FPGA-Koprozessoren Einsatz finden und sie werden auf unterschiedlichen FPGA-Koprozessoren ausgeführt. Dadurch ergibt sich zum einen ein guter Überblick über die einzelnen FPGA-Koprozessoren und deren Programmierschnittstelle und zum anderen ein allgemeingültiger Überblick bezüglich der oben genannten Kriterien.

Untersucht wurden ausschließlich Anwendungen die eine Datenverarbeitung auf Funktions- oder Programm-Ebene durchführen. Anwendungen der Maschinenbefehls-Ebene wurden nicht analysiert, da sie zum

einen den Overhead der Anwendung unverhältnismäßig erhöhen und zum anderen aufgrund der niedrigen Datenaustauschrate heutzutage nicht mehr zum Einsatz kommen.

4.1.1.1 Anwendungen

Die untersuchten Anwendungen werden für die Analyse in die beiden Gruppen der datenflußorientierten und der rechenintensiven Anwendungen unterteilt. Nachfolgend werden die einzelnen Anwendungen kurz erläutert und in der Tabelle 4.1 werden die analysierten Werte zusammengefaßt.

Datenflußanwendungen:

HEP-Mustererkennung [MSS00, KLL+98]: In der Hochenergiephysik verarbeiten die FPGA-Koprozessoren Detektordaten auf der Suche nach seltenen Partikelteilchen, die für die spätere Analyse abzuspeichern sind. Die Detektorbilder werden während der Übertragung mit Mustererkennungsalgorithmen verarbeitet und die Ergebnisse anschließend ausgelesen.

DES Ver-/Entschlüsselung [Pat00]: Verschlüsseln von Daten durch den DES Algorithmus, um z.B. die Übertragung von Daten gegen ungewünschtes Abhören zu sichern. Auch hier werden die Daten zeitgleich zu der Übertragung verschlüsselt.

Photoshop Filter [SS98]: Die Filterung von 2-dimensionalen Bilddaten wird von einem FPGA-Koprozessor durchgführt. Durch die Flexibilität des FPGA-Koprozessors können verschiedene Filter auf einem System ausgeführt werden. Angebunden ist der FPGA-Koprozessor an das Programm Photoshop.

Genomdatenbankanalyse [LM95]: Der Homologievergleich einzelner Gensequenzen mit den Datenbankeinträgen ist die Aufgabe die durch einen FPGA-Koprozessor ausgeführt wird. Die Vergleiche werden durchgeführt während die Daten an den FPGA-Baustein übertragen werden.

DTP Programm [MS98]: PostScript Rendering ist ein sehr rechenaufwendiger Schritt der die Postscript-Seitenbeschreibung in ein pixelorientiertes Bild umwandelt. In einer Studie werden Bézier Kurven mit Hilfe eines FPGA-Koprozessor in ein Pixelbild transformiert.

Verkehrszeichenerkennung [Hez]: Positionsbestimmung von Verkehrszeichen mit einem speziellen Mustererkennungsalgorithmus der auf Bilddaten basiert. Das Ziel ist die Positionsbestimmung für eine weiterführende Analyse. Die vielen Vergleiche werden dabei auf den übertragenden Bilddaten durchgeführt.

Volumenvisualisierung [VHM+99]: Visualisieren eines 3-dimensionalen Datensatzes ist eine weitere datenintensive Anwendung die auf einem FPGA-Koprozessor ausgeführt wird. Die generierten Bilddaten werden online erzeugt und sofort übertragen.

Rechenintensive Anwendungen:

Image Interpolation [HLA98]: Diese Anwendung aus dem Bereich der Bildverarbeitung interpoliert die Bilddaten zwischen den benachbarten Pixeln und vergrößert somit die Bildfläche. Die Originaldaten werden nach der Übertragung verarbeitet und anschließend zurückgelesen.

Mustererkennung [RH97]: Das Erkennen von verschiedenen Objekten in einem Radarbild ist Ziel dieser Mustererkennung. Die interessanten Bereiche werden von der Anwendung mit mehreren Gruppen von Mustern verglichen und das Ergebnis zurückgegeben. Auch hier ist der Rechenaufwand aufgrund der vielen Muster 85mal höher als für die Übertragung der Radarbilder.

Stereobildanalyse [WV97]: Die Anwendung erstellt basierend auf zwei Bildern eine Tiefenkarte der aufgenommenen Szenerie. Der Vergleich jedes einzelnen Pixels des einen Bildes mit einer Vielzahl von Pixeln in dem anderen Bild verdeutlicht den hohen Rechenaufwand dieser Anwendung.

Genetische Algorithmen [GN96]: Das FPGA-Koprozessor System ermittelt die optimale Lösung des Traveling-Salesman Problems unter Verwendung eines genetischen Algorithmus. Auch dieses Anwendung ist aufgrund der vielen Schleifendurchläufe sehr rechenaufwendig.

Optimierung [AFA+99]: Die Optimierung von Schnittmustern für die Textilindustrie ist ein weiterer rechenintensive Prozess der auf einem FPGA-Koprozessor ausgeführt wird. Ziel der Optimierung ist eine Minimierung des Verschnitts zu erreichen.

Proteinstrukturvorhersage [SBM00]: Gesucht wird hier eine Proteinstruktur mit der minimalen Energie. Die Energieberechnung ist ein rechenintensiver Teil, der auf dem FPGA-Koprozessor ausgeführt wird.

Eine weitere Umsetzung berechnet sowohl den Energieterm als auch die Proteinstruktur und durchläuft viele Schleifen bei der Berechnung der optimalen Proteinstruktur.

DES Keybreak [KD98]: Die Suche nach einem Schlüssel durch einfaches Probieren ist ein sehr rechenintensiver Prozess, der nahezu keinen Datentransfer erfordert. Die Analyse des gesamten Suchraums auf einem FPGA-Koprozessor erfordert mehrere Jahre Rechenzeit.

4.1.1.2 Anwendungsanalyse

Wie in der Tabelle 4.1 zu sehen ist, erstrecken sich die Ausführungszeiten der einzelnen Anwendungen von wenigen Millisekunden bis zu mehreren Jahren, wobei die durchschnittliche Ausführungszeit im Bereich von mehreren 100ms liegt.

Die Betrachtung eines Beispiels macht die Notwendigkeit für ein preemptives Multitasking Betriebssystem deutlich. Angenommen ein Benutzer hat die Berechnung der Traveling-Salesman Problem gestartet und kurz darauf startet ein anderer Benutzer die Berechnung eines Tiefenbildes mittels der Stereobildanalyse. In einem nichtpreemptiven Betriebssystem wird die zweite Anwendung erst ausgeführt wenn die erste beendet ist und das bedeutet, daß sich die Ausführungszeit aus Sicht des

	Anwendung	Datentransfer Bytes	Zeit	Ausführungs Zeit
1	HEP-Mustererkennung	64kByte	0.5ms	1.2ms
2	DES	2MByte	16ms	17.5ms
3	Verkehrszeichen-erkennung	105kByte	15ms	45ms
4	Volumenvisualisierung	1MByte	105ms	105ms
5	DTP	1MByte	250ms	500ms
6	Photoshop Filter	70MByte	583ms	714ms
7	Genomdatenanalyse	800MByte	6.6sek	90sek
8	Stereobildanalyse	77kByte	0.64ms	24ms
9	Image Interpolation	272kByte	2.2ms	188ms
10	Mustererkennung	16kByte	130μs	244ms
11	Genetische Algorithmen	480Byte	10μs	295sek
12	Optimierung	$\approx$100KByte	1ms	3360sek
13	ProteinStruktur (1)	37KByte	300μs	40ms
14	ProteinStruktur (2)	$\approx$100KByte	1ms	>86400sek
15	DES Keybreak	100Byte	<1ms	$67.8 * 10^9$sek

Tabelle 4.1: Ergebnisse der Anwendungsanalyse. *Angegeben sind jeweils die übertragene Datenmenge und die dafür benötigte Zeit und die Zeit für die Ausführung der gesamten Anwendung.*

Benutzers für die Tiefenbildberechnung im ungünstigsten Falle von 24ms auf über 295 Sekungen verlängert. Eine solche dramatische Verlängerung der Ausführungszeit um mehrere 1000% ist für einen praktischen Betrieb nicht vertretbar. Demgegenüber besteht in einem preemptiven Betriebssystem die Möglichkeit, die Ausführung der ersten Anwendung zu unterbrechen, die zweite Anwendung (Stereobildanalyse) zu verarbeiten und daran anschließend mit der ersten Anwendung fortzufahren. Im Ergebnis verlängert sich die Ausführungszeit der zweiten Anwendung nur minimal um die Zeit für einen Prozesswechsel und die gesamte Ausführungszeit für beide Anwendungen zusammen wird ebenfalls nur minimal durch den Overhead der Prozesswechsel verlängert.

Bei der Wahl der Zeitscheibe spielt die durchschnittliche Ausführungszeit eine wichtige Rolle. Eine Zeitscheiben die kürzer ist als die durchschnittliche Ausführungszeit bedeutet, daß sehr viele Anwendungen mindestens einmal unterbrochen werden und sich somit deren Ausführungszeit immer verlängert. Demgegenüber ist eine zu große Zeitscheibe gleichbedeutend mit einer langen Wartezeit für kurze Anwendungen. Aus dieser Überlegung ergibt sich die minimale Zeitscheibenlänge aus der durchschnittlichen Ausführungszeit der Anwendungen die auf dem FPGA-Koprozessor verarbeitet werden.

Wie in der zweiten Spalte der Tabelle 4.1 zu sehen ist werden vor allem bei den Datenflußanwendungen zum Teil sehr große Datenmengen zwischen dem Host-Speicher und dem FPGA-Koprozessor übertragen. Der effektive Datentransfer wird durch einen entsprechenden DMA Transfer durchgeführt, der nur bedingt für eine Unterbrechung geeignet ist. Um nun aber die Zeit, in der die Anwendung auf dem FPGA-Koprozessor verarbeitet wird, nicht durch einen sehr langen und nicht unterbrochenen Datentransfer über die gegebene Zeitscheiben hinaus zu verlängern werden große Datenmengen in mehrere kleinere Datentransfers unterteilt. Die Datenmenge der Teiltransfers orientiert sich dabei sowohl an der gewählten Zeitscheibenlänge als auch an der zur Verfügung stehenden Datentransferrate des jeweiligen FPGA-Koprozessors. Voraussetzung zur Unterteilung der Datenmenge ist auch hier die Möglichkeit der Prozessrekonstruktion auf dem FPGA-Baustein.

Ein weiterer Punkt, der nicht mit in die Tabelle aufgenommen wurde, ist die Nutzung von externen Elementen, die auf dem jeweiligen FPGA-Koprozessor vorhanden sind. Für diese Analyse steht dabei die Verwendung der lokalen RAM-Bausteine durch die entsprechenden Anwendungen in Vordergrund.

Mit Ausnahme der beiden Datenflußanwendungen Genomdatenbankanalyse und DES Ver-/Entschlüsselung und der Anwendung der DES-Schlüsselsuche verwenden alle aufgelisteten Anwendungen die lokalen RAM-Elemente, die von dem FPGA-Koprozessor bereitgestellt werden. Diese häufige Verwendung macht klar, daß zusätzlich zu dem Zustand des FPGA-Bausteins auch diese lokalen RAM-Elemente vom Betriebssystem zu sichern sind

Die Auswahl der analysierten Anwendungen zeigt einen representativen Querschnitt aller Anwendungen die auf den FPGA-Koprozessor verarbeitet werden. Bei einer genaueren Betrachtung wird deutlich, daß sowohl sehr spezielle Anwendungen aus einzelnen Wissenschaftsbereichen (Physik, Bioinformatik) aber auch viele allgemeine Anwendungen (Bildbe-/Bildverarbeitung, Datenverschlüsselung, Mustererkennung und Visualisierung) durch den FPGA-Koprozessor in ihrer Ausführung beschleunigt werden.

Eine Bereitstellung der allgemeinen Anwendungen an alle Benutzer eines Rechnersystems erfordert eine entsprechende Koordinierung der Ausführung und einen Schutz der einzelnen Anwendungen gegen eine Datenmanipulation von aussen. Eine solche Bereitstellung der Anwendungen an alle Benutzer eines Systems ist zwangsläufig mit der Einführung eines Betriebssystemes für den FPGA-Koprozessor verbunden. Dieses Betriebssystem koodiniert die Abläufe und den Schutz des jeweiligen Anwendungen und gewährleistet eine störungs- und konfliktfreie Ausführung der einzelnen Anwendungen.

4.1.1.3 Zusammenfassung

Alle untersuchten Anwendungen sind noch einmal übersichtlich in dem Diagramm 4.2 eingetragen. Die durchgeführte Anwendungsanalyse zeigt deutlich, daß eine Ausführung von mehreren Anwendungen auf dem FPGA-Koprozessor nur dann sinnvoll ist, wenn das Betriebssystem die einzelnen Anwendungen in ihrer Ausführung verdrängen kann, denn nur ein solches preemptives Multitasking Betriebssystem ist in der Lage, ein Blockieren einzelner Anwendungen zu verhindern. Die Anwendungsausführung auf dem FPGA-Koprozessor wird dazu in einzelne Zeitscheiben unterteilt, die den Anwendungen zugeordnet werden. Die minimale Länge dieser Zeitscheiben richtet sich nach der durchschnittlichen Ausführungszeit der Anwendungen, die bei mehreren 100ms liegt. Die maximale Zeitscheibenlänge bestimmt die Wartezeit bis zur Ausführung und ist nicht fest definiert. Diese Unterteilung in Zeitscheiben hat weiterhin Auswirkungen auf die Datenübertragung die bei großen Datenmengen unterteilt werden muß. Eine weitere Erkenntnis der Analyse zeigt, daß das lokale RAM auf dem FPGA-Koprozessor bei jedem Prozess-

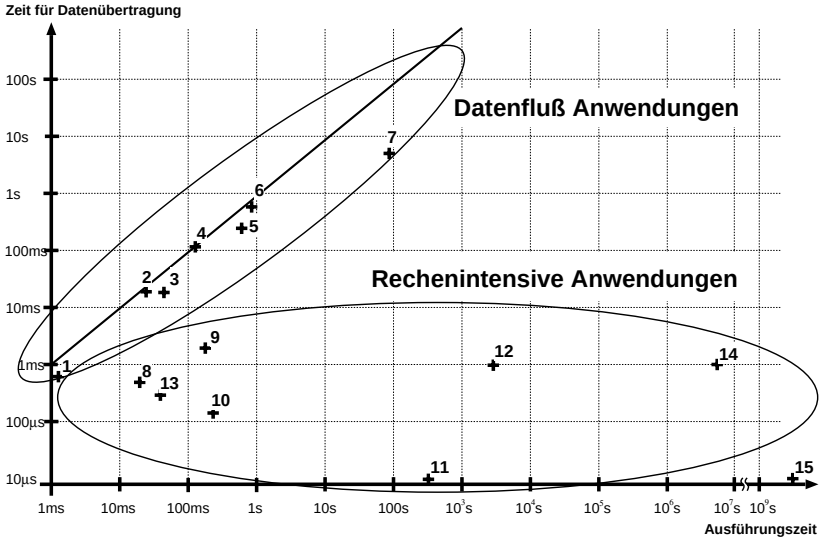


Abbildung 4.2: In dem Übersichtsdiagramm sind die einzelnen Anwendungen mit der jeweiligen Ausführungs- und Datenübertragungszeit dargestellt. *Die Anwendungen sind unterteilt in Datenfluß- und rechenintensive Anwendungen. Die einzelnen Nummern entsprechen den Nummern in der Tabelle 4.1.*

wechsel mit gesichert und rekonstruiert werden muß, da es von fast allen Anwendungen genutzt wird.

Diese Erkenntnisse zeigen, daß die Ausführung mehrerer unabhängiger Anwendungen auf dem FPGA-Koprozessor nur durch ein entsprechendes Betriebssystem sicher und störungsfrei durchgeführt werden kann.

4.2 Betriebssystem Anforderungen

Aus der durchgeführten Anwendungsanalyse ergeben sich verschiedene Anforderungen die bei der Konzeption des Betriebssystems beachtet werden müssen. Diese Anforderungen folgen zum einen direkt aus der Analyse der Anwendungen und zum anderen aus den in Abschnitt 3.4.4 formulierten Aufgaben des Betriebssystems. Im einzelnen sind das: eine hohe Datentransferrate, eine allgemeine Verwendbarkeit des FPGA-Koprozessors, eine gerechte Ausführungsreihenfolge der Anwendungen, eine parallele Nutzung des Host-Prozessors und des FPGA-Koprozessors, eine einheitliche API-Schnittstelle und die Sicherung des gesamten Anwendungszustandes bei einem Prozesswechsel. In der nachfolgenden Auflistung sind die einzelnen Punkte beschrieben:

- Die Datentransferrate zwischen dem Host- und dem FPGA-Koprozessor sollte nahe der theoretischen Datenrate sein, um eine optimale Anbindung der Datenflußanwendungen sicherzustellen und darüberhinaus die Zeit für einen Prozesswechels zu minimieren. Beeinflußt wird diese Datenrate maßgeblich von der Umsetzung des Gerätetreibers.

- Eine Verwendung des FPGA-Koprozessor soll von allen Benutzern und Anwendungen des Systems möglich sein und die einzelnen Anwendungen sollten sich nicht gegenseitig in ihrer Ausführung beeinflussen. Diese Prozesskontrolle und Koordinierung der einzelnen Anwendungen ist zentrale Aufgabe des Betriebssystems selbst.

- Eine weitere Aufgabe des Betriebssystems ist die faire Verteilung des FPGA-Koprozessors unter den konkurrierenden Anwendungen. Durch eine prioritätsbasierte Verteilung der Zeitscheiben soll diese gerechte Aufteilung gewährleistet werden.

- Eine einheitliche API-Schnittstelle soll die Verwendung des Betriebssystems vereinfachen und eine Unabhängigkeit bezüglich der verwendeten FPGA-Koprozessoren gewährleisten. Das API bildet dabei die Schnittstelle zwischen den einzelnen Anwendungen und dem Betriebssystem.

- Die Sicherung und anschließende Rekonstruktion des gesamten Anwendungszustandes, inklusive aller Register und RAM-Inhalte, ist eine weitere Aufgabe die während jedem Prozesswechsel vom Betriebssystem durchgeführt werden muß.
- Die letzte Anforderung an das Konzept des Betriebssystems ist die parallele Nutzung des Host- und des FPGA-Koprozessors. Dies erfordert eine entkoppelte Ausführung der Anwendungen auf beiden Prozessoren. Zusammen mit dieser Entkopplung ist auch der Einfluß des FPGA-Betriebssystems auf das Host-Betriebssystem und die dort ausgeführten Anwendungen so gering wie möglich zu halten.

Diese genannten Anforderungen sind bei der Konzeption des Betriebssystems für die FPGA-Koprozessoren zu beachten, um eine allgemeine Verwendbarkeit und ein Höchstmaß an Leistungsfähigkeit und zu erreichen.

4.3 Grundlegender Aufbau

Der grundlegende Aufbau des Betriebssystems für den FPGA-Koprozessor wird bestimmt durch die zuvor genannten Anforderungen und durch den zur Verwendung kommenden FPGA-Koprozessor selbst.

Die FPGA-Koprozessoren sind in der Regel als eine Einsteckkarte für einen der standardisierten Bussysteme aufgebaut und daher leicht in ein vorhandenes Rechnersystem zu integrieren. Aufgrund der geforderten hohen Datenaustauschrate ist ein entsprechend leistungsfähiges Bussystem auszuwählen. Die heutzutage weit verbreiteten PCI und CPCI Bussysteme stellen eine theoretische Datenrate von 132MByte/s bereit und sind somit für den Aufbau des FPGA-Betriebssystems gut geeignet. Der PCI-Bus wird daher auch bei ≈58% aller FPGA-Koprozessoren verwenden Die Anbindung an das Betriebssystem erfolgt über einen entsprechenden Gerätetreiber und eine spezielle FPGA-Koprozessor abhängige Software-Bibliothek. Der allgemeine Zugriff der Anwendungen auf den FPGA-Koprozessor wird über die einheitliche API-Schnittstelle verwirklicht.

Der Aufbau des gesamten Betriebssystem für FPGA-Koprozessoren ist somit unterteilt in mehrere Schichten, die aufeinander aufbauen und in einzelne Funktionsblöcke unterteilt sind. Das gesamte Schichtenmodell ist in Abbildung 4.3 dargestellt. In dem untersten Funktionsblock

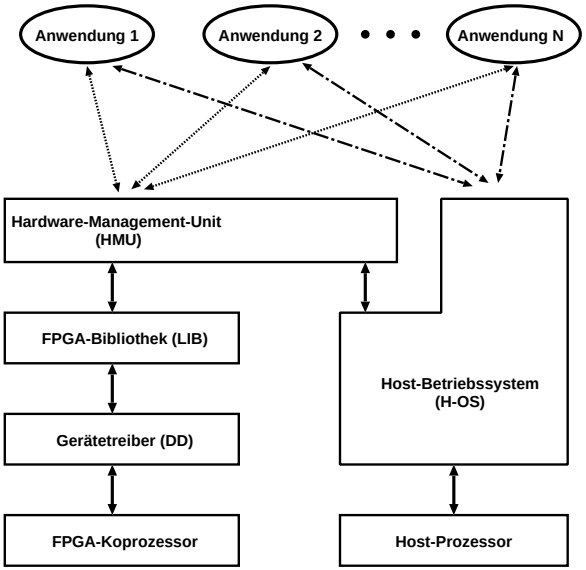


Abbildung 4.3: Allgemeines Schichtenmodell des Betriebssystems für FPGA-Koprozessoren.

des Modells sind die beiden parallel arbeitenden Prozessoren – FPGA-Koprozessor und Host-Prozessor – dargestellt. Die gesamte Steuerung des Host-Prozessors wird von dem entsprechenden Host-Betriebssystem (H-OS) ausgeführt und entspricht einem Rechnersystem ohne ein FPGA-Koprozessor. Durch die Erweiterung des FPGA-Betriebssystems kommt auf der linken Seite der Gerätetreiber (DD), die FPGA-Koprozessor Bibliothek (LIB) und die Hardware-Management-Unit (HMU) hinzu.

Der Gerätetreiber stellt den direkten Zugriff auf den FPGA-Koprozessor und die entsprechenden Register her. Durch das Verändern der

Registerinhalte werden indirekt die daran angeschlossenen Bausteine wie z.B. die Taktgeneratoren oder der FPGA-Baustein eingestellt. Die darauf aufbauende Bibliothek (LIB) koordiniert die einzelnen Registerzugriffe des Gerätetreibers und fügt diese zu einzelnen Funktionen zusammen. Diese, in mehrere Gruppen eingeteilte Funktionen, ermöglichen z.B. das Setzen der Taktgeneratoren, die Konfiguration bzw. der Readback der FPGA-Bausteine oder die Übertragung von Daten durch einfache Funktionsaufrufe. Die Bibliothek umfasst somit alle Funktionen, die zum Betrieb des FPGA-Koprozessor notwendig sind und stellt diese der darüberliegenden Hardware-Management-Unit zur Verfügung. Die Aufgabe der HMU beinhaltet die Ablaufsteuerung der einzelnen Anwendungen und die Durchführung der Prozesswechsel auf dem FPGA-Koprozessor. Jede Anwendung, die Daten auf dem FPGA-Koprozessor verarbeitet, ist sowohl mit dem Host-Betriebssystem aber auch über die einheitliche API-Schnittstelle mit der HMU verbunden.

Dieser Aufbau des Betriebssystems in mehreren Schichten ermöglicht eine einfache Unterstützung von mehreren FPGA-Koprozessoren. Durch den Austausch des Gerätetreibers und der FPGA-Prozessor spezifischen Software-Bibliothek wird eine solche Portierung erreicht. Besteht ebenfalls eine Portierung der FPGA-Schaltung so kann unter Verwendung des gleichen Programms für den Host-Rechner die Anwendung auf mehreren FPGA-Koprozessoren ausgeführt werden.

4.4 Umsetzungskonzepte

In diesem Abschnitt werden drei verschiedene Konzepte für den Aufbau des FPGA-Betriebssystems erörtert und entsprechend den gegebenen Randbedingungen bewertet. Ziel dieser Bewertung ist die Auswahl des am besten geeigneten Konzeptes, das in dem anschließenden Schritt umgesetzt wird. Die drei zur Auswahl stehenden Konzepte sind: ein Client-Server Modell, ein erweiterter Gerätetreiber und eine vollständige Integration in ein vorhandenens Betriebssystem.

Die Bewertung dieser drei Konzepte erfolgt nach den in Abschnitt 4.2 aufgelisteten Anforderungen. Einzige Ausnahme dabei ist die Datenaustauschrate, die nur durch den Gerätetreiber bestimmt wird. Über

diese in Abschnitt 4.2 genannten Anforderungen hinaus wird auch die Portabilität des Betriebssystems auf mehrere Host-Betriebssysteme und mehrere FPGA-Koprozessoren, der Einfluß auf das Host-Betriebssystem und die Kommunikation zwischen Anwendung und Betriebssystem zur Bewertung herangezogen. Ein letztes, eher praktisches Bewertungskriterium ist die Ausführungsebene des Betriebssystems, die einen Einfluß auf den Entwicklungsaufwand hat.

In den nachfolgenden drei Abschnitten werden die einzelnen Konzepte kurz erläutert und anhand der Bewertungskriterien werden die jeweiligen Vor- und Nachteile genannt.

4.4.1 Client-Server Modell

Der Client-Server Ansatz ist das erste Modell mit dem die Funktionalität des FPGA-Betriebssystems umgesetzt werden kann. Der Server übernimmt dabei die Aufgaben der Anwendungsverwaltung, der Koordinierung und der Steuerung des FPGA-Koprozessors. Wie in Abbildung 4.4 dargestellt, arbeiten sowohl die HMU des Servers als auch die FPGA-Koprozessor Bibliothek in der Ausführungsebene 'User-Space' und kann daher mit herkömmlichen Programmierwerkzeugen wie Compiler und Debugger erstellt und einfach getestet werden. Die API-Schnittstelle des FPGA-Betriebssystems liegt innerhalb der Client-Anwendung und übernimmt den gesamten Datenaustausch zwischen Client und Server. Das Konzept der Client-Server Kommunikation basiert auf einer standardisierte Interprozess Kommunikation die von dem Host-Betriebssystem bereitgestellt wird und daher eine Portierung des gesamten FPGA-Betriebssystems auf andere Rechnersysteme sehr vereinfacht. Desweiteren ist auch eine Portierung auf andere FPGA-Koprozessor durch den Austausch des Gerätetreibers und der Bibliothek einfach zu realisieren. Aus der Abbildung wird ebenfalls ersichtlich, daß sowohl der Host- als auch der FPGA-Prozessor parallel und unabhängig voneinander betrieben werden.

Ein weiterer Vorteil dieser Architektur ist eine mögliche Netzwerkanbindung durch Verwendung von Sockets[1] anstatt der Interprozess Kom-

[1] Sockets sind standardisierte Betriebssystemelemente die einen Datenaustausch

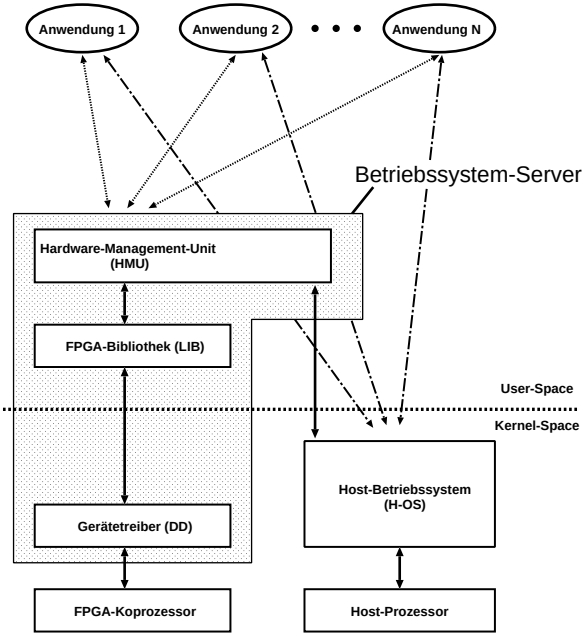


Abbildung 4.4: Schichtenmodell des Client-Server Betriebssystems.

munikation. Ein solcher Netzwerkbetrieb ist jedoch nur bei sehr rechenintensiven Anwendungen effektiv einsetzbar da die Datentransferrate durch die Netzwerkverbindung und nicht durch den Gerätetreiber beschränkt wird.

Bei dem Client-Server Modell werden alle Anforderungen die an den Multitaskingbetrieb gestellt werden innerhalb des Servers realisiert. Die Vorteile dieses Konzepts liegen bei der Portabilität, dem vergleichsweise einfachen Aufbau und den Testmöglichkeiten dieses Client-Server Modells. Die Nachteile dieser Umsetzung sind zum einen der zusätzliche Kommunikationsoverhead der durch den Datenaustausch zwischen Cli-

über ein Netzwerk ermöglichen.

ent und Server entsteht und zum anderen die Beeinflussung des Host-Systems durch die häufigen Host-Betriebssystemaufrufe bei dieser Kommunikation.

4.4.2 Erweiterter Gerätetreiber

Das zweite Konzept ist die Erweiterung der Gerätetreiberfunktionalität. Bei diesem Konzept werden sowohl die FPGA-Koprozessor Bibliothek als auch die Hardware-Management-Unit in den Gerätetreiber integriert. Die nachfolgende Abbildung 4.5 illustriert das entsprechende Schichtenmodell des erweiterten Gerätetreibers. Im Gegensatz zum Client-

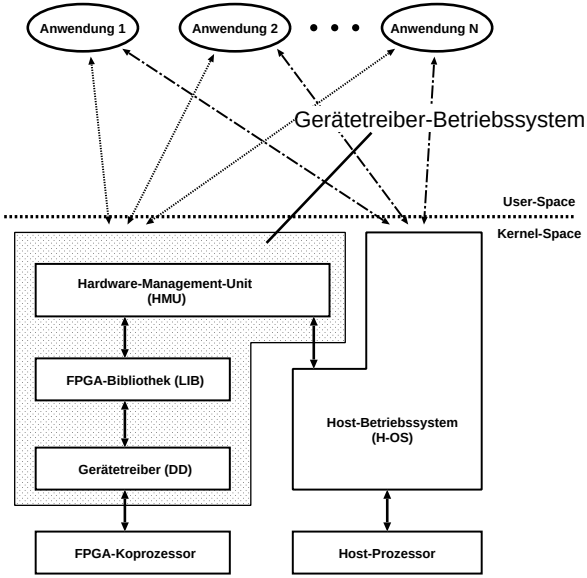


Abbildung 4.5: Schichtenmodell des erweiterten Gerätetreibers.

Server Konzept arbeitet das gesamte FPGA-Betriebssystem im 'Kernel-Space' und die Kommunikation zwischen Anwendung und dem FPGA-Betriebssystem erfolgt über Aufrufe des erweiterten Gerätetreibers. Die

Steuerung des FPGA-Koprozessors erfolgt auch hier parallel und unabhängig zu der Ausführung der Anwendungen auf den Host-Prozessor.

Auch in diesem Konzept wird die geforderte Funktionalität innerhalb der HMU realisiert. Der Vorteil dieses Gerätetreibers ist die effektive Kommunikation zwischen der Anwendung und dem FPGA-Betriebssystem, die über einfache Funktionsaufrufe realisiert wird. Darüberhinaus entstehen nur wenig Wechsel zwischen dem 'User-Space' und dem 'Kernel-Space', da die gesamte Steuerung des FPGA-Koprozessor im Gerätetreiber und somit im 'Kernel-Space' durchgeführt wird. Nachteilig wirkt sich dieser erweiterte Gerätetreiber auf die Portabilität des FPGA-Betriebssystems aus, da für jede Kombination aus FPGA-Koprozessor und Host-Betriebssystem ein eigener Gerätetreiber erstellt und getestet werden muß. Zusätzlich ist der Entwicklungsaufwand vor allem aufgrund der Ausführung im 'Kernel-Space' hoch.

4.4.3 Betriebssystemintegration

Die vollständige Integration in ein vorhandenes Betriebssystem kennzeichnet sich durch einen koordinierten Ablauf der Anwendung auf beiden Prozessorelementen, so wie es bei Anwendungen der Maschinenbefehls-Ebene vorausgesetzt wird. Diese Koordinierung erfordert vom Betriebssystem eine gemeinsame Auswahl der einzelnen Anwendungsteile für den FPGA- und den Host-Prozessor. Die Kommunikation erfolgt auch hier über Systemfunktionsaufrufe, die von dem veränderten Betriebssystem bereitgestellt werden. Abbildung 4.6 zeigt diese Integration graphisch dar. Bedingt durch den gemeinsamen Ablauf der Anwendungen auf dem Host-Prozessor und dem FPGA-Koprozessor entsteht nur eine Verbindung zwischen den Anwendungen und dem Betriebssystem.

Die Vorteile eines solchen Betriebssystems ist die Einbeziehung der erweiterten Funktionalität in die Entwicklungswerkzeuge wie z.B dem Compilern und die Nutzung des FPGA-Koprozessors durch das Betriebssystem selbst. Darüberhinaus wird die schon optimierte Schnittstelle zwischen Anwendung und Betriebssystem für die Kommunikation genutzt. Nachteilig wirkt sich eine solche Integration auf die Portabilität aus, da diese Integration eine Anpassung im Betriebssystem selbst er-

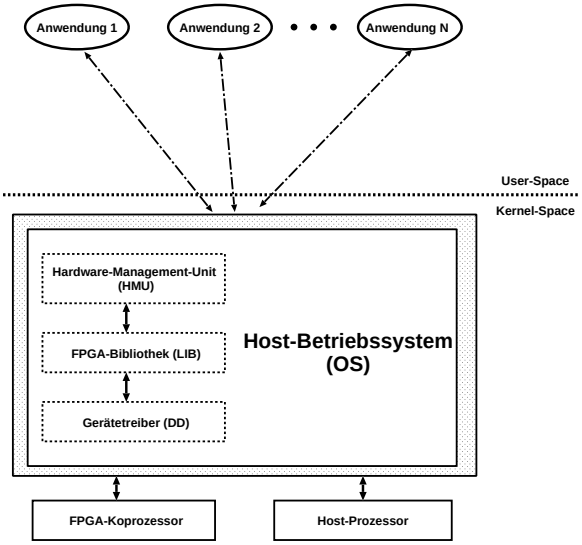


Abbildung 4.6: Schichtenmodell des vollständig integrierten FPGA-Betriebssystems.

fordert. Diese Anpassung muß für jedes Betriebssystem aber auch für jeden weiteren unterstützten FPGA-Koprozessor einzeln durchgeführt werden, was einen hohen Arbeitsaufwand bedeutet. Darüberhinaus ist die Integration in ein vorhandenes Betriebssystem mit einem sehr hohen Aufwand verbunden.

4.4.4 Diskussion

Unter Berücksichtigung der Anforderungen, die in Abschnitt 4.2 aufgelistet sind, werden die drei oben genannten Modelle bewertet. Alle drei Modelle erfüllen die geforderte Funktionalität die an das FPGA-Betriebssystem gestellt werden. Im Detail konzentriert sich die Auswahl daher auf fünf Anforderungen, die zu gleichen Teilen in die Gesamtbewertung eingehen.

Die Performance des FPGA-Koprozessors wird maßgeblich von der Umsetzung des Gerätetreibers und dem FPGA-Koprozessor selbst bestimmt und ist in der Regel gut. Durch der Verwendung des gleichen Gerätetreibers in allen Modellen sind die Ausirkungen auf die Konzepte gleich. Nur die Leistungsfähigkeit des Client-Server Modells ist aufgrund des für die Kommunikation notwendigen Overheads geringfügig schlechter. Dieser Overhead, der nicht von der Datenmenge abhängt, beeinflußt die Kommandoausführungszeit und verlängert z.B. den Datentransfer von 128kByte von 5.4ms auf 5.8ms. Unter Berücksichtigung der bei den verschiedenen Anwendungen analysierten Datenmengen und des geringer werdenden Effektes bei großen Datenmengen kann dieser Overhead vernachlässigt werden.

Die zweite Anforderung für die Beurteilung ist die Portabilität des Betriebssystems. Unterschieden wird dabei in die Portierbarkeit zur Unterstützung weiterer FPGA-Koprozessoren und zum Betrieb auf anderen Host-Betriebssysteme. Die Unterstützung eines anderen FPGA-Koprozessors ist durch die Bereitstellung des Gerätetreiber und der FPGA-Koprozessor Bibliothek einfach durchzuführen. Diese beiden Module sind hardwareabhängig und werden daher von den Herstellern zusammen mit dem FPGA-Koprozessor ausgeliefert. Durch Auswechseln dieser beiden Funktionsblöcke ist der Einsatz eines anderen FPGA-Koprozessors[2] schnell zu realisieren. Die Portierung auf ein anderes Host-Betriebssystem ist vor allem bei dem erweiterten Gerätetreibermodell und bei dem integrierten Betriebssystemodell aufwendig. So ist eine Portierung des integrierten Modells nur möglich, wenn die internen Quellen des Betriebssystems bekannt sind und somit Veränderungen im Betriebssystem selbst durchgeführt werden können. Bei dem erweiterten Gerätetreiber erfolgt die Portierung ohne Kenntnis des Betriebssystems, aber dennoch ist die Integration und der Aufbau eines Gerätetreibers bei jedem Host-Betriebssystem unterschiedlich, was eine solche Portierung ebenfalls nicht begünstigt. Im Gegensatz dazu ist die Portierung des Client-Server Modells sehr einfach möglich, da nur der vom Hersteller des FPGA-Koprozessor mitgelieferte Gerätetreiber und die Software-

[2]Neben der Auswechslung der beiden Module muß der FPGA-Koprozessor auch die in Abschnitt 3.4 erfüllen.

Bibliothek ausgetauscht werden müssen. Der Server ist direkt und ohne Anpassungen portierbar, da er auf standardisierten Elementen aufbaut.

Eine weitere Anforderung bezüglich der Kommunikation zwischen dem Client und dem Server fordert, daß diese möglichst effektiv durchgeführt wird. Sowohl die Betriebssystemintegration als auch die Lösung des erweiterten Gerätetreibers verwendet Systemaufrufe für die Kontrolle und den Datenaustausch mit der FPGA-Anwendung. Diese Systemaufrufe sind sehr schnell und bedingt durch die häufigen Aufrufe effektiv umgesetzt. Darüberhinaus ermöglicht die Ausführung im 'Kernel-Space' einen direkten Zugriff auf den Datenspeicher was in dem Client-Server Modell nicht möglich ist. Die Kommunikation in dem Client-Server Modell verwendet die standardisierte Interprozess Kommunikation, die mit einem geringen Overhead verbunden ist. Dieser Overhead begründet sich durch den wirklichen Austausch von Daten, die Systemaufrufe für die Kommunikation und die Prozesswechsel auf dem Host-Prozessor. Bei der Betrachtung einer Anwendung wird deutlich, wie gering der Einfluß dieses Overheads auf die gesamte Ausführungszeit ist. So benötigt z.B. die Filterung von Bilddaten (Anwendung 3 in Tabelle 4.1) nur 11 Kommandoaufrufe des FPGA-Betriebssystems für die gesamte Ausführung.

Die Durchführung der Prozesswechsel auf dem FPGA-Koprozessor erfordern zusätzliche Rechenleistung des Host-Prozessors, die einen Einfluß auf die anderen Host-Anwendungen haben. Ein minimaler Einfluß ist daher eine weitere Anforderung die an die zu untersuchenden Modelle gestellt werden. Die beiden Modelle des integrierten Betriebssystems und des erweiterten Gerätetreibers führen die Betriebssystemfunktionen im Kernel-Mode aus, der sich unter anderem dadurch kennzeichnet, daß er nur von Interrupts unterbrochen werden kann. Bei einer gemessenen Prozesswechselzeit von $\approx$80ms ergibt sich somit eine nicht unerhebliche Beeinträchtigung der laufenden Host-Anwendungen die während dieser Zeit nicht ausgeführt werden können. Bei dem Client-Server Modell erfolgt die Verarbeitung als eine weitere Host-Anwendung die somit auch durch eine andere Anwendung verdrängt werden kann. Daher ist bei dem Client-Server Modell der Einfluß auf andere Anwendungen geringer. Der zu erwartende Einfluß auf die Prozesswechselzeit kann durch die Wahl

einer hohen Ausführungspriorität des Servers minimiert werden.

Die letzte betrachtete Anforderung ist der Entwicklungsaufwand zur Umsetzung des Betriebssystems. Auch hier ist das integrierte Betriebssystemodell das aufwendigste, da neben der Umsetzung der Funktionalität auch die Anpassungen im Host-Betriebssystem durchgeführt werden müssen. Der Entwicklungsaufwand für den erweiterten Gerätetreiber ist ebenfalls aufwendig, da das FPGA-Betriebssystem im Kernel-Mode arbeitet. Einzig das Client-Server Modell kann als gewöhnliche Anwendung erstellt werden und verarbeitet die Daten im 'User-Space'. Dennoch ist der Aufwand für das Client-Server Modell höher aufgrund der Realisierung der Kommunikation zwischen Client und Server.

Die Beurteilung der ausgewählten Anforderungen zeigt, daß sowohl der erweiterte Gerätetreiber als auch das integrierte Betriebssystem Vorteile bei der Performance und der Kommunikation besitzen. Der Vorteil bei beiden Anforderungen kommt durch den Overhead bei der Kommunikation zustande. Darüberhinaus ist die, für die Ausführung von Anwendungen der Maschinenbefehlsebene notwendige enge Kopplung zwischen dem FPGA- und dem Host-Betriebssystem nur bei dem integrierten Betriebssystem gegeben.

Demgegenüber stehen die Vorteile des Client-Server Modells, die sowohl eine einfache Portierung, einen geringeren Entwicklungsaufwand und einen geringeren Einfluß auf die Ausführung weiterer Host-Anwendungen gewährleisten. Unter Berücksichtigung der Anwendungen aus dem Abschnitt 4.1 ist die Anzahl der aufzurufenden Kommandos sehr gering. Daher ist auch der Einfluß der notwendigen Client-Server Kommunikation auf die Ausführungsperformance nur gering. Die Einschränkung bei einem Client-Server Betriebssystem nur Anwendungen der Funktions- oder der Programmebene verarbeiten zu können stellt keine Beeinträchtigung der Verwendbarkeit da, denn in der Regel werden heutzutage nur diese Anwendungen durch einen FPGA-Koprozessor beschleunigt.

Die weitere Betrachtung der Anwendungsanalyse zeigt, daß die Host- und die FPGA-Anwendungen in voneinander unabhängigen Zeitscheiben verarbeitet werden müssen, um den systembedingten Overhead zu minimieren. Diese unabhängige Umsetzung ist am einfachsten und ohne einen hohen Performanceverlust mit dem Client-Server Modell zu realisieren.

4.5 Zusammenfassung

Dieses Kapitel analysiert unterschiedliche Anwendungen, die auf verschiedenen FPGA-Koprozessoren ausgeführt werden. Diese Untersuchung dient zur Bestimmung des Nutzens eines FPGA-Betriebssystems für die Anwendungen die heutzutage auf den FPGA-Koprozessoren ausgeführt werden. Im Mittelpunkt dieser Analyse steht die jeweilige Zeit für die Übertragung der Daten und die Zeit, die zur Ausführung der Anwendung benötigt wird. Erst diese Daten geben Aufschluß darüber, ob die Unterbrechung der Anwendungen auf dem FPGA-Koprozessor notwendig sind. Die ermittelten Ausführungszeiten liegen zwischen wenigen Millisekunden und mehreren Jahren, wobei die durchschnittliche Laufzeit mehrere 100ms beträgt. Desweiteren zeigt die Analyse, daß bei fast allen Anwendungen die verfügbaren lokalen RAM-Elemente mitverwendet werden. Bedingt durch die sehr variierenden Ausführungszeiten wird klar, daß für die allgemeine Nutzung des FPGA-Koprozessors ein preemptives Multitasking Betriebssystem notwendig ist. Nur ein solches Betriebssystem ermöglicht eine schnelle und faire Ausführung der Anwendungen bei einer nur minimalen Beeinflussung der Ausführungszeiten.

Zusammen mit den Aufgaben aus Abschnitt 3.4.4 werden die Anforderungen an ein solches Betriebssystem definiert und die grundlegenden Komponenten beschrieben. Für die Umsetzung stehen drei unterschiedliche Konzepte zur Auswahl: ein Client-Server Modell, ein erweiterter Gerätetreiber und eine in das Host-Betriebssystem integrierte Lösung. Anhand der gegebenen und erarbeiteten Anforderungen werden diese drei Umsetzungskonzepte bewertet. Diese Bewertung zeigt, daß sowohl der erweiterte Gerätetreiber, als auch die Integration in das Host-Betriebssystem aufgrund eines hohen Portierungs- und Entwicklungsaufwands und einer nur kleinen zu erwartenden Performanceverbesserung keine entscheidende Vorteile gegenüber dem Client-Server Modell besitzen. Vielmehr ist das Client-Server Modell am besten geeignet, um ein portables, allgemein verwendbares und schnell zu realisierendes preemptives Multitasking Betriebssystem für FPGA-Koprozessoren aufzubauen.

Kapitel 5

Implementierung des FPGA-Betriebssystems

Wie zuvor im Kapitel 4 gezeigt wurde ist das Client-Server Modell am Besten für die Implementierung des FPGA-Betriebssystem geeignet. Aus diesem Grund ist dieses Modell für die Realisierung des FPGA-Betriebssystems auf Basis eines modernen FPGA-Koprozessors ausgewählt worden.

Dieses Kapitel beschreibt die konkrete Umsetzung dieses Client-Server Betriebssystems (CS-OS) das eine allgemeine Verwendung des FPGA-Koprozessors durch verschiedene Benutzer und aus unabhängigen Anwendungen heraus ermöglicht. Nach einer kurzen Zusammenfassung der gestellten Anforderungen werden die Grundlagen des Client-Server Betriebsssystem beschrieben. Im Anschluß an diese theoretischen Grundlagen von Betriebssystemen folgt eine Beschreibung der verwendeten FPGA-Koprozessor Karte und der Aufbau des Softwaremodells das dem Betriebssystem zugrundeliegt. Beschrieben werden die einzelnen zum Einsatz kommenden Softwaremodule inklusive deren Funktionsweise. Abschließend werden die an dem gesamten FPGA-Betriebssystem durchgeführten Messungen dargestellt und analysiert. Ein Vergleich mit den vorhandenen FPGA-Betriebssystemen und die Erörterung der durch den

eingesetzten FPGA-Koprozessor entstandenen Einschränkungen werden in einem weiteren Abschnitt diskutiert.

5.1 Anforderungen

Bei der Implementierung des FPGA-Betriebssystems sind verschiedene Anforderungen zu beachten, die die Funktionsfähigkeit und die Performance des Betriebssystems beeinflussen. Die einzelnen, an das FPGA-Betriebssystem gestellten Anforderungen sind ausführlich in Abschnitt 3.4.4 und in Abschnitt 4.2 erläutert bzw. diskutiert und werden hier nur kurz beschrieben. Für das gewählte Client-Server Modell sind vor allem die folgenden Anforderungen für die Implementierung wichtig:

- Portabilität
- Verteilung der FPGA-Rechenzeit
- Performance

Die Portabilität des Client-Server Betriebssystem umfasst zwei mögliche Portierungen. Zum einen soll das Client-Server Betriebssystem auf mehreren unterschiedlichen Host-Betriebssystemen ausführbar sein und zum anderen sollen mehrere FPGA-Koprozessoren unterstützt werden. Erreicht wird diese umfassende Portabilität durch den Einsatz von standardisierten Kommunikationsmechanismen und durch die Ausführung des Betriebssystems als eine 'User-Space' Anwendung.

Eine faire Verteilung der auf dem FPGA-Koprozessor zur Verfügung stehenden Rechenzeit durch das Client-Server Betriebssystem ist eine weitere wichtige Anforderung. Diese Verteilung gewährleistet, daß jede der konkurrierenden Anwendungen den FPGA-Koprozessor für die Ausführung zur Verfügung gestellt bekommt. Gesteuert wird diese Verteilung durch Prioritäten und die Ausführung selbst.

Die dritte Anforderung an das Client-Server Betriebssystem betrifft die Performance bei der Verarbeitung der einzelnen Anwendungen. Bei dem Client-Server Modell erfolgt der Datenaustausch zwischen Anwendung und Betriebssystem über Interprozesskommunikation. Aus diesem

Grund ist bei der Implementierung dieser Kommunikation auf einen geringen Overhead zu achten, um die Gesamtperformance nicht zu beeinflussen.

Über diese drei Anforderungen hinaus sollte das Betriebssystem eine einheitliche API-Schnittstelle auf der Client Seite bereitstellen, die im Zusammenhang mit der Simulation der FPGA-Schaltungen die gesamte Entwicklung vereinfacht.

5.2 Betriebssystem Grundlagen

Bei der Realisierung des Client-Server Betriebssystems sind speziell die oben genannten Anforderungen zu berücksichtigen, wobei die anderen, hier nicht genannten Anforderungen, ebenfalls erfüllt sein müssen.

Dieser Abschnitt beschreibt die grundlegenden Konzepte, die bei der Implementierung des Client-Server Betriebssystem Anwendung finden. Diese Grundlagen sind unterteilt in den Aufbau der Kommunikation zwischen Client und Server, dem Zustandsmodell der Anwendungen innerhalb des Client-Server Betriebssystems und die gewählte Schedulding-Strategie, die die Ausführungsreihenfolge bestimmt. Die Auswahl und Implementierung dieser drei Modelle ist ausschlaggebend für die Performance, die Portabilität und die faire Zuteilung des FPGA-Koprozessors an die einzelnen Anwendungen. Jedes Modell wird in den nachfolgenden Abschnitten ausführlich erläutert und kurz bewertet.

5.2.1 Kommunikationsmodell

Das Kommunikationsmodell beschreibt die Durchführung des Datenaustauschs zwischen den Clients und dem Server, der zur Übermittlung der auszuführenden Kommandos verwendet wird. Zur Vermeidung eines Overheads ist diese Kommunikation bei dem Client-Server Betriebssystem so auszuführen, daß auch bei großen Datenmengen ein effektiver Datenaustausch erfolgen kann. Darüber hinaus ist ein Kommunikationmechanismus auszuwählen, der zum einen eine einfache Portierung auf andere Host-Betriebssysteme ermöglicht und zum anderen eine für die

sequenzielle Ausführung der einzelnen Kommandos notwendige und zuverlässige Kommunikation gewährleisten.

Aufgrund der Anforderungen, die an die Kommunikation gestellt werden, wird die standardisierte Interprozess Kommunikation (IPC) nach dem System V Standard [Ste90] ausgewählt. Ausschlaggebend für diese Wahl ist vor allem die Portabilität, denn IPC ist auf vielen Betriebssystemen verfügbar. Für den Datenaustausch und die Synchronisation werden folgende Komponenten des IPC System V eingesetzt:

Semaphoren werden für die Synchronisation zwischen dem Server und dem Client eingesetzt um z.B. einen gleichzeitigen Zugriff auf ein gemeinsam genutztes Shared Memory Element zu verhindern.

Message Queues werden für den Austausch der Kommandos eingesetzt, denn sie ermöglichen einen unidirektionalen Datenaustausch zwischen Client und Server. Aufgrund des unidirektionalen Kanals werden stets zwei entgegengesetzt arbeitende Message Queues aufgebaut.

Shared Memories dienen zum Austausch von großen Datenmengen zwischen den Clients und dem Server. Angehängt an jedes Shared Memory ist eine Semaphore, die den gleichzeitigen Zugriff auf die Daten verhindert.

Das zugrundeliegende Kommunikationsmodell, das bei dem Client-Server Betriebssystem zur Anwendung kommt, ist abgeschlossen, d.h. jedes von Client ausgesendete Kommando wird nach der Ausführung mit einer Antwort bestätigt. Die Notwendigkeit der Antwortpakete wird deutlich bei der Betrachtung des folgenden Beispiels: Bei der Filterung von Bildern werden zuerst die Bilddaten zum FPGA-Koprozessor übertragen und anschließend wird die Filterung gestartet. Erfolgt die Kommunikation ohne Antwortpakete, so kann die Filterung gestartet werden, obwohl die Datenübertragung noch nicht beendet ist. Aufgrund der nicht vorhandenen Daten kann somit ein falsches Ergebnisbild errechnet werden. Dieses Beispiel macht klar, daß die blockierende Eigenschaft der Funktionaufrufe innerhalb der Client-Anwendung erhalten bleiben muß und Optimierungen der Kommunikation wie z.B. das Sliding-Window

Prinzip, das bei der TCP/IP Datenübertragung [Lie00] zum Einsatz kommt, hier nicht eingesetzt werden können.

In der nachfolgenden Darstellung 5.1 ist das Kommunikationsdiagramm für das 'Init'-Kommando aufgezeigt. Dieses 'Init'-Kommando meldet einen neuen Client bei dem Server-Betriebssystem an und generiert alle notwendigen Kommunikationkanäle für den Betrieb. Nach der

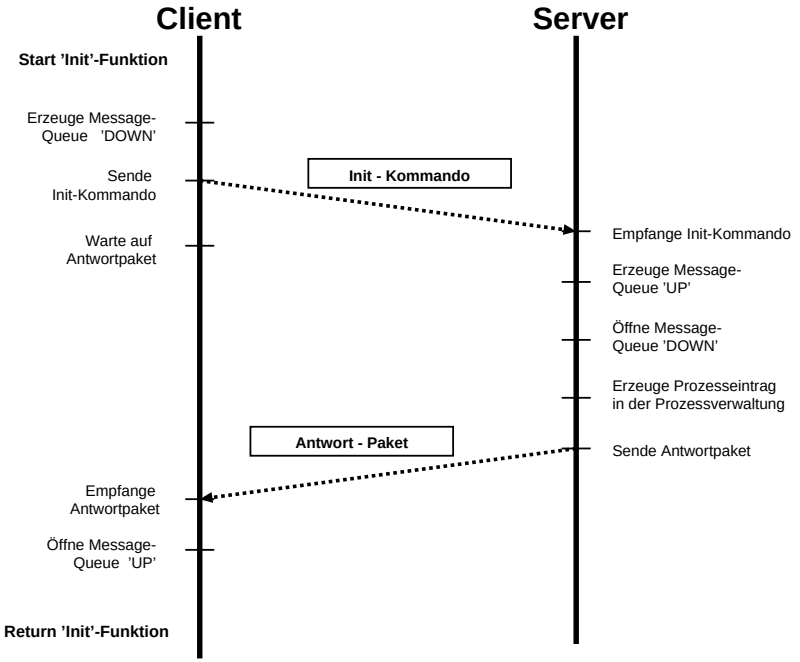


Abbildung 5.1: Kommunikationsmodell beim Anmelden eines neuen Clients beim Server. *Dieses Kommando meldet einen Client beim Server an und baut alle notwendigen Kommunikationskanäle auf.*

Generierung des 'DOWN'-Kanals, über den die Antwortpakete an den Client zurückgesendet werden, meldet sich der neue Client beim Server an. Hierzu wird ein weiterer gemeinsam genutzter Kanal verwendet. Der

Server führt das empfangene 'Init'-Kommando aus. Dabei generiert er seinerseits einen 'UP'-Kanal und fügt den neuen Client in die Prozessliste ein. Zum Abschluß sendet der Server das Antwortpaket zurück an den Client der daraufhin die 'Init'-Funktion beenden kann. Anhand des Beispiels der 'Init'-Funktion soll verdeutlicht werden, wie die Kommunikation zwischen dem Server und den einzelnen Clients in dem implementierten Client-Server Betriebssystem durchgeführt wird.

Der ersten Anforderung an die Kommunikation – wenig Overhead durch die zusätzliche notwendige Kommunikation – wird durch ein kleines Datenpaket Rechnung getragen. Dieses Datenpaket ist in Abbildung 5.2 illustriert. Die Struktur ist so ausgelegt, daß die Parameter aller Kom-

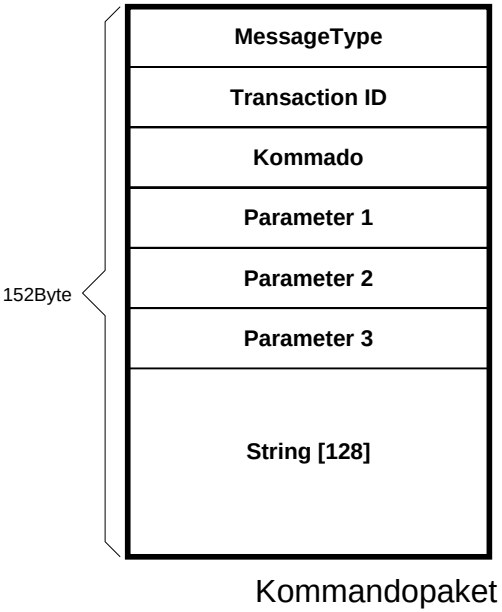


Abbildung 5.2: Struktur der Datenpakete die für die Kommunikation zwischen Client und Server eingesetzt werden.

mandos mit diesem Datenpaket zum Server übertragen werden können. Die einzelnen Felder des Datenpakets beinhalten:

- **MessageType** zur Unterscheidung von unterschiedlichen Kommandopaketen. Dieses Feld wird von der IPC verwendet.
- **Transaktionsnummer** zum Protokollieren der Zugriffsreihenfolge des Clients.
- **Kommando** kodiert die einzelnen Kommandos mit einer systemweit einheitlichen und einzigartigen Nummer.
- **Parameter 1-3** übertragen die jeweiligen Kommandoparameter zwischen Client und Server.
- **String** aus 128 Bytes zur Übertragung von Message Queue Namen oder mehrerer zusätzlichen Parametern.

Größere Datenmengen wie z.B. Bilddaten oder Genomdatenbanken werden über Shared Memory Elemente ausgetauscht, die es dem Server erlauben direkt auf die Daten des Clients zuzugreifen, obwohl der Server nicht im Kernel-Mode arbeitet. Diese Zugriffsmöglichkeit ist vergleichbar mit den Zugriff durch einen Gerätetreiber oder durch das Betriebssystem.

Durch die Wahl der Interprozess Kommunikation für den Datenaustausch zwischen Client und Server ist das entstehende Client-Server Betriebssystem einfach auf andere Host-Betriebssysteme portierbar und verfügt über eine hohe Performance aufgrund der geringen Datenmengen, die ausgetauscht werden.

5.2.2 Zustandsmodell des Betriebssystems

Zur Unterscheidung der verschiedenen Zustände, in denen sich die einzelnen am Server angemeldeten Anwendungen befinden, wird das nachfolgende Zustandsmodell eingeführt. Aufgrund dieses Zustandsmodells wird die Auswahl der Anwendung getroffen, die durch den nächsten Prozesswechsel in Besitz des FPGA-Koprozessor kommt.

Für die Anwendungen, die sowohl auf dem Host- als auch auf dem FPGA-Koprozessor ausgeführt werden, sind die Zustände in zwei Gruppen unterteilt. Wie in der Abbildung 5.3 graphisch dargestellt, wird die erste Gruppe nur im Client-Server Betriebssystem ausgeführt, während die zweite Gruppe die Ausführung auf dem FPGA-Koprozessor beschreibt.

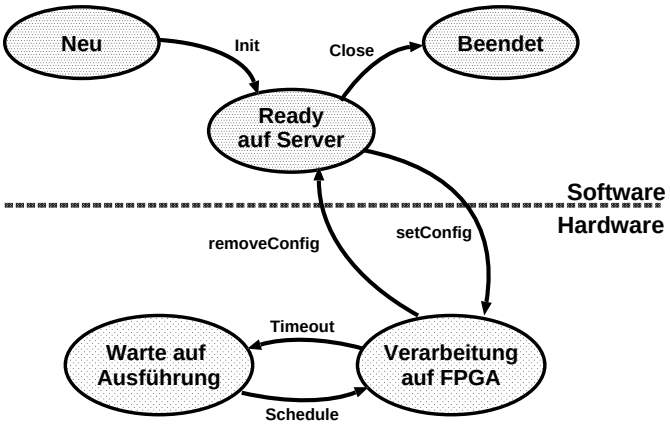


Abbildung 5.3: Zustandsmodell der Anwendungen die durch das Client-Server Betriebssystem ausgeführt werden.

Die einzelnen Zustände sind wie folgt definiert.

Neu: Neu definiert eine Anwendung, die aus Sicht des Client-Server Betriebssystem nicht gestartet ist. Dieser Zustand kann sowohl einen Host-Prozess beschreiben der noch nicht gestartet ist, als auch einen Host-Prozess der gestartet ist, sich aber noch nicht bei dem Client-Server Betriebssystem angemeldet hat. Erst mit der Anmeldung (Init) wechselt der Zustand in 'Ready auf Server'.

Ready auf Server: Dieser Zustand bedeutet, daß die Anwendung sich beim Server angemeldet hat, aber noch kein FPGA Konfigurationsbitstrom geladen ist, der auf dem FPGA-Koprozessor ausgeführt

werden könnte. Trotzdem werden in diesem Zustand Kommandos von Client ausgeführt, auch ohne daß der FPGA-Baustein konfiguriert ist. Durch die Abmeldung (Close) des Host-Prozesses wird der Wechsel in den 'Beendet' Zustand überführt und die Anwendung wird beendet. Das Setzen (setConfig) eines zuvor geladenen Bitstroms verändert den Zustand in 'Verarbeitung auf FPGA'.

Beendet: definiert die Aktivität der Anwendung als beendet. Das bedeutet, wie beim Zustand 'Neu', das der Host-Prozess entweder sich nur beim Client-Server Betriebssystem abgemeldet hat oder komplett beendet ist.

Verarbeitung auf FPGA: Dies ist der Zustand in der die Anwendung auf dem FPGA-Baustein verarbeitet wird. Erreicht wird dieser Zustand durch den Wechsel vom Zustand 'Ready auf Server' sobald der FPGA-Baustein konfiguriert wird oder wenn die Anwendung zuvor von der Verarbeitung verdrängt wurde und nun erneut ausgeführt werden soll. In dem letzten Fall wechselt der Zustand von 'Warte auf Ausführung' in den Verarbeitungszustand. Demgegenüber wechselt eine Anwendung in den 'Warte auf Ausführung' Zustand sobald die Zeitscheibe aufgebraucht ist (Timeout). In den Zustand 'Ready auf Server' wird gewechselt, sobald die Konfiguration aus dem FPGA entfernt wird (removeConfig). Mit diesem zweiten Wechsel aus dem Verarbeitungszustand wird gleichzeitig auch die Hardwareebene verlassen, denn die Anwendung verfügt über keine FPGA-Schaltung mehr, die auf dem FPGA-Koprozessor ausgeführt werden könnte.

Warte auf Ausführung: ist der letzte Zustand die eine Anwendung innerhalb des Client-Server Betriebssystem annehmen kann. In diesem Zustand verweilen alle Anwendungen die zuvor verarbeitet wurden und dann nach Ablauf der Zeitscheibe in ihrer Ausführung unterbrochen wurden. Bei der Unterbrechung (Timeout) wechseln die Anwendungen vom verarbeitenden in den wartenden Zustand. Umgekehrt wechseln die Anwendungen vom wartenden in den verarbeitenden Zustand wenn sie erneut zur Ausführung gelangen

(Schedule). Während die Prozesse in diesem Zustand verweilen werden zwischen Client und Server keine Daten ausgetauscht.

Am Beispiel der Mustererkennunganwendung soll der Ablauf durch die verschiedenen Zustände verdeutlicht werden. Zu Beginn ruft das Host-Programm die 'Init'-Funktion auf, die den Wechsel vom Zustand 'Neu' in den Zustand 'Ready auf Server' veranlasst. Daran anschließend wird das FPGA-Design geladen, die Takteinstellungen verändert und weitere Vorbereitungen getroffen. Mit dem Funktionsaufruf 'Config FPGA' wird der Zustand in 'Verarbeitung auf FPGA' überführt und die Daten werden verarbeitet. Bei einem Prozesswechsel auf dem FPGA-Koprozessor springt der Zustand nun zwischen 'Verarbeitung' und 'Warten' hin und her. Nach Beendigung der Mustererkennung wird die FPGA-Schaltung mit der 'ClearFPGA' Funktion beendet und der Zustand springt auf 'Ready auf Server' zurück. Nach weiteren Aufräumarbeiten durch das Host-Programm wird die Anwendung innerhalb des Client-Server Betriebssystem durch die 'Close'-Funktion beendet und wechselt in den 'Beendet' Zustand.

5.2.3 Scheduling Strategie

Das dritte Konzept, das der Implementierung des Client-Server Betriebssystems zugrundeliegt, befasst sich mit der Scheduling-Strategie. Diese bestimmt die Ausführungsreihenfolge der einzelnen Anwendungen durch das Client-Server Betriebssystem.

Die unterschiedlichen Scheduling-Strategien müssen hinsichtlich ihrer Qualität und Eignung für den Einsatz in dem Client-Server Betriebssystem bewertet werden. Die wichtigen Kriterien die zur Bewertung herangezogen werden sind:

Prozessorauslastung: Maß für die Auslastung ist die Zeit, in der Anwendungen auf den Prozessor ausgeführt werden. Die gewählte Scheduling-Strategie sollte diese Auslastung maximieren.

Durchsatz: Der Durchsatz wird bestimmt durch die pro Zeiteinheit fertiggestellten Anwendungen. Ebenso wie die Auslastung ist dieser Wert durch die Wahl der Scheduling-Strategie zu maximieren.

Turnaroundzeit: Die Turnaroundzeit ist die Zeit, die zwischen zwei aufeinanderfolgenden Zuteilungen des Prozessors an eine Anwendung vergeht. Sie ist durch die Wahl der Scheduling-Strategie zu minimieren.

Wartezeit: Die Wartezeit ist die Zeit, die eine Anwendung auf die Zuteilung des Prozessors wartet. Diese Zeit gilt es ebenfalls zu minimieren.

Antwortzeit: Die Antwortzeit definiert die Zeitspanne zwischen der Ankunft z.B. eines Kommandos und der Reaktion in Form der Ausführung dieses Kommandos auf dem FPGA-Koprozessor. Auch diese Zeit gilt es zu minimieren.

Realzeit: Vorgegebene maximale Antwortzeit in der das Betriebssystem auf ein Ereignis reagieren muß.

Abgesehen von der Realzeit, die nur für Realzeitbetriebssysteme von Bedeutung ist, werden alle anderen Kriterien zur Bewertung der jeweiligen Scheduling-Strategie eingesetzt. Aufgrund der zum Teil gegensätzlichen Auswirkungen bei der Optimierung der Kriterien ist abhängig von der jeweiligen Betriebsform jeweils ein Kompromiß zu schließen.

In den nachfolgenden Abschnitten werden die einzelnen Scheduling-Strategien kurz vorgestellt und anhand der oben genannten Kriterien für den Einsatz in dem Client-Server Betriebssystem bewertet.

5.2.3.1 First-Come, First-Served (FCFS)

Das FCFS Schedulingverfahren gehört zu der Gruppe der nichtpreemptiven Verfahren und teilt den Prozessor in der Reihenfolge der Auftragseingänge zu. Die Verwaltung der Anwendungen und die Realisierung dieser Scheduling-Strategie ist sehr einfach und führt daher zu einer sehr hohen Auslastung des Prozessors. Prozesswechsel finden nur bei einem Betriebssystemaufruf oder einer freiwilligen Abgabe des Prozessors durch die jeweilig laufende Anwendung statt, d.h. eine sehr rechenintensive Anwendung kann den Ablauf blockieren.

Dieses FCFS Verfahren hat eine sehr hohe Prozessorauslastung, aber durch die nichtpreemptive Verarbeitung sind die Warte- und Antwortzeiten sehr lastabhängig und unterliegen hohen Schwankungen. Darüber hinaus ist das Verfahren aufgrund der Möglichkeit einer langen Blockade durch eine rechenintensive Anwendung nicht für den Einsatz im Client-Server Betriebssystem geeignet.

5.2.3.2 Shortest Job First (SJF)

Das SJF Schedulingverfahren ist ähnlich zu dem FCFS Verfahren jedoch wird die Reihenfolge der Anwendungen nicht nach dem Auftragseingang sondern nach der voraussichtlichen Ausführungslänge sortiert. Dieses Verfahren kann sowohl preemptiv als auch nichtpreemptiv ausgeführt werden.

Aufgrund des geringen Verwaltungsaufwands ist auch bei dieser Scheduling-Strategie eine hohe Auslastung des Prozessors zu erwarten und durch das Vorziehen der Anwendungen mit kurzen Ausführungslängen werden gleichzeitig die Wartezeiten auf ein Minimum reduziert. Dennoch ist der Einsatz nur bedingt realisierbar, denn die Ausführungslänge ist a priori vor Ablauf nicht bekannt und kann nur aufgrund der letzten Prozessorzuteilungen geschätzt werden. Des weiteren ist die Antwortzeit ebenso wie bei dem FCFS Verfahren sehr hoch und unterliegt zusammen mit der Wartezeit großen Schwankungen. Eine fehlende Priorisierung der Anwendungen und die nichtdeterministische Schedulingreihenfolge sind weitere Gründe die einen Einsatz in dem Client-Server Betriebssystem verhindern.

5.2.3.3 Round-Robin (RR)

Round-Robin ist eine preemptive Scheduling-Strategie die eine gleichmäßige Aufteilung der verfügbaren Prozessorzeit auf alle Anwendungen zum Ziel hat. Das RR Verfahren unterteilt die gesamte Prozessorrechenzeit in einzelne Zeitquanten (auch Zeitscheiben genannt), in denen eine Anwendung verarbeitet wird. Der Prozessor wird dann, entweder freiwillig durch den Aufruf einer Betriebssystemfunktion oder nach Ablauf der Zeitscheibe, der Anwendung wieder entzogen und die Anwendung muß erneut eine Zeitscheibe anfordern. Die jeweils auf eine neue Zeitscheibe

wartenden Anwendungen bekommen diese nach dem First-In First-Out (FIFO) Prinzip zugewiesen.

Durch die in der Ausführungszeit limitierten Zeitscheiben wird der verfügbare Prozessor fair unter den konkurrierenden Anwendungen verteilt. Somit wird der Durchsatz der Anwendungen optimiert und die Warte- und Turnaroundzeit sind deterministisch. Nachteilig wirkt sich das RR Verfahren auf I/O intensive Anwendungen aus. Während rechnenintensive Anwendungen die Zeitscheiben immer voll ausnutzen können werden die Zeitscheiben der I/O intensiven Anwendungen durch den Aufruf von Betriebssystemfunktionen in der Regel vor deren Ablauf unterbrochen.

Für den Einsatz des RR Verfahrens in den Client-Server Betriebssystem ist die Wahl der Zeitscheibe von entscheidender Bedeutung. Bei einer sehr kleinen Zeitscheibe von weniger als 100ms und einer Prozesswechelzeit von 80ms wäre die Prozessorauslastung denkbar schlecht, denn nur 20% der Zeit würden für die Ausführung der Anwendungen verbleiben. Bei steigenden Zeitscheibenlängen verbessert sich die Prozessorauslastung, aber gleichzeitig werden auch die Antwort- und die Turnaroundzeiten der Anwendungen verschlechtert.

5.2.3.4 Prioritätsbasiertes Scheduling

Bei den prioritätsbasierten Schedulingverfahren wird unterschieden in dynamische und statische Prioritäten. Statische Prioritäten werden bei der Generierung der Anwendung angegeben und können nicht vom jeweiligen Betriebssystem oder den Anwender verändert werden. Dynamische Prioritäten werden demgegenüber jeder Anwendung individuell zugewiesen. Die Zuteilung des Prozessors erfolgt abhängig von der Priorität, wobei die Anwendung mit der höchsten Priorität den Prozessor zugeteilt bekommt. Bei der nichtpreemptiven Variante erfolgt ein Prozesswechsel, sobald die Anwendung den Prozessor freiwillig oder durch ein Betriebssystemaufruf abgibt. Der Prozessor wird dann der wartenden Anwendung mit der höhsten Priorität zugewiesen. In der preemptiven Variante erfolgt ein Prozesswechsel sofort, wenn eine neue Anwendung in die Warteliste aufgenommen wird, die eine höhere Priorität besitzt als die derzeit ausgeführte Anwendung.

Das prioritätsbasierte Schedulingverfahren ermöglicht eine Verteilung der Prozessorzeit abhängig von der Wichtigkeit der jeweiligen Anwendungen. Durch diese Priorisierung besteht aber auch die Gefahr, daß Anwendungen mit niedriger Priorität nur sehr langsam oder gar nicht verarbeitet werden. Ein solches Verhalten ist gekennzeichnet durch ebenfalls sehr lange und variierende Turnaround-, Warte- und Antwortzeiten.

5.2.3.5 Multilevel-Feedback-Scheduling (MLFS)

Dieses Schedulingverfahren kombiniert mehrere Verfahren miteinander, um die jeweils entstehenden Nachteile zu vermindern. Dieses MLFS wird in vielen UNIX basierten Betriebssystemen eingesetzt und basiert auf dem zuvor beschriebenen Round-Robin Verfahren. Im Gegensatz zu dem RR Verfahren kommen hier mehrere Wartelisten zum Einsatz die unterschiedlich langen Zeitscheiben zugeordnet sind. Zu sehen ist dies in Abbildung 5.4. Die Verarbeitung innerhalb der Wartelisten erfolgt mit dem FCFS Verfahren. Anwendungen werden entsprechend ihres aktuel-

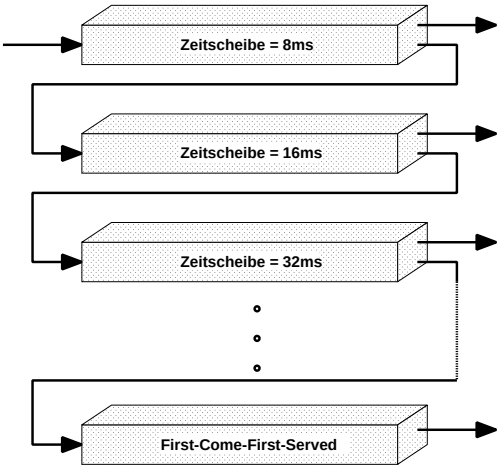


Abbildung 5.4: Wartelisten des Multilevel-Feedback-Scheduling.

len Rechenbedarfs aber auch abhängig von der Vergangenheit zwischen den einzelnen Wartelisten ausgetauscht. Diese Kombination von mehreren Schedulingverfahren erlaubt eine gerechte Prozessorverteilung unter sowohl rechenintensiven als auch I/O intensiven Anwendungen.

Durch die Verwendung der Zeitscheiben wird auch bei diesem Schedulingverfahren eine sehr gerechte Verteilung der Rechenzeit des Prozessors unter den konkurrierenden Anwendungen erreicht. Dies ist im direkten Vergleich zu einem reinen RR Verfahrens sogar noch besser, da I/O intensive Anwendungen besonders behandelt werden. Gekennzeichnet wird dieses kombinierte Verfahren durch einen guten Durchsatz, aber auch durch geringe Antwort- und Wartezeiten. Darüberhinaus ist auch hier eine prioritätsbasierte Verarbeitung der Anwendungen gegeben. Aufgrund des hohen Aufwands zur Verwaltung und zur permanenten Ermittlung der dynamischen Prioritäten ist die Prozessorauslastung jedoch nicht optimal. Unter Betrachtung des Client-Server Betriebssystems werden auf dem FPGA-Koprozessor nur rechenintensive Anwendungen ausgeführt, die eine aufwendige Unterscheidung in unterschiedliche Zeitscheiben nicht rechtfertigen.

5.2.4 Realisierte Schedulingstrategie

Die Auswahl der zum Einsatz kommenden Scheduling-Strategie ist sowohl von den Anforderungen an das Client-Server Betriebssystem als auch von den gegebenen Randbedingungen abhängig.

Eine Anforderung an das Betriebssystem ist die faire Verteilung der zur Verfügung stehenden Rechenzeit auf dem FPGA-Koprozessor unter den konkurrierenden Anwendungen. Eine weitere Forderung ist die dynamisch steuerbare prioritätsbasierte Ausführung der Anwendungen. Diese beiden Anforderungen sollen sicherstellen, daß die Ausführung wichtiger Anwendungen bevorzugt wird aber gleichzeitig andere weniger wichtige Anwendungen trotzdem verarbeitet werden.

Weitere Randbedingungen, die durch den FPGA-Koprozessor entstehen, sind zum einen die relativ lange Zeit von ≈80ms (siehe Messungen auf Seite 173) die zur Durchführung eines Prozesswechsels auf dem FPGA-Koprozessor notwendig ist und zum anderen die Tatsache, daß

aus Sicht des Host-Prozessors bei den Anwendungen auf dem FPGA-Koprozessor nur I/O Zugriffe durchgeführt werden. Im Gegensatz zu den I/O Zugriffen einer Host-Anwendung sind diese Zugriffe *nicht* geeignet um die Anwendung an dieser Stelle zu unterbrechen, denn die Daten werden sofort nach der Übertragung an den FPGA-Koprozessor verarbeitet. Im Vergleich dazu muß eine Host-Anwendung warten bis die angeforderten Daten verfügbar sind.

Unter Berücksichtigung einer hohen Prozessorauslastung ist bei den langen Prozesswechselzeiten auf dem FPGA-Koprozessor nur eine Einteilung in feste Zeitfenster sinnvoll. Eine solche feste Einteilung der Zeitscheiben ist bei dem Einsatz des Round-Robin Schedulingverfahrens gegeben, das somit die Basis für das Scheduling Verfahren des Client-Server Betriebssystems bildet. Eine Kombination mit dem Feedback Verfahren oder die Verwendung unterschiedlich langen Zeitscheiben, wie sie beim Multilevel-Feedback Scheduling zum Einsatz kommen, ist hier nicht notwendig, da aus Sicht des Client-Server Betriebssystems *nur* rechenintensive Anwendungen ausgeführt werden. Jede Anwendung auf dem FPGA-Koprozessor verwendet die gesamte Zeitscheibe und nur bei der Beendigung des Programms wird diese unter Umständen nicht voll in Anspruch genommen.

Die zweite Anforderung nach einer dynamischen prioritätsbasieren Ausführung wird durch die Einführung der sogenannten 'Multi-Slot' Strategie erreicht. Ziel dieser Strategie ist eine Verteilung der Rechenzeit die den Prioritäten der konkurrierenden Anwendungen entspricht. Grundlage dieser 'Multi-Slot' Strategie ist das zuvor ausgewählte Round-Robin Verfahren, das durch seine gerechte Rechenzeitverteilung gekennzeichnet ist. Aufgrund der langen Prozesswechselzeit wird bei dem neuen 'Multi-Slot' Verfahren nicht nach jedem Ablauf einer Zeitscheibe gewechselt, sondern nur nach einer prioritätsabhängigen Anzahl von Zeitscheiben. Dieses Verfahren, das nachfolgend noch detaillierter beschrieben ist, verhindert ein Abfallen der Prozessorauslastung durch übermäßig viele Prozesswechsel.

5.2.4.1 'Multi-Slot' Scheduling:

Das 'Multi-Slot' Verfahren basiert auf einer dynamischen Bewertung, die mit einem 'Aging' Verfahren[1] bestimmt werden. Bei jeder Anwendung die nach einer Wartezeit wieder zur Ausführung kommt, wird ein sog. Aging-Grundwert um einen Offset-Wert erhöht. Dieser Aging-Wert, der die Bewertung der Anwendung angibt, wird prioritätsabhängig bei dem Prozesswechsel und nach der Beendigung jeder Zeitscheibe verringert. Sobald der Aging-Wert den Grundwert von 1 unterschreitet wird er durch die nächste wartende Anwendung abgelöst und am Ende der Round-Robin Liste wieder angehängt. Durch die prioritätsabhängige Verringerung des Aging-Wertes wird die gesamte zur Verfügung stehende Rechenzeit entsprechend den Prioritäten aufteilt. Die Berechnung des Wertes, der von dem Aging-Wert subtrahiert wird, erfolgt mit der nachfolgenden Formel:

$$Aging{-}Decrement = \frac{1}{Priority} \quad , (Priority \in Z \;\; ; \;\; 0 < Priority \leq 10) \quad (5.1)$$

Durch dieses 'Multi-Slot' Verfahren wird eine Verteilung der Rechenzeit erreicht, die der jeweiligen Lastsituation und den einzelnen Prioritäten entspricht. Das Diagramm 5.5 zeigt einen Ausschnitt des Ablaufs dieses 'Multi-Slot' Round-Robin Verfahrens, das in dem Client-Server Betriebssystem Anwendung findet.

Die Wahl des Offset-Wertes, der beim Prozesswechsel abzüglich des prozessabhängigen Aging-Decrement zum Aging-Grundwert addiert wird, bestimmt die Anzahl der Zeitscheiben, die der Anwendung zur Verfügung stehen. Somit steuert dieser Offsetwert auch indirekt die Anzahl der durchgeführten Prozesswechsel und die daran gekoppelte Prozessorauslastung. Dieser Zusammenhang ist noch einmal graphisch in dem Diagramm 5.6 illustriert. Ist eine Zeitscheibe 200ms lang und dauert ein Prozesswechsel 80ms, so errechnet sich bei einem Offsetwert von 0.5 ein Systemoverhead von 12%. Bei einem Offsetwert von 2.0 beträgt dieser Systemoverhead nur noch 3.5%.

[1] Die 'Aging' Technik verringert dynamisch die Bewertung einer Anwendung abhängig von der Länge der Ausführungszeit auf dem Prozessor.

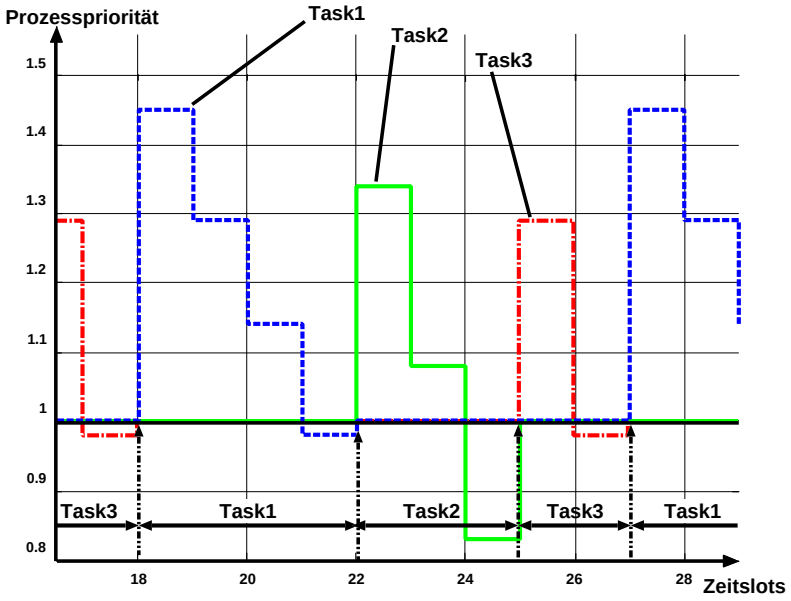


Abbildung 5.5: Ausschnitt der dynamischen Prozessbewertung mit dem 'Multi-Slot' Verfahren. *Gezeigt werden 3 Tasks mit folgenden Prioritäten: Task 1: Priorität 8;Task 2: Priorität 5;Task 3: Priorität 3.*

Die Wahl dieses Offsetwertes bestimmt zusammen mit der Länge der Zeitscheibe das Verhalten des Client-Server Betriebssystems bezüglich der in Abschnitt 5.2.3 genannten Kriterien. Das Erhöhen des Offsetwertes hat zur Folge, daß die Prozessorauslastung steigt, aber gleichzeitig wird auch der Durchsatz, und die Turnaround-, die Warte- und die Antwortzeit erhöht. Gleiche Kriterien gelten ebenfalls für die Verlängerung der Zeitscheiben, wobei diese sich an der durchschnittlichen Ausführungszeit der Anwendungen orientiert (siehe Abschnitt 4.1.1.2).

Unter den gegebenen Randbedingungen und den an das Client-Server Betriebssystem gestellten Anforderungen ist die hier vorgestellte und realisierte 'Multi-Slot' Scheduling-Strategie am besten geeignet, um die

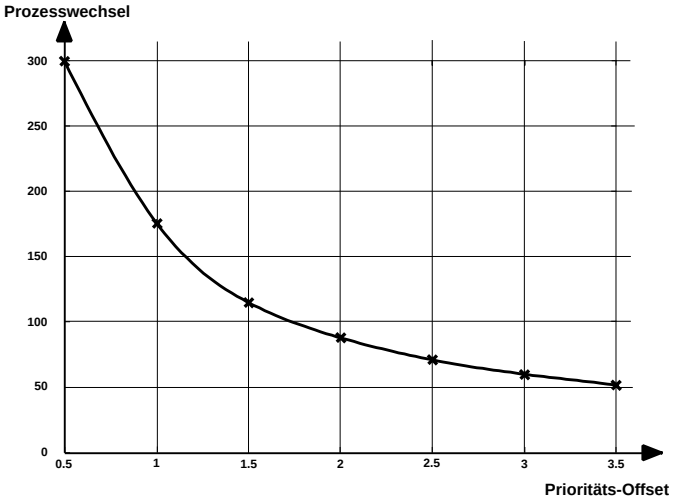


Abbildung 5.6: Das Diagramm stellet die Anzahl der Prozesswechsel in Abhängigkeit des Offset-Wertes dar. *Die Werte wurden für drei Anwendungen mit Prioritäten 8, 5, und 3 berechnet.*

jeweils nächste auszuführende Anwendung auszuwählen. Hinzu kommt noch, daß die Bestimmung sehr schnell durchgeführt werden kann und somit den Systemoverhead nicht weiter vergrößert. Durch das 'Multi-Slot' Verfahren ist das Client-Server Betriebssystem in der Lage, die Prioritäten der einzelnen Anwendungen zu berücksichtigen und gleichzeitig einer niedrigen Prozessorauslastung durch die langen Prozesswechselzeiten entgegenzuwirken.

5.3 Zielplattform

Für den Test des Client-Server Betriebssystems kommt ein FPGA-Koprozessor der Firma Silicon Software zum Einsatz. Bei dieser Einsteckkarte handelt es sich um einen klassischen FPGA-Koprozessor, der über den PCI-Bus mit dem Host-Prozessor verbunden ist. Der FPGA-Ko-

prozessor ist bestückt mit einem PCI-Interfacebaustein, einem FPGA-Baustein XV400 der Firma Xilinx, einem synchronen RAM-Block, einem SDRAM-Stecker und einer externen Steckverbindung. Die schematischen Blöcke und deren Verbindungen sind in Abbildung 5.7 dargestellt.

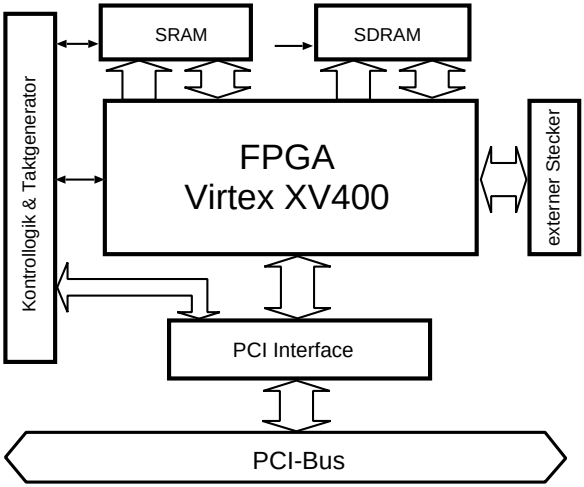


Abbildung 5.7: Blockschaltbild des FPGA-Koprozessor microEnable II.

Der Einsatz dieses FPGA-Koprozessors ist verbunden mit den in Abschnitt 3.4.2 gestellten Anforderungen nach einer schnellen Konfigurations- und Readbackschnittstelle, einer Kontrolle über die Takterzeugung und einen Zugriff auf die lokalen SRAM-Bänke. Im nachfolgenden wird die Realisierung dieser drei Anforderungen im Detail beschrieben und deren Auswirkung auf das Client-Server Betriebssystem dargestellt. Weitere Eigenschaften des FPGA-Koprozessor sind nur sekundär von Bedeutung und werden daher nicht weiter erläutert. Sie können in [NL99] nachgelesen werden.

Konfigurations- und Readbackschnittstelle: Zur Durchführung der Konfiguration und des Readback steht dem Client-Server Betriebssystem die 8Bit breite 'SelectMap' Schnittstelle des FPGA-Bau-

steins zur Verfügung, die auf der microEnable II mit maximal 40MHz betrieben werden kann. Dies ermöglicht eine theoretische Konfigurationszeit von nur 7, 8ms. Gleiches gilt für den Readback.

Takterzeugung: Aufgrund des allgemeinen Charakters dieses FPGA-Koprozessor ist es *nicht* möglich die Taktgeneratoren durch ein spezielles Signal gesteuert abzuschalten, um die FPGA-Schaltung anzuhalten. Diese nichtvorhandene Steuerungsmöglichkeit limitiert die auszuführenden Anwendungen auf statische Schaltungen, die intern abschaltbar sind, bzw. sich statisch Verhalten.

RAM-Zugriffe: Auf dem FPGA-Koprozessor microEnable II ist kein direkter Zugriff auf das lokale synchrone RAM möglich, so daß eine Sicherung des RAM-Inhalts nur durch eine entsprechende FPGA-Schaltung ermöglicht wird.

Somit muß das Client-Server Betriebssystem nach dem Sichern des FPGA-Status eine weitere FPGA-Schaltung geladen werden, die die Sicherung und Rekonstruktion des RAM-Inhalts erlaubt. Alternativ dazu kann das Client-Server Betriebssystem auf die Sicherung des RAM-Inhalts verzichten, wodurch nur maximal eine Anwendung dieses RAM ohne einen Datenverlust nutzen kann.

Trotz der nicht erfüllten Anforderungen, die zum Betrieb des Multitasking-Betriebssystems notwendig sind, kann dieser FPGA-Koprozessor als Zielplattform verwendet werden, um die Messungen des Client-Server Betriebssystems durchzuführen. Aus diesem Grunde wurden das Client-Server Betriebssystem für diesen FPGA-Koprozessor umgesetzt.

5.4 Modularchitektur

Eine weitere Anforderung, die bei der Ausführung des Client-Server Betriebssystemens zu beachten ist, ist die Portabilität des Betriebssystems. Zur Einhaltung dieser Anforderung werden die notwendigen Teilkomponenten des Client-Server Betriebssystems in mehrere Module unterteilt.

Ziel dieser Modularisierung ist die Erleichterung der Portierung auf andere Host-Betriebssysteme, aber auch die Portierung auf andere FPGA-Koprozessor. Ein weiteres Ziel der Modularisierung liegt in der Möglichkeit der Wiederverwendung der Module in anderen Projekten und der einfacheren Verifikation der Funktionalität der Modulfunktionen.

Jedes dieser Module faßt Funktionen zusammen, die einer funktionalen Bindung unterliegen. So werden z.B. alle Funktionen der Interprozess Kommunikation (IPC) in einem eigenen Modul zusammengefaßt, so daß bei der Portierung von einem Host-Betriebssystem zum anderen nur dort Anpassungen durchgeführt werden müssen.

Das gesamte Client-Server Betriebssystem unterteilt sich in acht Module die jeweils aufeinander aufbauen. Abbildung 5.8 zeigt die einzelnen Module mit ihren jeweiligen Verbindungen. Wie in der Darstellung zu erkennen ist, baut der Server auf den hardwarenahen Modulen auf, die aus einem Zeitgeber (RTC), dem FPGA-Koprozessor mit den entsprechenden Modulen aber auch aus der Bitstrom-Bibliothek besteht. Diese Module stellen den Server die benötigte Funktionalität zum Steuern des FPGA-Koprozessor zur Verfügung. Auf der rechten Seite ist der Client dargestellt, der die API-Schnittstelle zu den einzelnen Anwendungen bereitstellt. Verbunden sind Client und Server über das Interprozess Kommunikation Modul (ICP), das in beiden Modulen verwendet wird.

Hier sei noch einmal auf die leichte Portierbarkeit dieser Modularchitektur hingewiesen. Wie die Abbildung zeigt, ist man in der Lage, z.B. durch das Austauschen bzw. das Erweitern der Bitstromarchitektur Module, schnell und einfach neue FPGA-Bausteine zu unterstützen, ohne den gesamten Server verändern zu müssen. In gleicher Weise ist auch die Portierung auf andere FPGA-Koprozessoren oder andere Betriebssysteme einfach durchzuführen.

Im nachfolgenden werden die einzelnen Module in ihrer Funktionalität und ihrem Aufbau kurz erläutert.

5.4.1 Real-Time Clock

Die Real-Time Clock (RTC) ist ein Zeitgeber der von dem Host-Betriebssystem zur Verfügung gestellt wird. Die RTC wird herangezogen, um ein

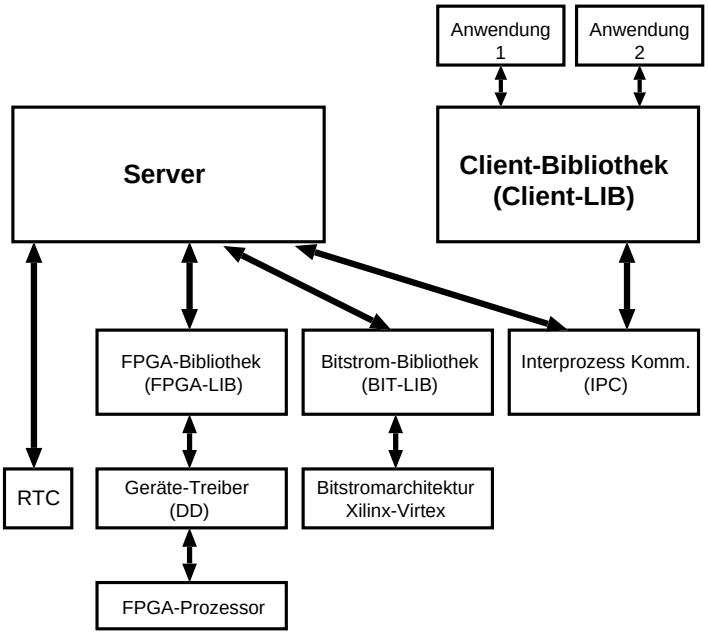


Abbildung 5.8: Modularchitektur des Client-Server Betriebssystems mit den einzelnen Modulen und deren Zusammenhang untereinander.

periodisches Signal zu erzeugen, das die Zeitscheiben des Round-Robin Schedulingverfahrens vorgibt. Die unter Linux zum Einsatz kommende RTC erlaubt eine Taktgenerierung zwischen 8KHz und 2Hz, das durch entsprechende Teiler noch weiter in seiner Frequenz verringert werden kann. Dieses periodische Signal wird direkt im Server-Modul verarbeitet und steuert somit die Ausführungshäufigkeit des Client-Server Betriebssystems.

5.4.2 Gerätetreiber

Der Gerätetreiber (DD) ermöglicht alle Zugriffe auf den FPGA-Koprozessor. Dafür stellt der DD der darüberliegenden FPGA-Bibliothek alle Funktionen zur Verfügung mit denen die Register auf dem FPGA-Koprozessor angesprochen werden können. Darüberhinaus regelt der DD auch das Einblenden von Speicherbereichen des FPGA-Koprozessors in die Prozessumgebung auf dem Host-Prozessor, übernimmt die Steuerung des PCI-Interfacebausteins, fängt die Interrupts des FPGA-Koprozessor ab und stellt Funktionen bereit, die einen effektiven Datentransfer per DMA ermöglichen.

Zum Einsatz kommt hier ein kommerzielles Produkt[Jun00], das diese Funktionalität für eine Vielzahl von PCI-Interfacebausteinen bereitstellt. Dieser DD ist nicht nur für viele PCI-Karten einsetzbar, sondern unterstützt auch noch mehrere Betriebssysteme wie z.B. Linux, Win98, WinNT oder Solaris. Diese vielfältige Unterstützung und die damit verbundene einfache Portierbarkeit ist der Grund für den Einsatz dieses Gerätetreibers.

5.4.3 FPGA-Bibliothek

Aufbauend auf dem Gerätetreiber stellt die FPGA-Bibliothek die notwendigen Funktionen zur Steuerung der einzelnen Komponenten auf den FPGA-Koprozessor bereit. Im Gegensatz zu dem Gerätetreiber, der nur den Zugriff auf die verfügbaren Register ermöglicht, fassen die Funktionen meist mehrere Registerzugriffe zusammen, um z.B. den FPGA-Baustein zu konfigurieren. Unterteilt ist das FPGA-Bibliotheks Modul meist in die folgenden funktionalen Untergruppen:

- Konfiguration und Readback des FPGA-Bausteins
- Datenübertragung zwischen FPGA-Koprozessor und Host-Speicher
- Einstellen der Takterzeugung
- Abfrage und Kontrolle von Signalen

Die Schnittstelle, die die FPGA-Bibliothek dem Server bereitstellt, ist unabhängig von dem verwendeten FPGA-Koprozessor, so daß eine Portierung des gesamten Client-Server Betriebssystems auf andere FPGA-Koprozessoren einfach durchgeführt werden kann.

5.4.4 Bitstrom-Bibliothek

Die Bitstrom-Bibliothek stellt dem Server ebenfalls eine Schnittstelle zur Verfügung die alle Funktionen zur Zustandsrekonstruktion einer FPGA-Schaltung beinhaltet (siehe auch Abschnitt 3.5). Zur Unterstützung der Portierbarkeit sind die Funktionen der einzelnen, unterstützten Bausteinfamilien in eigene Module zusammengefaßt, die gegeneinander ausgetauscht werden können.

Notwendig wird diese Kapselung, da die Bitstromarchitektur nicht nur zwischen den einzelnen FPGA-Herstellern, sondern auch innerhalb der Produktfamilien stark voneinander abweichen können.

5.4.5 Interprozess Kommunikation

Die Interprozess Kommunikation (IPC) dient der Synchronisation und dem Austausch der Kommandos zwischen den Clients und dem Server. Bei der Auswahl des IPC stand auch die Anforderung an die Portierbarkeit im Vordergrund, denn IPC System V ist ein standardisierter Kommunikationsmechanismus, der auf allen relevanten Betriebssystemen zur Verfügung steht.

Das Modul, das die gesamte IPC Kommunikation bereitstellt, wird sowohl vom Client als auch vom Server verwendet, um den Datenaustausch zu realisieren. Die Verwendung des gleichen Moduls ist zum einen praktisch, denn die Entwicklung ist nur einmal durchzuführen, aber auch notwendig, da beide – Server und Clients – die gleichen Datenformate verwenden und somit eine zuverlässige Kommunikation sichergestellt ist.

5.4.6 Client

Das Client Modul bildet die standardisierte Schnittstelle zu den Anwendungen, die unabhängig von dem verwendeten FPGA-Koprozessor und

des zum Einsatz kommenden Host-Betriebssystem ist. Diese Schnittstelle selbst ist untergliedert in folgende Funktionsgruppen:

- Ab- und Anmelden am Server
- Kontrolle über den FPGA-Koprozessor und die Takterzeugung
- Laden, Konfiguration und Readback des FPGA-Bausteins
- Datenaustausch zwischen Anwendung und FPGA-Koprozessor

Ausgeführt ist diese Schnittstelle als eine Funktionsbibliothek, die in die entsprechende Anwendung integriert wird. Beim Aufruf der Funktionen werden dann entsprechende Datenpakete zwischen Client und Server ausgetauscht und die Funktionen auf dem Server ausgeführt.

5.4.7 Server

Der Server selbst ist die zentrale Einheit des Client-Server Betriebssystems, daß die Kommunikation mit dem jeweils aktiven Client übernimmt, die nächste auszuführende Anwendung mittels des 'Multi-Slot' Schedulingverfahrens auswählt und die realen Zugriffe auf den FPGA-Koprozessor durchführt.

Wie aus der Abbildung 5.8 zu entnehmen ist, verwendet der Server dazu die Funktionalität der folgenden Module: Interprozess Kommunikation, Bitstrom-Bibliothek, FPGA-Bibliothek und RTC. Durch diesen gewählten Schichtenaufbau ist der Server selbst vollkommen unabhängig von dem verwendeten Host-Betriebssystem und dem zum Einsatz kommenden FPGA-Koprozessors, was eine Portierung des gesamten Client-Server Betriebssystems vereinfacht. Eine detaillierte Beschreibung des Aufbaus und der Funktionsweise des Servers ist in dem folgenden Abschnitt dargestellt.

5.5 Funktionsweise

In dem vorhergehenden Abschnitt wurden die einzelnen Module beschrieben, die zum Aufbau des Servers und des Clients notwendig sind. Dieser

Abschnitt beschreibt den funktionellen Ablauf innerhalb des Servers und des Clients, der bei der Verarbeitung der Anwendungen ausgeführt wird.

Der Server selbst ist im Gegensatz zu anderen Servern, wie z.B. einem Web-Server [Eil00] oder einem Datenbankserver [DG00], auf zwei permanent laufende Threads[2] begrenzt. Dies ist möglich, da der Server immer nur eine Anwendung zu einem Zeitpunkt verarbeitet. Getrennt wird dabei in einen Thread der für das periodische Scheduling zuständig ist, und einen Thread der die Kommandos des jeweilig aktiven Threads verarbeitet. Beide Serverthreads und der Ablauf aus Sicht des Clients werden in den nachfolgenden Abschnitten detailliert erläutert.

5.5.1 Scheduling-Thread

Hauptaufgabe des Scheduling-Threads ist die Auswahl der Anwendung die innerhalb der nächsten Zeitscheibe verarbeitet wird. Zusätzlich gehört aber auch das Abmelden eines beendeten Clients und das Anmelden eines neuen Clients zu den Aufgaben dieses Threads.

Der gesamte Ablauf des Threads ist in der Abbildung 5.9 dargestellt. Der Scheduling-Thread wird, angestoßen von dem periodischen Signal der Real-Time Clock, in regelmäßigen Zeitabständen ausgeführt. Diese Zeitabstände entsprechen der Länge der Zeitscheibe und können entsprechend eingestellt werden. Nach dem Start prüft der Scheduling-Thread ob ein neuer Client sich anmelden oder ein beendeter Client sich abmelden will. Diese beiden Operationen sind somit nur synchron zu der Betriebssystemausführung möglich. Sobald kein Client sich an- bzw. abmelden will, wird mittels des 'Multi-Slot' Schedulingverfahrens der Aging-Werte der gerade aktiven Anwendung verringert und mit dem Aging-Grundwert der anderen Anwendungen verglichen. Bei einem angezeigten Prozesswechsel wird dann die laufende Anwendung angehalten und gesichert, bevor die wartende Anwendung zur Ausführung vorbereitet wird. Die detaillierte Vorgehensweise bei einem Prozesswechsel ist in Abschnitt 3.7 beschrieben.

[2] Ein Thread ist ein nebenläufiger Prozess, der in dem Adress- und Datenraum der Anwendung ausgeführt wird.

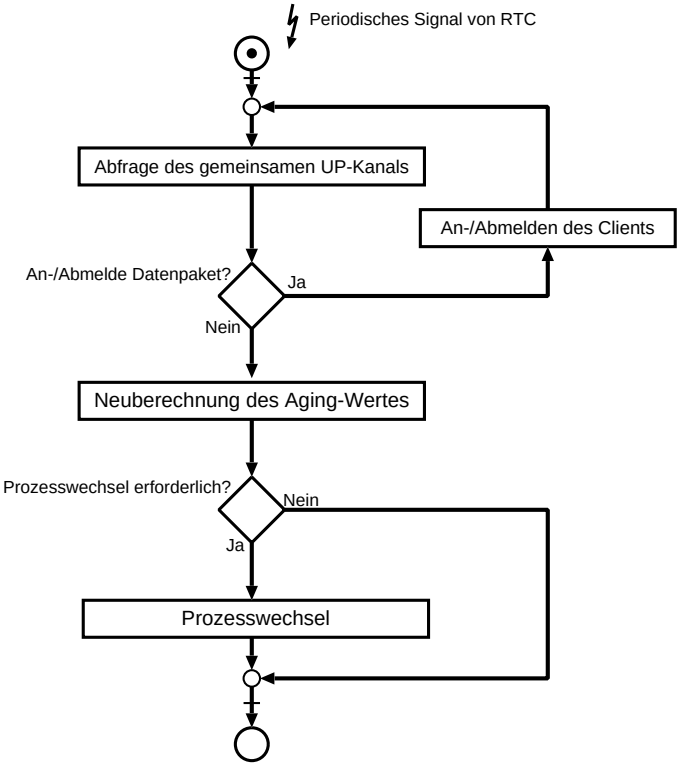


Abbildung 5.9: Ablaufdiagramm des Scheduling-Threads mit den entsprechenden Funktionsblöcken.

Innerhalb dieses Scheduling-Threads werden die gesamten Informationen aller Anwendungen gespeichert, um bei der Aktivierung diese an den Kommando-Thread weiterzugeben. Zur Sicherung und zur Synchronisation zwischen dem Scheduling- und dem Kommando-Thread werden auch hier IPC Semaphoren eingesetzt. Diese Synchronisation ist zum einen für die Übertragung der Anwendungsdaten aber auch zur Unterbrechung des Kommando-Threads notwendig, wenn keine Anwendung für die Verarbeitung bereitstehen.

5.5.2 Kommando-Thread

Die Aufgabe dieses Kommando-Threads ist die Ausführung der Clientkommandos auf dem FPGA-Koprozessor. Die folgende Abbildung 5.10 zeigt den inneren Teil des Kommando-Threads, der während der Ausführung eine Anwendung durchlaufen wird.

Der gesamte Kommando-Thread wird in einer Endlosschleife verarbeitet, die gesteuert durch eine Sempahore nur durchlaufen wird, wenn mindestens eine Anwendung verarbeitet werden kann. Dieses Vorgehen verhindert das Durchlaufen der äußersten Endlosschleife ohne das eine Anwendung auf den FPGA-Koprozessor zugreift und minimiert somit den Einfluß des Client-Server Betriebssystems auf andere Anwendungen auf dem Host-Prozessor. Nach der Anmeldung eines Clients und dem Laden einer FPGA-Schaltung kann diese vom Scheduling-Thread konfiguriert werden. Sobald der FPGA-Baustein konfiguriert ist übernimmt der Kommando-Thread die Kommunikation mit dem aktivierten Client und verarbeitet dessen Kommandos. Nach dem Empfang des Kommandos wird dieses analysiert und entsprechend dem Kommando wird der notwendige Zugriff auf den FPGA-Koprozessor durchgeführt. Das Ergebnis des Zugriffs wird anschließend in einem Antwortpaket an den Client zurückgesendet.

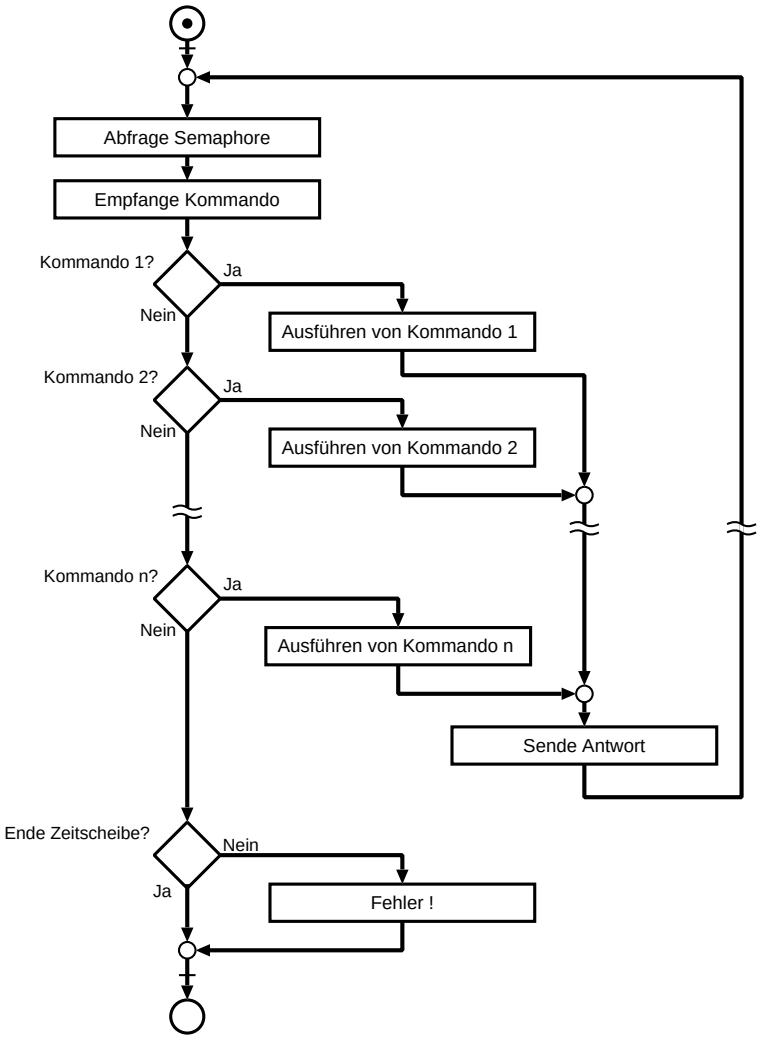


Abbildung 5.10: Ablaufdiagramm des Kommando-Threads mit den entsprechenden Funktionsblöcken. *Dargestellt ist hier nur die innere Schleife, die innerhalb der Endlosschleife die Clientkommandos verarbeitet.*

Diese Ausführung der Kommandos wird unterbrochen, indem der Scheduling-Thread mit einem speziellen Kommando den Kommando-Thread unterbricht, um die Anwendung zu wechseln. Nach Beendigung des Prozesswechsels erhält der Kommando-Thread über die Semaphore ein erneutes Startsignal und beginnt wieder mit der Kommandoausführung.

5.5.3 Client

Aus Sicht des Clients bzw. der Anwendung ergibt sich keine Veränderung, denn die gesamte Kommunikation zwischen dem Client und dem Server ist in den Funktionen der Client-Bibliothek gekapselt.

Nach Aufruf einer Funktion wird ein entsprechnedes Kommandopaket generiert und an den Server übermittelt. Nach der Ausführung durch das Client-Server Betriebssystem wird das Ergebnis in einem weiteren Antwortpaket an den Client zurückgesendet und über den Rückgabewert der Funktion an die Anwendung übergeben.

5.6 Messungen

Zur Bewertung der Leistungsfähigkeit des entwickelten Client-Server Betriebssystems wurden verschiedene Messungen auf einem Computersystem mit integrierten FPGA-Koprozessor microEnable II durchgeführt.

Ein Teil der Messungen wurde zur Ermittlung der grundlegenden Leistungsfähigkeit der PCI-Datenübertragung durchgeführt. Die Konfiguration und der Readback basieren auf dieser PCI-Datenübertragung und bilden zusammen mit den Zeiten, die für die Bitstromrekonstruktion benötigt werden, die erreichbare Prozesswechselzeit des Systems.

Die daran anschließenden Messungen wurden unter Anwendung des Client-Server Betriebssystems durchgeführt, um dessen Leistungsfähigkeit während des Betriebs zu ermitteln. Zur Bestimmung des systematischen Overheads, der durch die Verwaltung der Anwendungen und die Prozesswechsel entsteht, wurde der Kommunikationsoverhead, die Prozesswechselzeiten und die Länge der Zeitscheiben gemessen. Abschließend wurde in einer weiteren Meßreihe der gesamte Einfluß auf eine Gruppe von Anwendungen bestimmt.

5.6.1 Messaufbau

Alle nachfolgenden Messungen wurden auf einem Computersystem durchgeführt, das über die folgenden Eigenschaften verfügt:

- Pentium I Prozessor mit 166MHz und 256MByte Speicher,
- einem MicroEnableII FPGA-Koprozessor zusammen mit 2 weiteren PCI Karten (Grafik- und Netzwerkkarte) angeschlossen am PCI-Bus,
- kommerzieller Gerätetreiber (WinDriver),
- Linux Host-Betriebssystem (Kernel Version 2.2.14)

Alle Messungen wurden, soweit dies möglich war, ohne weitere auf dem System laufende Anwendungen bzw. Prozesse durchgeführt, um unverfälschte Ergebnisse zu erhalten. Zur Ermittlung der Ausführungszeiten wurde der im Prozessor integrierte Taktzähler verwendet, der über eine Nanoskunden genaue Auflösung verfügt und somit auch sehr kleine Zeitmessungen zulässt.

5.6.2 PCI-Datenübertragung

Die theoretische Datenübertragungsrate über den PCI-Bus beträgt 132 MByte/s. Erreicht werden diese hohen Datentransferraten durch den Einsatz von DMA-Transfers, die ohne den Host-Prozessor direkt in den Host-Speicher schreiben oder Daten auslesen. Aufgrund des Einsatzes des DMA-Transfers sowohl für die Konfiguration als auch für den Readback des FPGA-Bausteins besteht ein direkter Zusammenhang zwischen der maximalen Datentransferrate und der Prozesswechselzeit.

Wie schon 1997 von Mark Shand [MS97] gezeigt wurde, ist die maximal erreichbare Datentransferrate abhängig von den jeweiligen verwendeten Komponenten[3] und kann somit von einem zum anderen Rechner

[3]Die maximale Datentransferrate wird hauptsächlich von dem verwendeten Chipsatz, aber auch von der Grundlast des PCI-Buses bestimmt. Die Grundlast faßt den Datentransfer von anderen PCI-Karten zusammen.

stark schwanken. Aufgrund des hohen Einflußes der maximalen Datentransferrate auf die Prozesswechselzeit ist es wichtig diese für das oben beschriebene Computersystem zu ermitteln. Nur so wird eine spätere Bewertung des Client-Server Betriebssystems möglich.

Zur Bestimmung der Datentransferrate wird die Zeit gemessen, die von dem Aufruf der DMA-Funktion bis zu dessen Rückkehr vergeht. Dadurch werden neben der eigentlichen Übertragung über den PCI-Bus auch die notwendigen Vorbereitungsschritte, wie das Einlagern der Speicher-Pages und den Aufbau einer Scatther-Gather Liste, mit berücksichtigt. Insgesamt wurden drei verschiedene DMA-Transfers vermessen: Direkter DMA-Transfer aus dem Speicher, ein DMA-Transfer mit vorherigen kopieren des Speichers und der im Client-Server Betriebssystem implementierte DMA-Transfer.

Direkter DMA-Transfer: Der direkte DMA-Transfer aus dem Speicher wird verwendet, um z.B. den Konfigurationsbitstrom zum FPGA-Baustein zu übertragen. Bei dieser Übertragung werden die Daten direkt im Speicher vorbereitet und anschließend über den PCI-Bus gesendet. Aufgrund des direkten Speicherzugriffs erreicht diese Übertragungsart die beste Datenrate, die gleichbedeutend mit der für das Computersystem maximal erreichbaren Datenrate ist.

Memcopy und DMA-Transfer: Der zweite DMA-Datenübertragung kopiert die zu übertragenden Daten zuvor in einen eigenen Speicherbereich, um sie dann von diesem zweiten Speicher per direktem DMA-Transfer zu übertragen. Diese aufwendigere Art der Übertragung ist bei dem eingesetzten Gerätetreiber notwendig, um Daten aus einem Shared-Memory Element übertragen zu können. Verwendet werden die Shared-Memory Elemente unter anderem für den effektiven Datenaustausch zwischen Client und Server.

Server DMA-Transfer: Bei der dritten DMA-Datenübertragung wurde die dazu benötigte Zeit mit einem, an den Server angeschlossenen Client gemessen. Diese Messung wurde durchgeführt, um die Abhängigkeit des Kommunikationsoverheads auf die Paketgröße zu ermitteln.

Zu jeder der drei oben aufgeführten Übertragungsarten wurden Messungen mit einer variierenden Datenblocklänge durchgeführt. Gemessen wurden die jeweils benötigte Ausführungszeite und die sich daraus ergebenden Datenübertragungsraten. Die Abbildung 5.11 zeigt die Ausführungszeiten für die Read-Zugriffe und die Abbildung 5.12 für die Write-Zugriffe. Die entsprechenden Datenübertragungsraten sind in Abbildung 5.13 für den Read-Zugriff und in Abbildung 5.14 für den Write-Zugriff abgebildet.

Bei Betrachtung der beiden Ausführungszeitdiagramme 5.11 und 5.12 wird deutlich, daß die DMA-Funktion des Gerätetreibers einen Overhead von $\approx 350\mu$s benötigt. Desweiteren ist zu sehen, daß der Kommunikationsoverhead zwischen Server und Client weiter 100μs in Anspruch nimmt. Aufgrund des konstanten Offsets macht sich dieser Overhead nur bei sehr kleinen Datenpaketen bemerkbar und kann bei Übertragung von mehr als 5kByte vernachlässigt werden.

Die Auswirkungen bei der Übertragung von Daten aus einem Shared-Memory Element sind dagegen mit einer Verdoppelung der Ausführungszeit verbunden. Somit halbiert sich die zur Verfügung stehende maximale Datentransferrate, sobald Daten von dem Client zum Server transferiert werden. Dieser Zusammenhang ist leicht in den Abbildungen 5.13 und 5.14 zu sehen. Es gilt zu beachten, daß diese Halbierung der Datentransferrate auf den eingesetzten Gerätetreiber zurückzuführen ist und das dieses Problem durch eine zukünftige Version behoben wird.

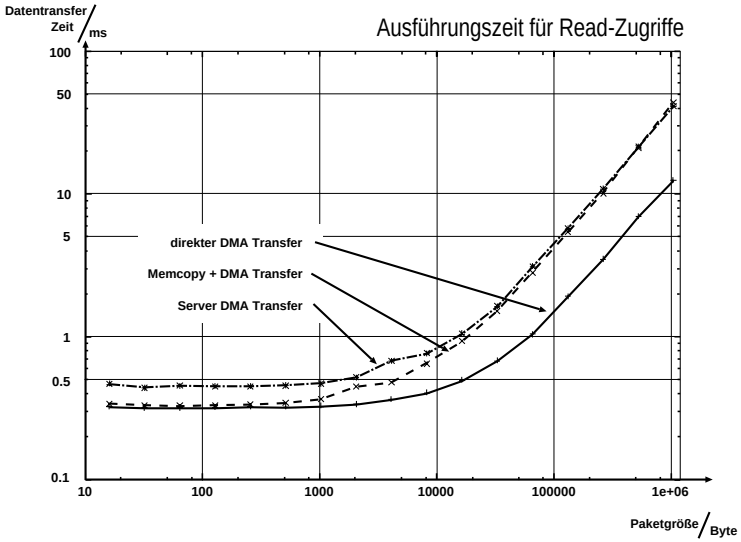


Abbildung 5.11: Abhängigkeit der Ausführungszeit von der zu übertragenden DMA Blockgröße.

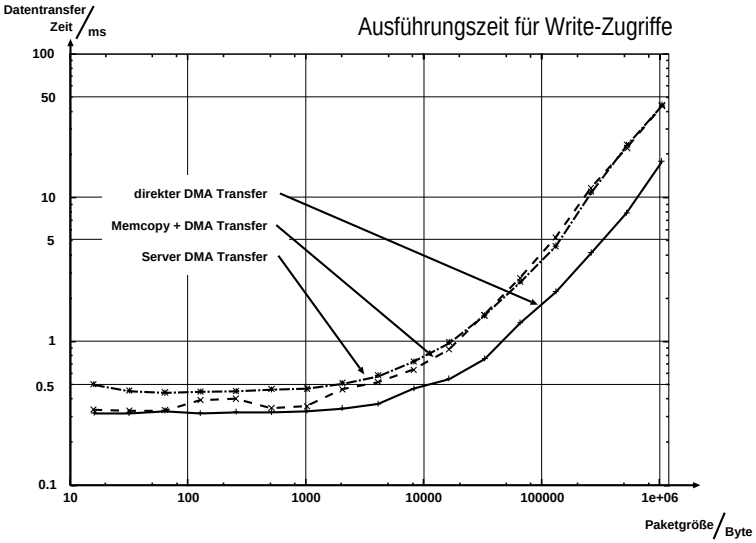


Abbildung 5.12: Abhängigkeit der Ausführungszeit von der zu übertragenden DMA Blockgröße.

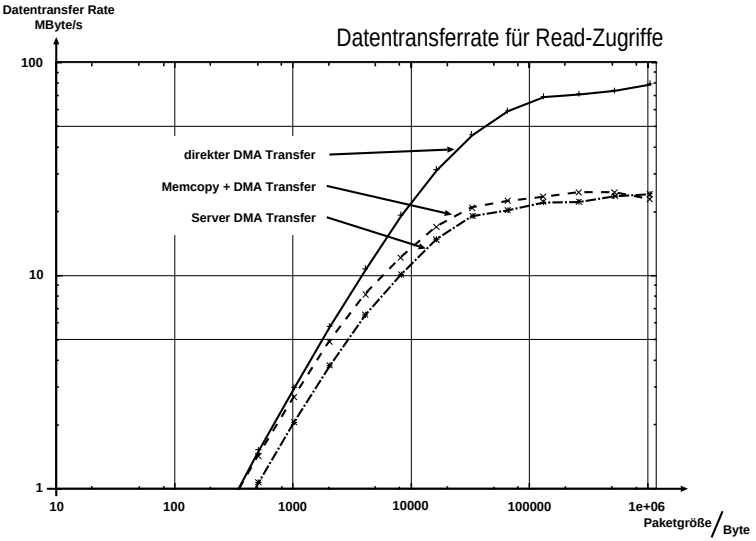


Abbildung 5.13: Abhängigkeit der Datentransferrate von der zu übertragenden DMA Blockgröße.

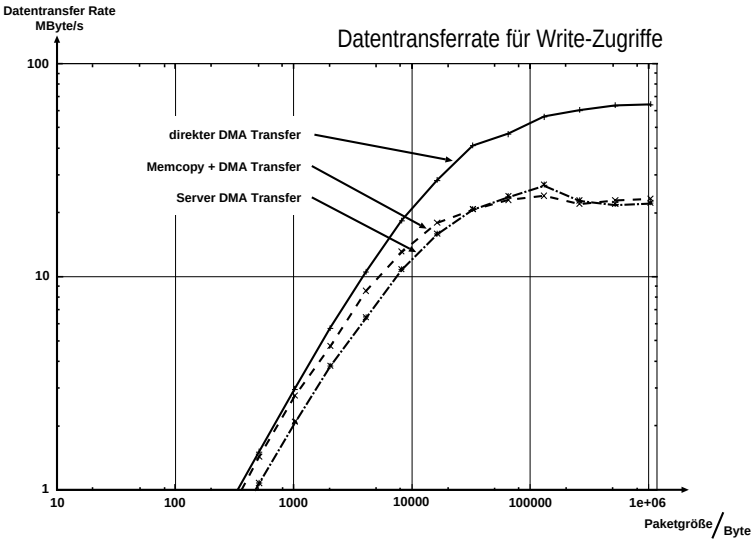


Abbildung 5.14: Abhängigkeit der Datentransferrate von der zu übertragenden DMA Blockgröße.

Zusammenfassend haben die Messungen ergeben, daß die maximal erreichbare Datentransferrate für das oben beschriebene Computersystem inklusive des FPGA-Koprozessors bei ≈80MByte/s für Read-Zugriffe und bei ≈65MByte/s für Write-Zugriffe liegt. Diese Datenraten halbieren sich aufgrund des Gerätetreibers, sobald Daten aus einem Shared-Memory Element übertragen werden. Unabhängig von der Übertragungsart ist der Einfluß der Datenblockgröße auf die zu erreichende Datentransferrate, die mit zunehmender Blockgröße ebenfalls zunimmt.

Darüberhinaus bleibt festzustellen, daß der Kommunikationsoverhead durch den Datenaustausch zwischen Client und Server ≈100μs beträgt und somit nur bei der Übertragung von kleinen Datenblöcken berücksichtigt werden muß.

Theoretisch ist aufgrund der gemessenen maximalen Datentransferrate von nur 80 bzw. 65MByte/s keine Veränderung der Prozesswechselzeit zu erwarten. Die zu übertragenden Konfigurationsdaten betragen ≈310kByte für den eingesetzten Virtex XV400 FPGA-Baustein und werden bei 65MByte/s in ≈5ms zum PCI-Interfacebaustein übertragen. Dort werden die 32Bit Datenworte in 4 Byteworte umgesetzt und mit einer Frequenz von 40MHz zum FPGA-Baustein übertragen. Diese zweite, byteweise Übertragung limitiert die Konfiguration daher auf eine theoretische Konfigurationszeit von ≈8ms.

5.6.3 Client-Server Leistungsfähigkeit

Die gesamte Leistungsfähigkeit des Client-Server Betriebssystems wird bestimmt durch den systembedingten Overhead, der innerhalb des Betriebssystems, aber auch durch den Datenaustausch zwischen Client und Server entsteht.

Dieser Abschnitt beschreibt kurz die Messungen, die zur Bestimmung dieses systembedingten Overheads durchgeführt wurden und bewertet die gemessenen Zeiten. Im einzelnen wurden die Zeiten für die Kommunikation zwischen Client und Server, die einzelnen Prozesswechsel und die Abweichungen von der definierten Zeitscheibe gemessen. Zur gesamten Beurteilung wurden abschließend noch weitere Messungen aus Sicht des Clients durchgeführt, die den Einfluß der Prozesswechsel auf die Ausführungsgeschwindigkeit der Anwendungen verdeutlichen.

5.6.3.1 Kommunikationsoverhead

Zur Übermittlung der einzelnen Kommandos zwischen Client und Server werden sog. IPC Message-Queues verwendet. Diese Message-Queues bieten eine hohe Flexibilität für die Portierung. Gleichzeitig wirken sie sich aber auch nachteilig auf die Ausführungszeit der Kommandos aus, denn die benötigt Zeit für die Übertragung muß zu der eigentlichen Ausführungszeit auf dem Server hinzuaddiert werden. Zur Bestimmung dieses Kommunikationsoverheads zwischen den Clients und dem Server wurde ein spezielles Kommando eingeführt, das auf dem Server keine Funktion ausführt und nur das Antwortpaket an den Client zurücksendet. Eine Messung der Ausführungszeit dieses speziellen Kommandos bestimmt die Zeit, die für die Kommunikation benötigt wird.

Mit diesem Versuchsaufbau wurden jeweils eine Messung durchgeführt, bei der nur ein Client mit den Server kommuniziert und in einer zweiten Messung kommunizierten drei Clients gleichzeitig mit dem Server. Die Ergebnisse dieser Messungen sind in Tabelle 5.1 zusammengestellt.

Messung	Kommunikations-overhead
ein Client	60μs
drei Clients	63μs

Tabelle 5.1: Kommunikationsoverhead zwischen den Clients und dem Server für jeweils ein Kommando.

Mit $\approx 60\mu$s hat die Kommunikation zwischen Client und Server nur einen sehr geringen Einfluß auf die gesamte Ausführungszeit der Anwendung, da diese in der Regel nur wenige Kommandos benötigen.[4] Desweiteren ist es zur effektiven Nutzung des FPGA-Koprozessors notwendig, eine möglichst hohe Datenaustauschrate mit dem Koprozessor zu erreichen. Dies ist jedoch nur bei der Übertragung von entsprechend großen Datenpaketen zwischen Client-Anwendung und FPGA-Koprozessor gegeben. Dargestellt ist dieser Zusammenhang auch in den Abbildungen

[4] Siehe auch das Beispiel der Bildfilterung auf Seite 129, das mit insgesamt nur 11 Kommandos komplett verarbeitet wird.

5.11 und 5.12. Aufgrund der mit der Datenmenge steigenden Übertragungszeiten ist der Einfluß des Kommunikationsoverheads von $\approx 60\mu$s nur bei sehr kleinen Datenmengen bemerkbar und kann im allgemeinen vernachlässigt werden.

5.6.3.2 Prozesswechseloverhead

Bei einem Wechsel der zu verarbeitenden Anwendung ist es notwendig den aktuellen Status der verdrängten FPGA-Schaltung mit Hilfe des Readbacks zu sichern und die FPGA-Schaltung der neu ausgewählten Anwendung wiederherzustellen und anschließend den FPGA-Koprozessor zu konfigurieren. Die für die einzelnen Schritte benötigten Zeiten addieren sich zu der gesamten Prozesswechselzeit, die bei jedem Prozesswechsel benötigt wird.

Während der Durchführung eines Prozesswechsels ist es nicht möglich, Daten auf dem FPGA-Koprozessor zu verarbeiten. Bedingt durch die Prozesswechsel verlängert sich somit die gesamte Ausführungszeit der einzelnen Anwendung und die Leistungsfähigkeit des Client-Server Betriebssystems sinkt. Um den Einfluß der Prozesswechselzeiten auf die Leistungsfähigkeit bewerten zu können wurden die Prozesswechelzeiten bei einer und bei drei gleichzeitig laufenden Anwendungen gemessen. Die Histogramme der Ausführungszeiten ist in der nachfolgenden Abbildung 5.15 dargestellt.

Wie in dem oberen Diagramm deutlich zu erkennen ist, beträgt die Prozesswechselzeit bei der Ausführung von nur einer Anwendung weniger als eine Millisekunde. In dieser Zeit wird das 'Multi-Slot Schedulingverfahrens ausgeführt und es wird überprüft, ob sich weitere Anwendungen am Server anmelden wollen. Aufgrund der einen Anwendung werden keine Prozesswechsel durchgeführt. Die 39 gemessenen Prozesswechselzeiten, die mehr als 500ms in Anspruch nehmen, ergeben sich durch die einmalige Initialisierung der Anwendung. Diese Initialisierung benötigt über eine Sekunde[5] und führt aufgrund des gegenseitigen Prozessausschluß der beiden Threads zu dieser fehlerhaften Messung.

[5] Das Laden des Konfigurationsbitstroms in den Host-Speicher nimmt mit rund einer Sekunde die meiste Zeit in Anspruch.

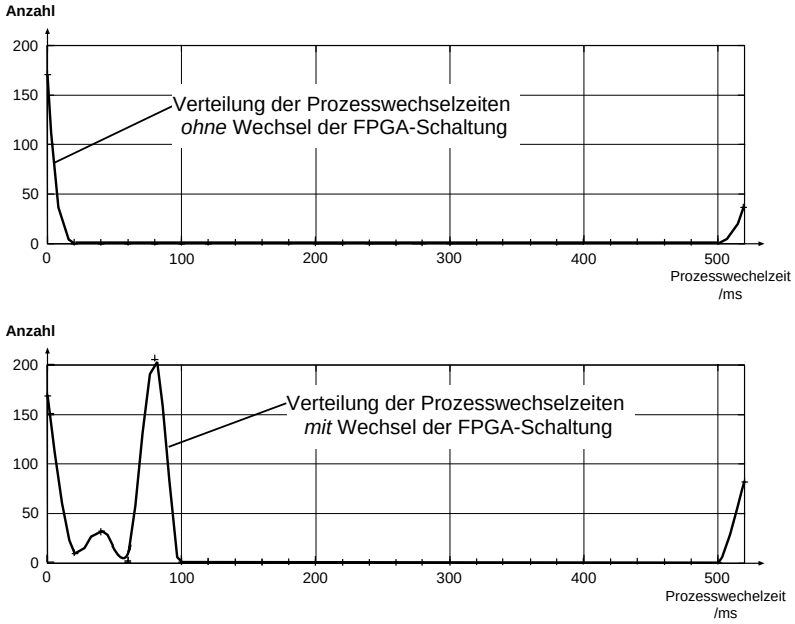


Abbildung 5.15: Histogramm der benötigten Prozesswechselzeiten. *Im oberen Histogramm ist die Zeitverteilung ohne einen Prozesswechsel und im unteren Histogramm mit durchgeführten Prozesswechseln dargestellt.*

Im Vergleich dazu ist in dem darunterliegenden Diagramm die Verteilung der gemessenen Prozesswechelszeiten dargestellt, die bei der gleichzeitigen Verarbeitung von drei Anwendungen auf dem Client-Server Betriebssystem entsteht. Auch hier sind Prozesswechselzeiten von weniger als einer Millisekunde zu sehen. Sie entstehen sobald kein Client am Server angeschlossen ist, aber auch durch das 'Multi-Slot' Schedulingverfahren wenn die Anwendung auf dem FPGA-Koprozessor nicht gewechselt wird. Auch die fehlerhaften Messungen durch die Initialisierung sind in dem Histogramm zu erkennen an den 84 Prozesswechselzeiten die über 500ms lang sind. Im Vergleich zu dem oberen Histogramm sind zwei wei-

tere Peaks der Prozesswechselzeiten bei 80ms und bei 40ms zu erkennen. Der Peak bei 80ms entsteht durch die Prozesswechsel, die die verdrängt FPGA-Schaltung zurücklesen und die zur Ausführung ausgewählte Anwendung wiederherstellen und konfigurieren. Diese Prozesswechsel sind aufgrund der gewählten Prioritäten am häufigsten aufgetreten. Ein weiterer kleinerer Peak bei 40ms entsteht durch Prozesswechsel, die beim Beendigen der Anwendung ausgeführt werden. Wird eine Anwendung beendet so ist es nicht mehr notwendig den Zustand des FPGA-Bausteins zu sichern. Durch den nicht ausgeführten Readback verringert sich die Prozesswechselzeit automatisch auf nur noch ≈40ms.

Aus dem unteren Histogramm wird deutlich, daß das realisierte Client-Server Betriebssystem bei der gleichzeitigen Ausführung von zwei oder mehr Anwendungen jeweils einen systembedingten Overhead von ≈80ms benötigt, um von der derzeit ausgeführten auf die nächste Anwendung umzuschalten. Bei der hier eingestellten Zeitscheibenlänge von 500ms ergibt sich somit ein Verlust von 16% der realen Aufführungszeit. Die Abbildung 5.16 zeigt diese verlängerte Ausführungszeit am Beispiel auf. Dargestellt ist eine Anwendung, die 2 Sekunden Rechenzeit benötigt. Im oberen Diagramm ist die optimale Ausführung ohne ein Prozesswechsel dargestellt. In der unteren Darstellung wird die Anwendung nach jeder Zeitscheibe unterbochen und kann erst nach der Prozesswechselzeit wieder ausgeführt werden. Durch diese zusätzlichen Prozesswechselzeiten ergibt sich eine theoretische Verlängerung der Ausführungszeit von den ursprünglichen 2 Sekunden auf 2,38 Sekunden. Dies entspricht einer Leistungsreduktion von 19%.

5.6.3.3 Zeitscheibenlänge

Die Zeit die für einen kompletten oder nur in Teilen durchgeführten Prozesswechsel notwendig ist, macht sich auch in der Länge der einzelnen Zeitscheiben bemerkbar. Messungen dieser Zeitscheibenlängen geben Aufschluß über die Ausführungszeit die den jeweiligen Anwendungen zur Verfügung steht und ist somit ein Maß für die gerechte Verteilung des FPGA-Koprozessors.

Gemessen wurden hier jeweils die Zeit, in der die Anwendung auf dem FPGA-Koprozessor ausgeführt wurde, ohne die Overheadzeiten durch

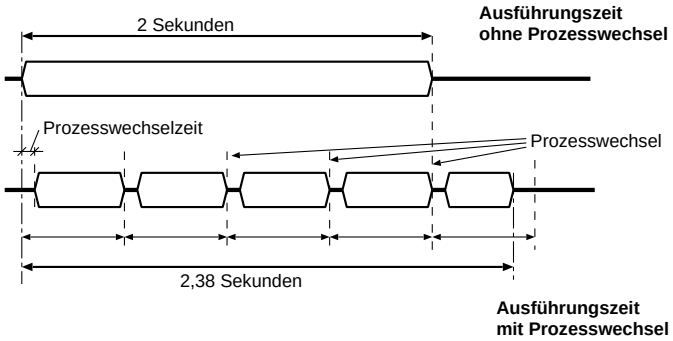


Abbildung 5.16: Auswirkung des systematischen Overheads auf eine Anwendung. *Im Beispiel beträgt der Overhead ≈80ms, die Zeitscheiben 500ms und die gesamte Anwendungsrechenzeit 2 Sekunden.*

die Prozesswechsel. Auch hier wurden unterschiedliche Messungen für nur eine Anwendung und für drei parallel laufende Anwendungen durchgeführt die in Abbildung 5.17 dargestellt sind.

Wie die Abbildung zeigt sind auch hier wieder die Auswirkungen der langen Initialisierung zu erkennen. Bedingt durch diese blockierende Funktion entstehen sehr kurze Zeitscheiben die im Bereich zwischen 100ms und 200ms auftreten. Desweiteren sind die, durch den 80ms langen Prozesswechsel eingeschränkten, realen Zeitscheiben bei der Verarbeitung mehrerer paralleler Anwendungen zu erkennen. Sie sind zwischen 420ms und 500ms lang. Der Peak bei 500ms kennzeichnet die durch die Einstellung des Zeitgebers eingestellte Zeitscheibenlänge. Diese wird sehr häufig gemessen wenn entweder nur eine Anwendung verarbeitet wird oder wenn der Server ohne eine Clientverbindung läuft. Die steil zulaufenden Histogrammpeaks besitzen ein geringe Standardabweichung. Das bedeutet, daß die zur Verfügung stehende Rechenzeit präzise und fair unter den konkurrierenden Anwendungen verteilt wird.

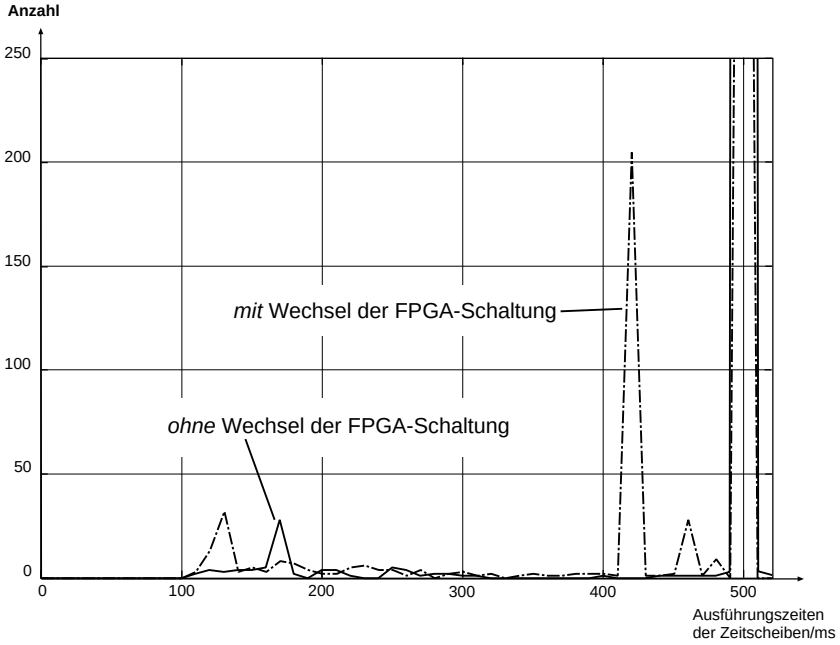


Abbildung 5.17: Histogramm der Zeitscheibenlängen. *Ein Histogramm zeigt die Verteilung der Zeitscheiben bei Ausführung von nur einer Anwendung und das andere bei drei parallel ausgeführten Anwendungen.*

5.6.3.4 Ausführungsgeschwindigkeit

Zur Verifikation der im vorherigen Abschnitt theoretisch bestimmten Verlängerungszeit wurde eine weitere Messung durchgeführt, welche die Verlängerung der Ausführungszeit von drei gleichzeitig ausgeführten Anwendungen ermittelt.

Die drei Anwendungen werden zum einen einzeln und hintereinander und zum anderen parallel ausgeführt. Gemessen wird dabei jeweils die Zeit, die zur Verarbeitung aller drei Anwendungen benötigt wird. Gestartet werden die drei Anwendungen jeweils mittels einer 'fork'-Systemfunktion, um einen objektiven Messwert zu erhalten. Nach der Durchführung von mehreren Messungen auf dem beschriebenen Computersystem wurden die folgenden Ausführungszeiten ermittelt:

Messung	Durchschnittliche Ausführungszeit
sequenzielle Ausführung	11,27 sek.
parallele Ausführung	12,60 sek.

Tabelle 5.2: Ausführungszeiten für eine sequenzielle und eine parallele Verarbeitung. Gemessen wurde die Ausführungszeit der Client-Anwendung.

Im Vergleich zu der zuvor ermittelten theoretischen Leistungsreduktion um 19% beträgt die mit dem System gemessene Leistungsreduktion weniger als 12%. Der Unterschied zu dem theoretischen Wert ergibt sich aus den Prozesswechseln, bei denen nur ein Teil der Einzelaufgaben durchgeführt werden müssen. So führen von den insgesamt 19 notwendigen Prozesswechseln nur 13 einen Readback, eine Rekonstruktion und eine Konfiguration durch. Werden jedoch mehrere Anwendungen mit langer Ausführungszeit parallel ausgeführt, so wird sich der gemessene Wert an den theoretischen Leistungswert annähern.

5.7 Ergebnis

Mit dem hier vorgestellten Client-Server Betriebssystem ist erstmals ein Betriebssystem für FPGA-Prozessoren verfügbar, das im Vergleich zu den bestehenden Betriebssystemen ein preemptives Multitasking durchführt. Der gewählte Ansatz beinhaltet dabei die vollständige Rekonstruktion des FPGA-Zustandes inklusive aller Register und RAM-Zellen.

Durch den Einsatz dieses Client-Server Betriebssystems wird die Verarbeitung von mehreren unabhängigen Anwendungen auf den FPGA-Koprozessor möglich. Darüberhinaus wird auch die Entwicklung und Programmierung der Anwendungen durch die Einführung einer einheitlichen Benutzerschnittstelle und die exklusive Nutzung des FPGA-Koprozessors durch nur eine Anwendung zu einer Zeit vereinfacht. Verbunden ist der Einsatz dieses Client-Server Betriebssystems mit einer minimalen Leistungsbeeinträchtigung von maximal 19%. Dies bedeutet, daß eine Anwendung, die nur über einen Beschleunigungsfaktor von 1.2 verfügt, durch den Einsatz des FPGA-Betriebssystems immer noch gleich schnell zu einer vergleichbaren Softwarelösung arbeiten würde. Diese Leistungswerte ergeben sich für eine Zeitscheibenlänge von 500ms, was der durchschnittlichen Ausführungslänge von Anwendungen entspricht, die auf den FPGA-Prozessoren ausgeführt werden.

Neben der erfüllten Anforderung nach einer leistungsstarken Implementierung ist auch die Portierbarkeit des gesamten Client-Server Betriebssystems erreicht worden. Das hier vorgestellte Client-Server Betriebssystems basiert auf einem allgemein verwendbaren FPGA-Koprozessor und kann durch den Austausch von einzelnen Softwaremodulen leicht auf andere FPGA-Koprozessor oder Host-Betriebssysteme portiert werden.

Obgleich das umgesetzte Client-Server Betriebssystem alle gestellten Aufgaben erfüllt, ist nicht die gesamte Funktionalität realisiert worden. Diese ist gegeben durch die Wahl des eingesetzen FPGA-Koprozessors der zwei der gestellten Anforderungen nicht erfüllt. Dies ist zum einen die nur unzureichende Kontrolle der Taktgeneratoren und zum anderen die realisierte RAM-Anbindung auf dem FPGA-Koprozessor. Neben diesen beiden Punkten ist auch eine weitere Verringerung der Prozess-

wechselzeiten durch eine optimierte Konfigurationsschnittstelle möglich. Im nächsten Abschnitt sind die Verbesserungspotentiale näher erläutert.

5.8 Verbesserungspotentiale

Obwohl das erzielte Ergebnis des Client-Server Betriebssystems eine parallele Verarbeitung von mehreren Anwendungen bei geringer Leistungseinschränkung ermöglicht, besteht für mehrere Punkte noch Verbesserungspotential. In diesem Abschnitt werden drei einzelne Punkte genannt, die bei einer optimalen Realisierung die Leistungsfähigkeit des gesamten Client-Server Betriebssystems weiter verbessern und die vollständige Funktionalität des FPGA-Koprozessors ermöglichen. Die nachfolgende Aufzählung beschreibt die Punkte im Detail:

Konfigurationsschnittstelle: Die auf dem verwendeten FPGA-Koprozessor microEnable II eingesetzte Schnittstelle für die Konfiguration und den Readback des FPGA-Bausteines verfügt über eine direkte Verbindung zu dem PCI-Interfacebaustein und kann aufgrund des Interfacebausteins nur mit maximal 40MHz betrieben werden. Somit beträgt die minimale Konfigurationszeit theoretisch 7,96ms. Aufgrund der geringen Speichertiefe des PCI-Interfacebausteins entstehen jedoch nur sehr kleine Bursts[6] auf dem PCI-Bus, so das die kleinste gemessene Konfigurationszeit nur 12ms beträgt. Zusätzliche Verzögerungen, z.B. durch die Auslagerung von Speicherseiten, verringern die durchschnittlichen Konfigurationszeiten im laufenden Betrieb auf nur ≈50ms.

Eine Optimierung dieser gesamten Anbindung an den zum Einsatz kommenden FPGA-Baustein verringert die Zeiten sowohl für die Konfiguration als auch für den Readback. Durch diese Optimierung wird die gesamte Prozesswechselzeit verringert, wodurch sich direkt die Leistungsfähigkeit des gesamten Client-Server Betriebssystems verbessert.

[6] Ein Burst ist ein zusammenhängendes Datenpaket, in dem aufeinanderfolgende Daten effektiv übertragen werden.

Taktkontrolle: Die auf der microEnable II realisierten Taktgeneratoren verfügen nicht über die Möglichkeit sie vollständig abzuschalten. Somit ist es nur über eine entsprechende Erweiterung der Schaltungsfunktionalität möglich, den Zustand für die Zeit des Readbacks 'einzufrieren'. Auch ein nachträgliches erweitern dieser Funktionalität ist nur für zwei der drei Taktgeneratoren machbar, da ein Takt für den Readback und die Konfiguration des FPGA-Bausteins permanent anliegen muß.

Ohne eine nachträgliche Veränderung der Taktgeneratoren oder die Erweiterung der Schaltungsfunktionalität ist die Gruppe der Anwendungen begrenzt auf reine Datenflußanwendungen bzw. auf Anwendungen die ihren gesamten Zustand innerhalb des FPGA-Bausteins nur über spezielle Steuersignale verändern. Nur bei diesen Anwendungen ist sichergestellt, daß nach dem Anhalten der FPGA-Schaltung sich der interne Zustand nicht weiter verändert.

RAM-Anbindung: Die größte Einschränkung der Funktionalität ist gegeben durch die direkte Anbindung des lokalen RAMs auf dem FPGA-Kopro-zessor. Dieses statische RAM ist direkt verbunden mit dem FPGA-Baustein und kann nur über diesen beschrieben oder ausgelesen werden. Da das Client-Server Betriebssystem nicht davon ausgehen kann, daß jede Anwendung einen solchen direkten Zugriff über die konfigurierte FPGA-Schaltung ermöglicht, muß bei jedem Prozesswechsel eine spezielle FPGA-Schaltung für den direkten Zugriff auf das RAM konfiguriert werden.

Die Sicherung und Rekonstruktion der Daten eines 2MByte großen RAMs benötigt theoretisch ≈33ms. Hierzu muß noch die Konfigurationszeit hinzuaddiert werden, die für den direkten Zugriff auf das RAM notwendig ist. Somit benötigt die Sicherung und die Rekonstruktion des RAM-Inhalts bis zu 83ms und verdoppelt die Prozesswechselzeit auf insgesamt 165ms.

Zur Realisierung dieser Verbesserungen wurde im Rahmen der Arbeit ein eigener FPGA-Koprozessor aufgebaut, der im speziellen das preemptive Multitasking unterstützt. Der Aufbau und das Konzept dieses FPGA-Koprozessors wird in dem folgenden Kapitel erläutert.

5.9 Zusammenfassung

In diesem Kapitel wurden die Realisierung des ausgewählten Client-Server Betriebssystems in Detail beschrieben. Das umgesetzte FPGA-Betriebssystem basiert dabei auf drei grundlegenden Konzepten. Zwei dieser Konzepte beschreiben die Kommunikation zwischen Client und Server und die Aktivitäten bzw. die Zustände der einzelnen Anwendungen im Server. Das dritte Konzept umfasst das Schedulingverfahren, das zur Auswahl der nächsten auszuführunden Anwendung eingesetzt wird.

Zur Auswahl diese Schedulingverfahrens wurden mehrere Bewertungskriterien eingefügt und die bestehenden Schedulingverfahren bezüglich ihrer Eignung für das Client-Server Betriebssystem bewertet. Aufgrund der Randbedingungen und Anforderungen hat sich das Round-Robin Verfahren als das geeignetste herausgestellt, das durch entsprechende Modifikationen zu einem 'Multi-Slot' Schedulingverfahren erweitert wurde. Dieses neue 'Multi-Slot' Verfahren erlaubt eine prioritätsbasierte Ausführung der Anwendungen und berücksichtigt die relativ hohen Prozesswechselzeiten in einem besonderen Maße.

Zusammen mit der Realisierung der Kommunikation und des gewählten Schedulingverfahren wird die dem Client-Server Betriebssystem zugrundeliegende Softwarearchitektur beschrieben, die eine hohe Leistungsfähigkeit und eine einfache Portierbarkeit des gesamten Systems ermöglicht. Die Funktionalität des Servers aber auch des Clients wird anhand des internen Funktionsablauf erläutert. Neben den zum Einsatz kommenden Softwaremodulen wird auch der eingesetzte FPGA-Koprozessor kurz beschrieben.

Zur Bewertung der Leistungsfähigkeit des gesamten Client-Server Betriebssystems werden zuerst die für das beschrieben System erreichbaren maximalen Datentransferraten für DMA-Übertragungen gemessen. Diese bilden die Grundlage zur späteren Bewertung des Betriebssystems. Danach werden einzelne Kenngrößen des Client-Server Betriebssystems vermessen, um die Leistungsfähigkeit des Systems zu beurteilen. Dazu gehören der Kommunikationsoverhead für den Datenaustausch zwischen Client und Server, die Verteilung der Prozesswechselzeiten bei

der Verarbeitung von einer bzw. mehrerer Anwendungen, die Verteilung der Zeitscheibenlängen als ein Maß für die Verteilung der Rechenzeit und letztendlich die Leistungseinschränkung durch das Betriebssystem aus Sicht der Anwendung. Wie diese Messungen zeigen, beträgt die durchschnittliche Prozesswechselzeit weniger als eine Millisekunde bei der Verarbeitung von nur einer Anwendung und ≈80ms bei mehreren parallel arbeitenden Anwendungen. Diese Prozesswechselzeiten bestimmen maßgeblich die Leistungsbeeinträchtigung der Anwendungen die gemessen bei weniger als 12% liegen. Der theoretische Leistungsverlust beträgt maximal 19% und wird nur bei sehr rechenintensiven und lang auszuführenden Anwendungen erreicht. Unter Berücksichtigung der theoretischen Leistungsverluste von maximal 19% werden alle Anwendungen mit einem Beschleunigungsfaktor von 1.2 und mehr durch den Einsatz des Client-Server Betriebssystems nur so beeinflußt das der Beschleunigungsfaktor größer als 1 bleibt.

Abschließend wird durch eine Analyse des gesamten Systems herausgearbeitet, an welchen Punkten noch weiteres Verbesserungspotential liegt. Es stellt sich heraus, daß die Einschränkungen durch den verwendeten FPGA-Koprozessor entstehen. Basierend auf diesen Einschränkungen und der Analyse wird in dem nachfolgenden Kapitel die Konzeptionierung eines speziellen FPGA-Koprozessors beschrieben der das preemptive Multitasking im besonderen Maß unterstützt.

Kapitel 6

Multitasking FPGA-Koprozessor

Die im vorherigen Kapitel beschriebene Realisierung des Client-Server Betriebssystems hat gezeigt, daß die Leistungseinschränkungen durch den Einsatz des FPGA-Betriebssystem sich maximal in einer 19% längeren Ausführungszeit für jede Anwendung zeigt. Diese Verlängerung der Ausführungszeit ist dabei direkt proportional zu der Zeit, die das Client-Server Betriebssystem benötigt, um die auszuführenden Anwendungen auf dem FPGA-Koprozessor zu wechseln.

Aus dieser Erkenntnis und aus der durchgeführten Analyse des gesamten Systems, einschließlich des verwendeten FPGA-Koprozessor microEnable II, ergeben sich drei Hauptpunkte die ein erhebliches Verbesserungspotential aufweisen. Die beiden Erkenntnisse, die das größten Potential aufweisen, sind zum einen die Umsetzung der Anbindung an die Konfigurationsschnittstelle auf dem FPGA-Koprozessor und zum anderen die aus Leistungsgründen nicht realisierte Sicherung und Rekonstruktion des RAM-Inhalts bei einem Prozesswechsel.

Dieses Kapitel beschreibt nun den Aufbau eines FPGA-Koprozessors der im speziellen für den preemptiven Multitaskingeinsatz konzipiert wurde und durch die Optimierung der ermittlten Problemstellen die

Prozesswechselzeiten auf ein Minimum verringert. Zusätzlich zu der Umsetzung einer Hardwareunterstützung für den Multitaskingbetrieb wird auch ein Softwarekonzept entwickelt, das die Entwicklung und Realisierung von neuen Anwendungen im besonderen unterstützt und vereinfacht. Die Verbesserungen zu dem bestehenden FPGA-Koprozessoren werden im Detail beschrieben und durch entsprechende Messungen mit dem FPGA-Koprozessor microEnable II verglichen. Darüberhinaus wird auch das gesamte Softwarekonzept detailliert beschrieben, das ebenfalls die Optimierung der Anwendungsausführung als Ziel verfolgt.

6.1 Konzept

Das umfassende Gesamtkonzept des neuen FPGA-Koprozessors besteht sowohl in speziellen Anpassungen zur Minimierung der Prozesswechselzeiten als auch in der Einführung eines neuen Ausführungsmodells für den FPGA-Koprozessor.

Das primäre Ziel ist es, den FPGA-Koprozessor optimal durch die parallel laufenden Anwendungen zu nutzen und zugleich die entstehenden Overheadzeiten zu minimieren. Das sekundäre Ziel ist das neue Ausführungsmodell, das den Multitaskingbetrieb unterstützt und zusammen mit einer kompletten Entwicklungsumgebung die Realisierung von neuen Anwendungen für den FPGA-Koprozessor erleichtert.

Die entscheidenden Veränderungen gegenüber bestehenden FPGA-Koprozessoren ergeben sich aus der durchgeführten Analyse des letzten Kapitels, den gestellten Anforderungen aber auch aus den Erfahrungen bei der Anwendungsentwicklung. Getrennt in die Konzepte für den FPGA-Koprozessor und die Softwareumgebung werden diese nachfolgend näher erläutert.

6.1.1 Multitasking Hardware

Die Konzepte zum Umsetzen eines effektiven Multitaskingbetriebs ergeben sich aus den zuvor gemittelten Erkenntnissen. Oberstes Ziel für die Optimierungen ist die Verringerung der Prozesswechelzeit, die den systembedingten Overhead bestimmt.

Unter Verfolgung dieses Ziels konzentrieren sich die Optimierungen auf eine verbesserte Anbindung des PCI-Busses an die Konfigurationsschnittstelle und die effektive Umsetzung von Sicherung und Rekonstruktion der lokalen RAMs auf dem FPGA-Koprozessor während eines Prozesswechels. Darüberhinaus wird die Umsetzung der Taktgenerierung angesprochen, die für den vollständigen Multitaskingbetrieb vorausgesetzt wird. Die Konzepte für die Umsetzung dieses Ziels auf dem neuen FPGA-Koprozessor, der das preemptive Multitasking im Besonderen unterstützt, wird im folgenden kurz erläutert:

Takterzeugung: Das Konzept der Takterzeugung umfasst im wesentlichen die in Abschnitt 3.4.2.2 genannten Anforderungen, die an den FPGA-Koprozessor gestellt werden. Zum Anhalten des Taktes werden die abschaltbaren Ausgangstreiber deaktiviert und über einen Pull-Down Widerstand[1] wird ein stabiles Signal erzeugt. Die sichere Abschaltung erfolgt mit Hilfe der in Abschnitt 3.4.2.2 beschriebenen Logik.

Wie die Erkenntnisse gezeigt haben ist es notwendig, die Konfigurationsschnittstelle mit einem eigenen permanenten Takt zu versorgen, da diese Schnittstelle auch nach dem Anhalten der FPGA-Schaltungen für den Readback und die Konfiguration genutzt wird. Dieser sollte nach Möglichkeit mit der maximal zulässigen Frequenz von 50MHz[2] getaktet werden um eine schnelle Konfiguration zu ermöglichen. Desweiteren wird auch auf einen Gebrauch von DLL- und PLL-Schaltungen verzichtet.

Konfigurationsschnittstelle: Zum Erreichen einer minimalen Prozesswechselzeit ist es notwendig alle an der Konfiguration und dem Readback beteiligten Bausteine auf dem FPGA-Koprozessor mit einzubeziehen. Aus diesem Grunde beschränkt sich das Konzept nicht nur auf die Auswahl des FPGA-Bausteins mit der schnellsten Konfigurationsschnittstelle, sondern umfasst darüberhinaus

[1] Ein Pull-Down Widerstand zieht das Signal auf einen LOW Pegel wenn dieser nicht von dem Taktgenerator getrieben wird.

[2] Bei 50MHz kann der Virtex Baustein die Konfigurationsdaten mti jedem Takt übernehmen.

auch eine optimale Anbindung des eingesetzten PCI-Busses an den Baustein und die Umsetzung auf dem Host-Prozessor.

Die Notwendigkeit eines solchen übergreifenden Konzeptes wird deutlich bei der Betrachtung der Konfigurationszeiten die mit dem FPGA-Koprozessor microEnable II gemessen wurden. Dort beträgt die minimale Konfigurationszeit 12ms und liegt damit rund 50% höher als die theoretische Zeit von nur 7,69ms. Zurückzuführen ist das unter anderem auf die kleinen Bursts auf dem PCI-Bus die durch eine nicht optimale Anbindung an den PCI-Bus entstehen.

RAM-Switch: Das dritte Konzept zum Verbessern der Prozesswechselzeiten beinhaltet die Integration des lokalen RAM-Inhalts in den Prozesswechsel. Eine solche Sicherung des RAM-Inhalts wurde bei dem realisierten Client-Server Betriebssystem nicht umgesetzt, da eine Sicherung auf dem FPGA-Koprozessor microEnable II nur durch einen Zugriff über den FPGA-Baustein durchgeführt werden kann. Dies bedeutet aber, daß sich die Prozesswechselzeit auf 165ms verdoppeln würde.

Das neue Konzept orientiert sich an den gemachten Erkenntnissen und umfasst sowohl einen direkten Zugriff des PCI-Busses auf die RAMs als auch einen RAM-Switch. Der direkte Zugriff des PCI-Busses auf den RAM-Inhalt reduziert z.B. die oben genannte Prozesswechselzeit um 30% auf nur noch 115ms, denn die Konfiguration mit einer FPGA-Schaltung, die den Zugriff ermöglicht, ist für die Sicherung des RAM-Inhalts nicht mehr notwendig. Zusätzlich ermöglicht dieser direkte Zugriff die Durchführung des im nächsten Abschnitt beschriebenen Ausführungsmodell.

Der RAM-Switch verbessert die Situation noch einmal und reduziert den Prozesswechsel weiter auf die Zeiten die für die Konfiguration, den Readback und die Zustandsrekonstruktion notwendig sind. Erreicht wird dies, indem die einzelen RAM-Bänke bei einem Prozesswechsel komplett abgetrennt werden und bei einer erneuten Ausführung wieder aktiviert werden. Durch dieses Vorgehen wird die Sicherung des RAM-Inhalts überflüssig, da eine abgetrennte RAM-Bank seinen Inhalt nicht verändert.

6.1.2 Softwarekonzept

Die nachfolgend beschriebenen Softwarekonzepte werden sowohl für eine Reduzierung der Prozesswechselzeiten als auch für eine Vereinfachung der Anwendungsentwicklung eingeführt. Sie ergeben sich aus den Erkenntnissen und Erfahrungen bei der Anwendungsentwicklung und sind bestimmt durch vorgegebene Randbedingungen.

Die Konzepte umfassen den Aufbau einer Softwareentwicklungsumgebung, die u.a. durch eine Nutzungseinschränkung des FPGA-Bausteins die Entwicklung vereinfacht, aber auch die Komplexität des FPGA-Betriebssystems minimiert. Erreicht wird das durch die exklusive Nutzung des FPGA-Bausteins durch nur eine Anwendung. Darüberhinaus ist die partielle Nutzung des Bausteins nicht vorgesehen. Eine zusätzliche Verringerung der Ausführungszeiten wird durch ein spezielles Ausführungskonzept erreicht, das durch einen speicherbasierten Datenaustausch eine zum Teil parallele Ausführung ermöglicht. Die einzelnen Softwarekonzepte werden nachfolgend kurz erläutert:

Partielle Nutzung des FPGA-Bausteins: Das Konzept eine partielle Nutzung des FPGA-Bausteins durch verschiedenen Anwendungen, wie z.B. bei dem Betriebssystem von G. Brebner (siehe Abschnitt 3.2.2 auf Seite 42), wird nicht unterstützt.

Wie dieses Betriebssystem gezeigt hat, ist für die partielle Nutzung eines FPGA-Bausteins ein zumeist komplexer Partitionierungalgorithmus notwendig, um die einzelnen FPGA-Schaltungen auf dem FPGA-Baustein ausführen zu können. Darüberhinaus hat die Anwendungsanalyse in Kapitel 4 gezeigt, daß nahezu alle Anwendungen externe RAM-Elemente verwenden und durch die gemeinsame Nutzung dieser RAMs die Leistungsfähigkeit jeder einzelnen Anwendung reduziert wird. Ein weiterer praktischer Grund für die exklusive Nutzung durch eine Anwendung ist die mangelhafte Unterstützung durch Place&Route-Werkzeuge die eine partielle Plazierung nicht oder nur sehr beschränkt zulassen.

Exklusive Anwendungsnutzung: Resultierend aus der nicht unterstützten partiellen Nutzung ergibt sich das Konzept der exklusiven Nutzung des FPGA-Bausteins durch jeweils nur eine Anwendung.

Dieses Konzept ermöglicht jeder Anwendung die gesamte zur Verfügung stehende Datenbandbreite zu externen Elementen, wie z.B. lokales RAM, zu nutzen. Desweiteren stehen der jeweiligen Anwendung auch die gesamten FPGA-Ressourcen zur Verfügung, so daß es z.B. aufgrund eines hohen Auslastungsgrads nicht zu einer Verringerung der Taktrate und damit zu einer Verringerung des Beschleunigungsfaktors kommt. Ein weiterer Vorteil, der durch die exklusive Nutzung entsteht, ist eine Verbeserung der Turnaround Zeiten bei der Entwicklung von neuen FPGA-Schaltungen. Aus Sicht der jeweiligen Anwendung bzw. der dazugehörigen FPGA-Schaltung sind die gesamten Ressourcen des FPGA-Koprozessors verfügbar. Dies ermöglicht es eine Simulationsumgebung zu generieren, die für alle Schaltungen eingesetzt werden kann.

Speicherbasiertes Ausführungsmodell: Durch den Einsatz des speicherbasierten Ausführungsmodells wird es möglich, die Ausführungszeiten der einzelnen Anwendungen weiter zu verbessern und dadurch den Einfluß der FPGA-Betriebssystems weiter zu vermindern.

Bei den derzeitigen Anwendungen werden die Daten erst übertragen, sobald der FPGA-Baustein mit der FPGA-Schaltung konfiguriert ist. Diese Vorgehensweise begründet sich in dem nicht vorhandenen direkten Zugriff auf die RAM-Bänke. Besteht jedoch die Möglichkeit des direkten Zugriffs während der Ausführung einer anderen Anwendung, so kann der Datenaustausch vor der Konfiguration und damit vor der eigentlichen Ausführung erfolgen. Das speicherbasierte Ausführungsmodell basiert auf diesem direkten RAM-Zugriff und überträgt die Daten *immer* über die RAM-Bänke. Dieser Datenaustausch erfolgt während der Ausführung der anderen Anwendung und verringert dadurch die gesamte Ausführungszeit der Anwendung. Für das Betriebssystem bedeutet das eine Verringerung der Turnaroundzeiten und eine gleichzeitige Erhöhung der Auslastung. Zusätzlich ist auch eine Beschleunigung der Datenflußanwendungen aufgrund der höheren Speicherbandbreite zu erwarten.

6.2 FPGA-Koprozessor Implementierung

Unter Berücksichtigung der bei der Anwendungsanalyse ermittelten Daten und der oben genannten Konzepte ist ein neuer FPGA-Koprozessor entstanden, der ein preemptives Multitasking im besonderen Maße unterstützt.

Die einzelnen Funktionsblöcke des neuen FPGA-Koprozessors sind in der nachfolgenden Abbildung 6.1 dargestellt.

Der neue FPGA-Koprozessor verwendet, wie die Mehrzahl der existierenden FPGA-Koprozessoren, den PCI-Bus als Datenverbindung zum Host-Prozessor. Der PCI-Bus wurde gewählt, da er über eine sehr hohe erreichbare Datenrate verfügt und weit verbreitet ist. Angebunden ist die Karte über eine PCI-Interfacebaustein der die gesamte Kommunikation über den PCI-Bus kontrolliert. Als Prozessorelement kommt ein FPGA-Baustein der Virtex Serie zum Einsatz, da diese Bausteine über eine sehr schnelle Konfigurationsschnittstelle verfügen (siehe Tabelle 3.2) und die Bitstromarchitektur dieser Bausteine gut dokumentiert ist. Der Virtex FPGA-Baustein bildet das zentrale Prozessorelement, auf dem die FPGA-Schaltungen ausgeführt werden. Verbunden ist der FPGA-Baustein mit dem PCI-Interfacebaustein, dem externen Stecker und mit dem RAM-Switch.

Die Verbindung zu dem PCI-Interfacebaustein wird für den direkten Zugriff auf anwendungsspezifische Register genutzt, welche die jeweilige FPGA-Schaltung kontrollieren und steuern. Die ebenfalls direkte Verbindung zu dem externen Stecker kann genutzt werden um z.B. Link-Module für einen Datentransfer anzubinden. Bei einer solchen Verbindung ist allerdings darauf zu achten, daß das angeschlossene Modul durch ein HIGH-Signal inaktiv geschaltet wird um z.B. ein Auslesen während der Prozesswechsel zu vermeiden. Die dritte Verbindung besteht zwischen dem FPGA-Baustein und dem RAM-Switch. Wie die Anwendungsanalyse gezeigt hat ist es vor allem für Bildverarbeitungsanwendungen von Vorteil wenn diese auf zwei voneinander unabhängige RAM-Anbindungen zurückgreifen können. Aus diesem Grund wurden für die FPGA-Schaltungen zwei unabhängige Schnittstellen zu dem RAM-Switch realisiert.

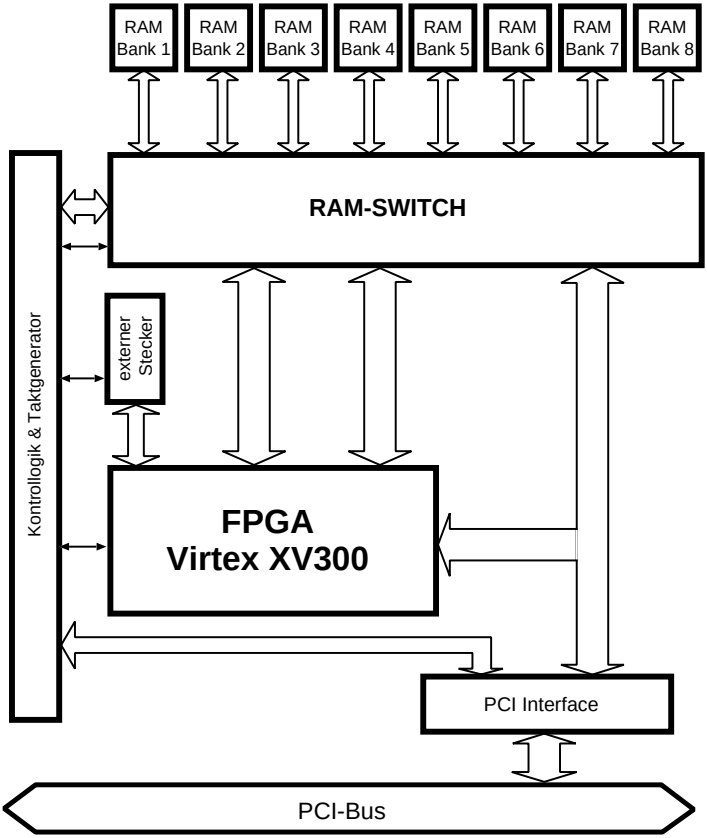


Abbildung 6.1: Blockschaltbild des neuen FPGA-Prozessors der ein preemptives Multitasking unterstützt.

Der RAM-Switch selbst umfasst auf der einen Seite die beiden Verbindungen zu dem FPGA-Baustein und die direkte Verbindung zu dem PCI-Interfacebaustein. Auf der anderen Seite sind insgesamt acht unabhängige RAM-Bausteine angeschlossen. Durch die Funktionalität des RAM-Switchs, die im folgenden Abschnitt näher erläutert ist, wird eine Sicherung des RAM-Inhalts über den PCI-Bus überflüssig und ein Ausführen der Anwendungen nach dem Konzept des speicherbasierten Datentransfer wird ermöglicht.

Die vierte Komponente auf dem neuen FPGA-Koprozessor übernimmt die gesamte Steuerung der einzelnen Komponenten, die Erzeugung der Taktsignale und realisiert eine effektive Anbindung der Konfigurationsschnittstelle an den PCI-Bus. Gesteuert wird neben der Takterzeugung vor allem der Zustand des RAM-Switchs. Die Takterzeugung und die Anbindung der Konfigurationsschnittstelle wird ebenfalls detailliert in den folgenden Abschnitten erläutert.

6.3 Multitaskingunterstützung

Wie schon erwähnt ist der neu konzipierte FPGA-Koprozessor vor allem auf den Betrieb mit einem preemptiven Multitasking Betriebssystem abgestimmt und minimiert die Zeit, die für einen Prozesswechel benötigt wird. Dies wird erreicht durch die zuvor beschriebenen Konzepte und deren spezielle Umsetzung auf dem FPGA-Koprozessor. Neben mehreren kleineren Verbesserungen, wie z.B. den Einsatz eines schnellen PCI-Interfacebausteins, wird die Multitaskingunterstützung mit der schnellen Prozesswechelzeit vor allem durch die Realisierung der folgenden drei Bereiche erreicht: Takterzeugung, Konfigurations- und Readbackschnittstelle und die Realisierung des RAM-Switchs. Im nachfolgenden werden die einzelnen Umsetzungen im Detail erläutert und in ihrer Funktionsweise erklärt.

6.3.1 Takterzeugung

Der Aufbau der Takterzeugung orientiert sich in erster Linie an den gestellten Anforderungen, die für den Multitaskingbetrieb erfüllt sein sollten. Diese Anforderungen sind:

- Die einzelnen Taktsignale müssen synchron und in einem ganzzahligen Verhältnis zueinander arbeiten.
- Das Taktsignal wird mit dem Takt angehalten der mit der niedrigsten Frequenz arbeitet. Dies ist der Basistakt.
- Die Taktfrequenz muß über einen großen Bereich einstellbar sein.
- Das Taktsignal für die Konfigurationsschnittstelle muß permanent verfügbar sein.

Aufgrund der letzten Anforderung an das Taktsignal, das für den Readback und die Konfiguration des Bausteins notwendig ist, wird sowohl das Taktsignal für die Konfigurationsschnittstelle als auch das Taktsignal für den lokalen Bus des PCI-Interfacebausteins mit einer permanenten und nicht veränderbaren Frequenz realisiert. Die Frequenz beträgt 40MHz für die Konfigurationsschnittstelle und 33MHz für den lokalen Bus.

Die anderen Taktfrequenzen, die zum Betreiben der FPGA-Schaltungen verwendet werden, müssen über einen weiten Bereich flexibel einstellbar sein. Diese hohe Flexibilität ist notwendig, da jede einzelne Schaltung aufgrund ihres spezifischen internen Timingverhaltens eine maximale Taktfrequenz besitzt. Aus diesem Grund werden für die Erzeugung dieser Taktsignale einstellbare Taktgeneratoren eingesetzt, die durch eine entsprechende Programmierung die Ausgangsfrequenz einstellen können. Der eingesetzte Baustein ICS525-01 [Cyp] von Cypress verfügt über einen sehr großen Frequenzbereich von 1MHz bis 160Mhz und kann über ein 'Power-Down' Pin seine Ausgangsfrequenz abschalten, um die FPGA-Schaltung anzuhalten. Die Reaktionszeit beträgt beim Abschalten 50ns und beim Wiedereinschalten bis zu 10ms. Aufgrund dieser langen Reaktionszeiten dieses 'Power-Down' Eingangs wäre ein sicheres Abschalten nur bis zu einer maximalen Frequenz von 16MHz[3] möglich. Bedingt durch diese Einschränkung wird auf dem neuen FPGA-Koprozessor eine Taktabschaltung mit sogenannten Quick-Switchs durchgeführt. Diese Quick-Switch Bausteine ermöglichen ein

[3]Die minimale Zeit zum Abschalten errechnet sich aus 50ns Reaktionszeit des Taktgenerators zuzüglich 10ns Verzögerungszeit durch die Taktsteuerungslogik.

Abschalten des Taktes in nur 4ns bei einer Taktdurchlaufverzögerung von nur 250ps. Die gesamte Schaltung mit den drei realisierten Taktgeneratoren ist in der Abbildung 6.2 dargestellt. Die maximale Takt-

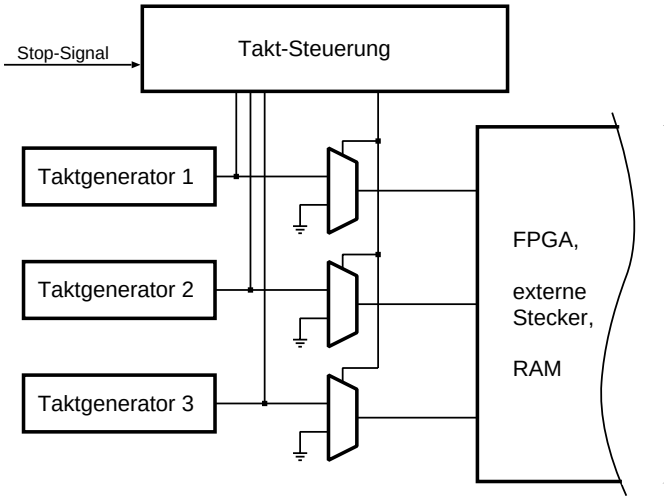


Abbildung 6.2: Schaltung zum Abschalten des Taktsignals mittels Quick-Switches.

frequenz zum sicheren Abschalten ergibt sich bei dieser Schaltung aus den Verzögerungen des Quick-Switches und der Taktsteuerungslogik und beträgt ≈70MHz. Diese maximale Taktfrequenz ist ausreichend, da die zu erwartende maximale Taktfrequenz der FPGA-Schaltung zwischen 40MHz und 60MHz liegt.

Eingeleitet wird die Taktabschaltung durch ein Zugriff des Betriebssystems auf ein Kontrollregister. Das Signal dieses Registers und die Signalisierung der FPGA-Schaltung bestimmen das Stop-Signal, denn der Takt darf nur abgeschaltet werden wenn kein externer Zugriff der FPGA-Schaltung durchgeführt wird. Sobald das Stop-Signal das Anhalten der Takte anzeigt und die Taktsignale in einem Zustand sind, die ein sicheres Abschalten ermöglichen, werden alle Taktsignale an dem Quick-Switch

auf ein LOW-Signal umgeschaltet und das Taktsignal dadurch angehalten. Das erneute Anschalten der Taktsignale beginnt mit der Freigabe des Stop-Signal durch das Betriebssystem. Dies bewirkt, daß abhängig von dem Zustand der Taktsignale die Quick-Switches erneut umgeschaltet werden und somit das Taktsignal wieder am FPGA-Baustein anliegt. Dieses kontrollierte An- bzw. Abschalten ist notwendig, um das System definiert abzuschalten um danach wieder mit den gleichen Takt und den gleichen Taktverhältnissen zu starten.

Die Anforderung nach der Synchronität der einzelnen Taktsignale wird durch die gemeinsame Taktung der drei Taktgeneratoren mit dem gleichen Referenztakt gewährleistet. Für die Einstellung der entsprechenden ganzzahligen Taktverhältnisse, die für das sichere Anhalten der FPGA-Schaltung gefordert werden, ist das FPGA-Betriebssystem verantwortlich, indem die einzelnen Taktgeneratoren entsprechend eingestellt werden.

Somit erfüllt das realisierte Taktsystem die Anforderungen, die für den preemptiven Multitaskingbetrieb gefordert werden und ermöglicht eine einfache und sichere Abschaltung der Taktfrequenzen bis zu einer Frequenz von 70MHz. Zudem wird durch die Umsetzung auf dem FPGA-Koprozessor die Anwendungsentwicklung vereinfacht, da keine zusätzliche Logik zum 'Einfrieren' der FPGA-Schaltung integriert werden muß bzw. keine Einschränkungen mehr bezüglich der verwendbaren Elemente bestehen.

6.3.2 Konfigurationsschnittstelle

Die Anbindung des Host-Prozessors an die Konfigurationsschnittstelle des FPGA-Bausteins ist bestimmend für die gesamte Prozesswechelszeit des FPGA-Betriebssystems. Die Prozesswechselzeit selbst besteht neben der Sicherung und Rekonstruktion des RAM-Inhalts im wesentlichen aus der benötigten Zeit für den Readback, die Rekonstruktion und die Konfiguration des FPGA-Bausteins. Eine minimale Prozesswechselzeit wird somit erreicht, indem die notwendigen Zeiten für die Konfiguration und den Readback durch eine optimale Anbindung minimiert werden.

Aus den Erkenntnissen mit dem Client-Server Betriebssystem erge-

ben sich vier Punkte, die zu einer optimalen Realisierung der gesamten Anbindung des FPGA-Betriebssystems an den verwendeten FPGA-Baustein auf dem FPGA-Koprozessor umgesetzt werden.

FPGA Bausteinwahl: Grundvoraussetzung für eine schnelle Konfiguration und einen schnellen Readback ist der Einsatz eines FPGA-Bausteins, der über eine schnelle parallele Konfigurationsschnittstelle verfügt. Ausgewählt wurde daher die Virtex Baustein Familie von Xilinx. Diese Bausteine besitzen eine 8Bit breite Schnittstelle die mit 50MHz betrieben werden kann und gehören daher zu dem FPGA-Bausteinen mit der derzeit schnellsten Konfigurationsschittstelle (siehe auch Tabelle 3.2 auf Seite 78).

Schnittstellenwahl: Zu einer optimalen Anbindung gehört auch die effektive Ausnutzung aller an der Anbindung beteiligten Komponenten. Dazu gehört im Falle der Konfiguration und des Readbacks die Konfigurationsschnittstelle des FPGA-Bausteins, der PCI-Interfacebaustein und die Datenübertragung über den PCI-Bus.

Die minimale Konfigurationszeit wird bestimmt durch die Konfigurationsschnittstelle des FPGA-Bausteins. Bei dem eingesetzten Virtex XV300, der über 214kByte Konfigurationsdaten verfügt, beträgt die theoretische Konfigurationszeit 4,4ms. Aufgrund der Taktrate des PCI-Busses von nur 33MHz würde bei einer byteorientierten Übertragung die minimale Konfigurationszeit auf 6,6ms steigen. Zusätzlich zu der Verlängerung der Konfigurationszeit würden bei dieser Lösung auch 642kByte nicht genutzte Daten zusätzlich übertragen, die die Gesamtlast auf dem PCI-Bus anheben und andere PCI-Übertragungen beeinflussen. Daher werden zur Vermeidung dieser längeren Konfigurationszeit jeweils vier Konfigurationsdatenworte in einem 32Bit PCI Zugriff übertragen. Dieses 32Bit Datenword wird nachfolgend entweder durch den PCI-Interfacebaustein oder durch eine andere zwischengeschaltete Logik auf vier 8Bit Datenworte umgesetzt und an die Konfigurationsschnittstelle angelegt.

Eine solche Übertragung der Daten, die ebenfalls für den Readback eingesetzt werden, ermöglichen die optimale Ausnutzung der

verfügbaren Komponenten und tragen somit zu der minimierten Prozesswechselzeit bei.

Übertragungsmode: Wie die Messungen an dem FPGA-Koprozessor microEnable II gezeigt haben, wird selbst bei Anwendung der zuvor beschriebenen Übertragung der Konfigurationsdaten die minimale Konfigurationszeit nicht erreicht. Die auf der microEnable II gemessene Konfigurationszeit von 12ms liegt dabei um rund 50% höher als die theoretische errechnete Zeit von 7,96ms[4]. Diese rund 50% längere Konfigurationszeit entsteht zum einen durch den Overhead der DMA-Übertragung und zum anderen durch die kleinen Datenbursts bei der Übertragung auf dem PCI-Bus.

Zur effektiven Übertragung des Konfigurationsdatenstroms wird die DMA-Übertragung gewählt. Bei jeder DMA-Übertragung werden die Daten zuerst im Speicher vorbereitet und danach wird die DMA-Steuereinheit eingestellt. Die Zeit für diese Vorbereitungen wird als Overhead bezeichnet und beträgt ≈0.3ms. Zu erkennen ist dies auch in den beiden Abbildungen 5.11 und 5.12.

Der größere Zeitverlust bei der Konfiguration entsteht jedoch durch kurze Datenbursts auf dem PCI-Bus, die aufgrund einer unterschiedlichen Dateneingangs- und Datenausgangsrate entstehen. In Fall des FPGA-Koprozessors microEnable II werden 32Bit Datenworte mit einer Taktfrequenz von 33MHz von PCI-Bus in den PCI-Interfacebaustein geschrieben und 8Bit Datenworte mit 40MHz an den FPGA-Baustein weitergegeben. Für die Zwischenspeicherung im PCI-Interfacebaustein steht ein 16 Wort tiefes FIFO zur Verfügung, das nach ≈20 PCI-Zugriffen gefüllt ist. Der Burstzugriff auf dem PCI-Bus wird daraufhin abgebrochen und der Bus wird für andere Übertragungen freigegeben. Beim Erreichen einer unteren Füllgrenze fordert der PCI-Interfacebaustein den Bus erneut an, um einen weiteren Burst von ≈20 Zugriffen durchzuführen. Die Auswirkung dieser kurzen Burstzugriffe zeigt sich bei einem direkten Vergleich der Übertragungszeiten. Die Übertragung der

[4]Der eingesetzte FPGA-Baustein XV400 verfügt über eine Konfigurationsbitstrom von 311KByte. Bei einer Ansteuerung mit 40MHz ergibt sich die oben genannte Zeit.

311kByte Konfigurationsdaten zur Konfiguration des FPGA-Bausteins XV400 benötigt 12ms, wobei eine reine Übertragung der Daten zum Interfacebaustein nur ≈4ms benötigen.

Aufgrund dieser Messungen und Erkenntnisse ist es notwendig bei der Übertragung der Konfigurationsdaten darauf zu achten, daß diese über die Kapazitäten des PCI-Interfacebausteins hinaus zwischengespeichert werden. Durch eine solche Zwischenspeicherung kann garantiert werden, daß der Konfigurationsdatenstrom ohne Unterbrechungen geladen werden kann und somit eine minimale Konfigurationzeiten erreicht wird. Zusätzlich wird auch der PCI-Bus effektiver genutzt, da größere Datenbursts übertragen werden.

Softwareoptimierung: Eine weitere Möglichkeit, die Prozesswechelzeit zu verringern besteht innerhalb der FPGA-Bibliotheksfunktionen, die die Konfigurations- bzw. den Readbackfunktionen umsetzen. Bei jeder Übertragung der Konfigurationdaten müssen die einzelnen Speicherpages im Speicher gehalten werden, damit sie z.B. mit einem DMA-Transfer ausgelesen werden können. Das Einlagern und Halten dieser Speicherpages wird im Rahmen der Vorbereitungen des DMA-Transfers durchgeführt und nach Beendigung des Transfers werden die Speicherpages von der Funktion wieder freigegeben.

Aufgrund der Häufigkeit in der dieser Speicher genutzt wird und die weitere zu erwartende Minimierung der Prozesswechselzeit werden die Speicherpages schon beim Anlegen im Speicher so gekennzeichnet, daß sie permament verfügbar sind und nicht vom Host-Betriebssystem ausgelagert werden können.

Alle zuvor ausführlich beschrieben Punkte zur Verbesserung der Prozesswechselzeit wurden auf dem neuen FPGA-Koprozessor realisiert. Als Prozessorelement des FPGA-Koprozessor kommt der beschriebene Virtex XV300 FPGA-Baustein von Xilinx zum Einsatz. Die Anbindung der Konfigurationsschnittstelle dieses Bausteins an den PCI-Bus erfolgt über einen schnellen PCI-Interfacebaustein und ein spezielles FIFO, das die Zwischenspeicherung der Konfigurationsdaten und gleichzeitig auch die Umsetzung der 32Bit breiten PCI-Datenworte auf vier 8Bit breite

Datenworte durchführt. Die gesamte Anbindung mit den zum Einsatz kommenden Komponenten ist in Abbildung 6.3 noch einmal graphisch dargestellt.

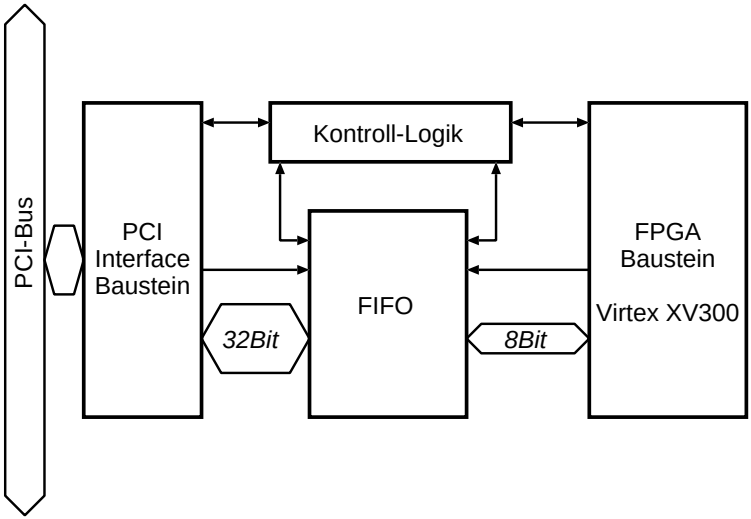


Abbildung 6.3: Anbindung der Konfigurationsschnittstelle des FPGA-Bausteins an den PCI-Bus. *Das zwischengeschaltete FIFO speichert die Daten und wandelt die 32Bit Datenwörter in vier aufeinanderfolgende 8Bit Datenwörter um.*

Die Größe des Konfigurations- und des Readbackbitstroms des eingesetzten FPGA-Baustein XV300 umfasst insgesamt 214kByte die über die 8Bit breite Konfigurationsschnittstelle geladen und zurückgelesen werden. Wie in der Abbildung zu erkennen ist, arbeitet die Verbindung zwischen dem FIFO und dem FPGA-Bausteins unabhängig von dem PCI-Bus und wird nur durch die Kontroll-Logik gesteuert. Getaktet wird diese Verbindung mit einem eigenen Takt der derzeit mit 40MHz arbeitet. Unter der Voraussetzung, daß das FIFO die gesamten Daten gespeichert hat ergibt sich bei 40MHz eine theoretische Konfigurationszeit von 5,5ms.

Das FIFO, das von dem PCI-Interfacebaustein mit 32Bit Datenwörtern gefüllt wird, kann insgesamt 1024 Datenworte zwischenspeichern. Die Daten werden mit einer Taktrate von 33MHz geschrieben, so daß die gesamte PCI-Datenrate erreicht werden kann. Bei dem gegebenen Taktverhältniss können somit bis zu 1377 Datenworte (≈5,5kByte) in einem Burst über den PCI-Bus übertragen werden. Der PCI-Zugriff wird unterbrochen, sobald das FIFO bis auf 64 Werte gefüllt ist und wiederaufgenommen wenn weniger als 64 Worte in FIFO gespeichert sind. Durch diesen vergleichsweise tiefen Speicher wird ein unterbrechungsfreier Konfigurationsdatenstrom gewährleistet, auch wenn der PCI-Bus gerade eine andere Übertragung durchführt. Die maximale Pufferzeit beträgt 6.4μs.

Aus Sicht des FPGA-Betriebssystems erhöht sich die Zeit z.B. für die Konfiguration noch durch den Overhead der DMA-Übertragung und die Kontrollzugriffe zur Überprüfung der korrekten Konfiguration des Bausteins. Insgesamt ist dieser notwendige Overhead auf das Mindestmaß beschränkt und beeinflußt die Gesamtzeit nur minimal. Durchgeführte Messungen mit dem System und den Funktionen sind in Abschnitt 6.4 beschrieben.

6.3.3 RAM-Switch

Das dritte Konzept, das auf dem neuen FPGA-Koprozessor realisiert wurde, ist der RAM-Switch, der die Prozesswechselzeit auf ein absolutes Minimum reduziert. Diese Reduktion ist zurückzuführen auf zwei Eigenschaften die der RAM-Switch umsetzt: Die direkte Verbindung des PCI-Busses mit den einzelnen RAM-Bänken und ein anwendungsabhängiges Verbinden bzw. Abtrennen der einzelnen RAM-Bänke durch den RAM-Switch.

Die direkte Verbindung des PCI-Busses mit den RAM-Bänken ermöglicht dem Host-Prozessor ein Auslesen der Daten, ohne das Konfigurieren des FPGA-Bausteins. Dadurch wird eine direkten Sicherung und Rekonstruktion der Daten aus dem RAM möglich. Die direkte Verbindung kann darüberhinaus noch effektiver genutzt werden, da aufgrund des RAM-Switchs ein direkter Zugriff auf die von der FPGA-Schaltung

nicht genutzen RAM-Bänke ermöglicht wird. Diese parallele Nutzung der Übertragungswege wird von dem neuen speicherbasierten Ausführungsmodell genutzt und beschleunigt zusätzlich die gesamte Anwendungsausführung.

Im Gegensatz zu der parallelen Nutzung der Datenpfade wirkt sich die zweite Eigenschaft des RAM-Switchs direkt auf die Prozesswechelzeit aus. Durch ein Zuweisen der einzelnen RAM-Bänken zu den Anwendungen, die auf dem FPGA-Koprozessor verarbeitet werden, ist die aufwendige Sicherung und Rekonstruktion der RAM-Inhalte nicht mehr notwendig. Die RAM-Bänke werden bei einem Prozesswechel durch den RAM-Switch von der entsprechenden RAM-Schnittstelle elektrisch abgetrennt und erst wieder bei der erneuten Ausführung der Anwendung angeschlossen. Während der Ausführung der anderen Anwendungen ist die RAM-Bank deaktiviert und der entsprechende Inhalt wird nicht verändert. Die Prozesswechselzeit verringert sich somit um ≈33ms,[5] da eine Sicherung und Rekonstruktion des Inhalts nicht mehr notwendig ist. In der nachfolgenden Abbildung 6.4 ist der Aufbau des RAM-Switchs dargestellt. Für den Zugriff auf die RAM-Bänke verfügt der Baustein über drei Schnittstellen die jeweils mit einem eigenen Bus innerhalb des RAM-Switchs verbunden sind. Eine Schnittstelle ist verbunden mit dem PCI-Interfacebaustein um einen direkten und parallel arbeitenden Zugriff des Host-Prozessors auf die RAM-Bänke zu ermöglichen. Die anderen beiden Schnittstellen verbinden die FPGA-Schaltung mit den ausgewählten RAM-Bänken und arbeiten unabhängig voneinander. Die elektrische Verbindung der einzelnen RAM-Bänke werden über sogenannte 'Bidirektionale Multiplexer' hergestellt. Diese programmierbaren Multiplexer verbinden die Steuer-, Address- und Datenleitungen des ausgewählten Busses mit dem eigentlichen RAM-Baustein und ermöglichen somit ein dynamisches Wechseln der RAM-Bänke abhängig von der jeweils zu verarbeitenden Anwendung. Aufgrund des lesenden wie auch schreibenden Zugriffs auf die RAM-Bänke sind die Multiplexer für einen bidirektionalen Datenaustausch auszulegen. Gesteuert werden die Multiplexer durch ein Steuerelement, das entsprechend der Situation und

[5] Ausgehend von einer PCI-Datentransferrate von 120MByte/s und einem Speicherumfang von insgesamt 2MByte.

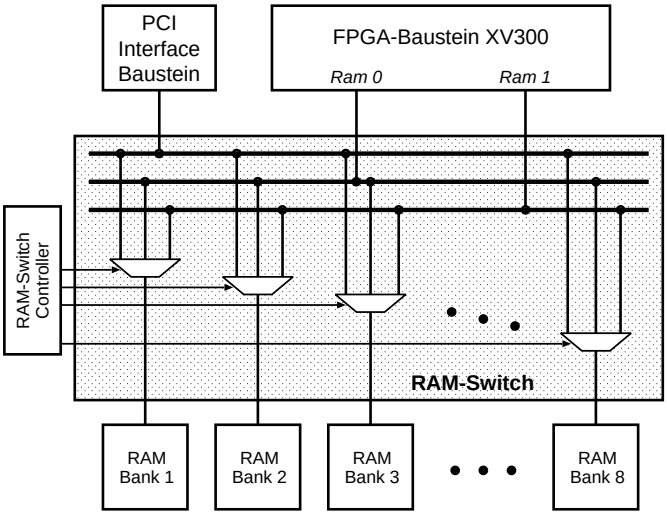


Abbildung 6.4: Aufbau und Architektur des RAM-Switchs.

der zu verarbeitenden Anwendung die RAM-Bänke entweder mit den FPGA-Schnittstellen oder mit der PCI-Schnittstelle verbindet oder die RAM-Bank abtrennt und dadurch inaktiv schaltet.

Bezüglich der Realisierung des RAM-Switchs und der Auswahl der RAM-Bausteine sind bestimmte Randbedingungen und Anforderungen einzuhalten, um zum einen die Prozesswechselzeit durch ein Wechsel des RAM-Switchs nicht unnötig zu verlängern und zum anderen um eine für die FPGA-Schaltung optimale Anbindung bereitzustellen. Diese Anforderungen betreffen folgende Punkte: Maximale Taktfrequenz, direkter Zugriff auf die RAM-Bank, schnelle Programmierbarkeit des RAM-Switchs und die Flexilität durch den Switch.

Maximale Taktfrequenz: Die Analyse verschiedener FPGA-Schaltungen und Anwendungen hat gezeigt, daß viele Anwendungen aufgrund der Speicherbandbeite in ihrer Beschleunigung begrenzt sind. Durch eine Verdoppelung der Speicherbandbreite, die z.B. bei der

Bildverarbeitung eingesetzt wird, kann die Anwendung einfach um einen weiteren Faktor 2 beschleunigt werden.

Diese Doppeltaktung soll auch auf dem neuen FPGA-Koprozessor ermöglicht werden. Somit ergibt sich für den RAM-Switch eine maximale Arbeitsfrequenz von 100MHz bei den zu erwartenden Taktfrequenzen zwischen 40MHz und 60MHz. Diese Anforderung sollte sowohl durch das Timing innerhalb des RAM-Switchs, als auch durch die an den RAM-Switch angeschlossenen Bausteine wie der FPGA-Baustein und die RAM-Bausteine selbst erfüllt sein.

Direkter Zugriff: Unabhängig von der Taktrate ist auch der direkte Zugriff auf das RAM für einen maximalen Datentransfer entscheidend. Viele Anwendungen, z.B. aus der Datenkompression oder der Bildverarbeitung, arbeiten in einem sogenannten 'Read-Modify-Write' Zyklus der ein Datenwort von Speicher liest, das Datenwort verändert und es sofort danach wieder abspeichert.

Zum Erreichen des maximalen Datendurchsatzes bei diesem Verarbeitungszyklus in speziellen, aber auch bei der Datenverarbeitung im allgemeinen ist bei der Realisierung des RAM-Switchs und der Auswahl der RAM-Bausteine darauf zu achten, daß keine zusätzlichen Waitstates entstehen die diesen Datendurchsatz mindern.

Kurze Programmierzeit: Aufgrund der Tatsache, daß der RAM-Switch nach dem Zurücklesen des FPGA-Schaltungszustands, aber vor der erneuten Konfiguration der nachfolgenden FPGA-Schaltung durchzuführen ist, muß die dazu notwendige Zeit mit zu der Prozesswechselzeit hinzugezählt werden. Somit ist ein schnelles Umschalten des RAM-Switchs notwendig damit das Ziel einer minimalen Prozesswechselzeit erreicht bleibt.

Flexibilität: Neben den Anforderungen, die einen direkten Einfluß auf die Leistungsfähigkeit des RAM-Switchs und die Prozesswechelzeit haben, erweitert die Forderung nach einer hohen Flexibilität die Einsatzmöglichkeiten des neuen FPGA-Koprozessors.

Die Verwendung einer RAM-Bank aus zwei unabhängigen Anwendungen oder FPGA-Schaltungen heraus soll auch auf der Ebene

der Hardware einen einfachen Datenaustausch nach dem Vorbild des 'Shared Memory' ermöglichen. Zusätzlich soll eine flexible Zusammenschaltung von mehreren RAM-Bänken an eine Schnittstelle ermöglicht werden, um den zum Teil sehr unterschiedlichen Anwendungsanforderungen nachzukommen.

Für die Realisierung des RAM-Switchs stehen drei unterschiedliche Basiskomponenten zur Auswahl, die unter Berücksichtigung der gestellten Anforderungen entsprechend bewertet werden können. Im einzelnen werden zur Umsetzung des RAM-Switchs programmierbare Logikbausteine, programmierbare Verbindungsbausteine und Busschalter (Quick-Switches) untersucht.

6.3.3.1 Complex Programmable Logic Devices (CPLDs)

Die CPLD-Bausteine sind wie die FPGA-Bausteine frei programmierbar aber speichern ihre Konfiguration fest ab. Im Rahmen des RAM-Switchs wird die Funktionalität der 'bidirektionalen Multiplexer' innerhalb der CPLD-Bausteine umgesetzt und entsprechend an die drei Busse und die acht RAM-Bausteine angeschlossen.

Die Untersuchung der Signalverzögerungszeit durch den CPLD-Baustein gibt Aufschluß über die maximal erreichbare Taktfrequenz und über den zu erreichenden Durchsatz. Die Signallaufzeiten betragen bei modernen CPLDs 6ns (Xilinx XC95288XL-6). Zusammen mit den 6ns Ausgangsverzögerung des Virtex FPGAs (XV300-4) und den 2ns Setup Zeit des RAM-Bausteins (IDT71V547) ergibt sich eine minimale Zugriffszeit von 14ns und daraus resultierend eine maximale Frequenz von ≈70MHz. Durch das Einfügen einer weiteren Registerstufe innerhalb des CPLDs kann diese maximale Frequenz zwar auf die geforderten 100MHz erhöht werden, aber durch diese Registerstufe wird auch ein direkter Zugriff unterbunden was den gesamten Datendurchsatz verringern kann. Zusätzlich ist anzumerken, daß ein CPLD-Baustein den bidirektionalen Datenbus nur durch Abschalten der Ausgangstreiber realisieren kann. Die Verzögerung dieser Ausgangstreiber beträgt bei dem gewählten CPLD 8ns und reduziert die maximal erreichbare Frequenz ohne Register auf 62MHz.
Die Zeit die zur Konfiguration der CPLD-Bausteine benötigt wird, liegt

für den ausgewählten Baustein XC95288XL-6 bei mehr als einer Minute und macht ein dynamisches Umkonfigurieren des Bausteins daher unmöglich. Die Funktionalität des Multiplexers wird durch den CPLD bereitgestellt und über individuelle Steuerleitungen werden die Multiplexer einzeln angesprochen. Dies ermöglicht ein Umschalten des RAM-Switchs innerhalb von $\approx 10\mu$s.
Ein drittes Kriterium, das zur Bewertung herangezogen wird, ist die Auslastung des Bausteins. Die interne Logik des CPLDs ist in der nachfolgenden Abbildung 6.5 vereinfacht dargestellt. Insgesamt verfügt je-

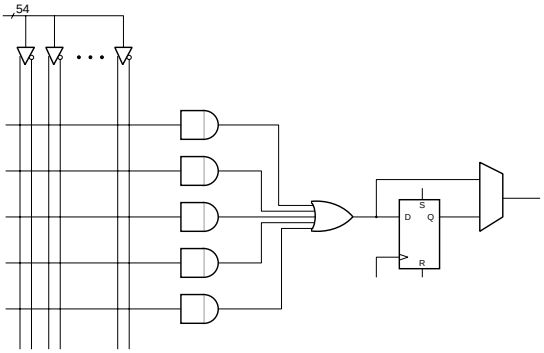


Abbildung 6.5: Interner Aufbau des Xilinx XC95288 CPLDs. *Die Abbildung zeigt den vereinfachten internen Aufbau einer Macrozelle.*

de Macrozelle über fünf Produktterme mit jeweils 54 Eingängen. Die 5 Produktterme werden ODER verknüpft und entweder in einem Register zwischengespeichert oder direkt als kombinatorisches Signal ausgegeben. Bei der Abbildung des Multiplexers werden für eine Signalleitung nur sechs der 54 Eingänge benötigt die in drei Produkttermen zusammengefaßt werden und aufgrund der direkten Anbindung direkt ausgegeben werden. Diese Verschaltung der internen Zellen führt zu einer nur minimalen Auslastung der Bausteine die bei unter 20% liegt.
Insgesamt sind aufgrund der sehr hohen Anzahl an zu verbindenden Pins 18 CPLD-Bausteine mit jeweils 117 Pins notwendig um die gesamte Funktionalität des RAM-Switchs umzusetzen.

6.3.3.2 Field Programmable Interconnects (FPICs)

FPIC-Bausteine werden eingesetzt, um dynamisch veränderbare Signal- oder Busverschaltungen zu realisieren. Sie basierend wie die FPGA-Bausteine auf einem statischen RAM und sind dadurch frei rekonfigurierbar. Die interne Switch-Matrix ermöglicht sowohl ein 1-zu-1 als auch eine N-zu-N Verbindung in verschiedenen Busbreiten und mit fest definierten Verzögerungszeiten. Dargestellt ist das Blockschaltbild eines FPIC-Bausteins in der nachfolgenden Abbildung 6.6.

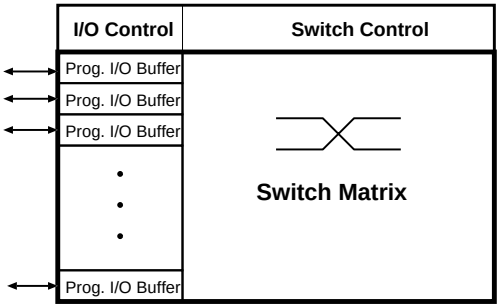


Abbildung 6.6: Vereinfachtes Blockdiagram des Field Programmable Interconnects von ICube.

Die Signalverzögerung des FPIC-Baustein ist fest definiert und für alle Bussignale identisch. Sie beträgt für den untersuchten FPIC-Baustein der Firma ICube (IQ160-100) 10ns. Zusammen mit den 6ns Ausgangsverzögerung des FPGA-Bausteins und der 2ns Setup Zeit des RAMs ergibt sich für die direkte Ansteuerung des RAMs eine maximale Frequenz von 55MHz. Wie auch bei der CPLD Lösung kann die Frequenz durch das Zwischenschalten einer Registerstufe verbessert werden, wodurch sich aber der gesamte Datendurchsatz vermindert. Ein 'Read-Modify-Write' Zyklus verlängert sich durch zusätzlich Waitstates von drei auf fünf Takte.
Zur Realisierung des bidirektionalen Multiplexers, der den Datenbus mit dem RAM verbindet, kann der FPIC in dem sogenannten Bus-Repeater Mode geschaltet werden. In diesem Mode sind die I/O Buffer als Ver-

bindung geschaltet, die ein am Eingang getriebenes Signal erkennen und es in die Switch-Matrix weiterleiten. Bei einem Betrieb in dem Bus-Repeater Mode sind somit weitere Steuersignale, die die Übertragungsrichtung angeben, nicht notwendig. Die folgende Abbildung 6.7 zeigt eine solche Bus-Repeater Verbindung im FPIC-Baustein.

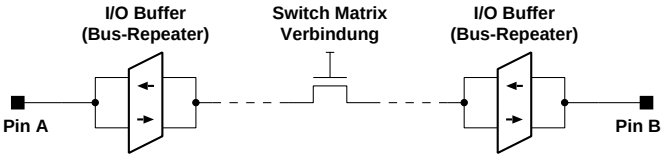


Abbildung 6.7: Bus-Repeater Verbindung innerhalb eines FPIC-Bausteins von ICube.

Die gesamte Signalverzögerung durch den FPIC beträgt auch bei Anwendung der Bus-Repeater Modes 10ns.
Aufgrund von fehlenden logischen Elementen in dem FPIC können die einzelnen Verbindungen nur durch ein Umkonfigurieren der Switch-Matrix erfolgen. Zum Konfigurieren dieser Matrix stehen am FPIC-Baustein zwei unabhängige Schnittstellen zur Verfügung. Zum einen ist das eine JTAG Schnittstelle über die der gesamten Baustein innerhalb von 2ms konfiguriert werden kann und zum anderen eine schnelle parallele Schnittstelle. Sie ermöglicht das Verändern einzelner Verbindungen innerhalb der Switch-Matrix so wie es für die Funktionalität des bidirektionale Multiplexers benötigt wird. Die Veränderung erfordert pro RAM-Baustein ≈500 Zugriffe, die bedingt durch die PCI-Schnittstelle innerhalb von 16μs ausgeführt werden können. Für die Realisierung des gesamten RAM-Switchs werden 13 FPIC-Bausteine (IQ160-100) benötigt, die den gesamten RAM-Switch innerhalb von nur 200μs umkonfigurieren können.

6.3.3.3 Quick-Switches

Die dritte Basiskomponente, die zur Realisierung des RAM-Switchs untersucht wurde, sind Busschalter oder Quick-Switches. Dabei handelt es

sich um schaltbare Elemente, die je nach anliegenden Steuersignalen eine elektrische Verbindung erstellen.

Der interne Aufbau der Quick-Switches ist in der Abbildung 6.8 dargestellt. Wie in der Abbildung zu erkennen ist, werden die einzelnen Ver-

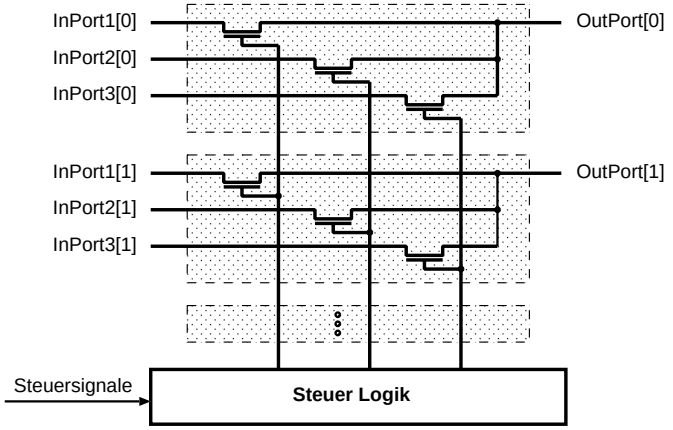


Abbildung 6.8: Interner Aufbau eines Quick-Switch Bausteins.

bindungen zwischen den drei unabhängigen 'Eingangs'-Ports und dem 'Ausgangs'-Port durch Feld-Effekt Transistoren (FET) hergestellt. Über drei Steuerleitungen wird ein entsprechendes Gatesignal am FET erzeugt, so daß entweder eine Verbindung zu Port1, Port2 oder Port3 hergestellt wird oder alle drei 'Eingangs'-Ports von dem 'Ausgangs'-Port abtrennt sind. Der durch die Verbindung enstehende Widerstand beträgt maximal 12Ω und kann bei den hohen Eingangsimpedanzen der aktiven Bausteine vernachlässigt werden.

Die Signale von dem Bus werden beim Einsatz der Quick-Switches nur minimal verzögert, da nur eine direkte Verbindung zwischen zwei Ports erstellt wird. Die angegebene und gemessene Verzögerungszeit beträgt daher nur 250ps. Bedingt durch diese minimale Signalverzögerung ist somit ein Betrieb des RAM-Switchs und der RAM-Bausteine mit den geforderten 100MHz kein Problem, da die gesamte Verzögerung nur

8ns beträgt. Darüberhinaus werden trotz zwischengeschaltetem RAM-Switch die RAMs direkt angesprochen und können somit bei allen Zugriffsarten mit dem vollen Datendurchsatz betrieben werden.
Die Quick-Switch Bausteine sind durch ihren internen Aufbau auch für den bidirektionalen Datenaustausch geeignet, da sie nicht aktive als Signalverstärker oder Treiber agieren.
Das Umschalten der einzelnen Verbindungen wird durch eine Änderung der Steuerleitungen gesteuert. Die Quick-Switch Bausteine selbst benötigen nur 5ns um eine Verbindung aufzubauen oder zu trennen. Dazu kommt noch die Übertragung der neuen, 40Bit umfassenden Steuerinformation von Betriebssystem zu dem FPGA-Prozessor, so daß insgesamt weniger als 10μs benötigt werden, um den RAM-Switch umzuschalten.

Der gesamte RAM-Switch umfasst neben den Quick-Switch Bausteinen auch die drei Busse, die dynamisch an die einzelnen RAM-Bausteine angeschlossen werden müssen. Zum Verbinden einer RAM-Bank sind fünf Quick-Switch Bausteine notwendig, so daß insgesamt 40 Quick-Switch Bausteine eingesetzt werden müssen.

Aufgrund der Signalverteilung ist ein Bussignal an jedem bidirektionalen Multiplexer angeschlossen. Im Betrieb ist z.B. eine RAM-Bank über den Multiplexer an einen Bus angeschlossen und die anderen sieben Multiplexer sind inaktiv bzw. abgetrennt. Obgleich die Eingänge abgetrennt sind, wirken sie weiterhin als eine kapazitive Last, die das Bussignal verformen können, so daß es nicht mehr korrekt erkannt wird. Zur Ermittlung der Auswirkungen dieser kapazitiven Last wurden Messungen durchgeführt. Die Details der Messungen und die Ergebnisse sind in Anhang A ausführlich dargestellt. Es konnte gezeigt werden, daß bei der entstehenden kapazitiven Last die Signale zwar verformt werden, aber trotzdem noch in der vorgegebenen Toleranzgrenze liegen. Die Messungen wurden dabei bis zu der angeforderten maximalen Frequenz von 100MHz durchgeführt.

6.3.3.4 Zusammenfassung

Wie die Untersuchung und die anschließende Bewertung der drei Basiskomponenten gezeigt hat, ist die Umsetzung der Funktionalität des bidirektionalen Multiplexers mit allen drei Basiskomponenten möglich.

Darüberhinaus werden auch die gestellten Flexibilitätsanforderungen von allen drei Modellen erfüllt.

Die weiteren für die Auswahl wichtigen Anforderungen umfassen die Zeit die zum Umschalten benötigt wird, die maximale Taktfrequenz innerhalb des RAM-Switchs und die Zugriffe die keine Waitstates erfordern. Die ermittelten Werte sind in der nachfolgenden Tabelle 6.1 zusammengefaßt:

Basis-komponenten	Programmier Zeit	Maximale Frequenz	Direkter Datentransfer
CPLD	10μs	62MHz	Ja
CPLD (Reg)	10μs	100MHz	Nein
FPIC	200μs	55MHz	Ja
FPIC (Reg)	200μs	100MHz	Nein
QuickSwitch	10μs	100MHz	Ja

Tabelle 6.1: Zusammenfassung der ermittelten Bewertungskriterien für die untersuchten Basiskomponenten.

Durch den Einsatz des CPLD-Baustein ist der Aufbau eines RAM-Switchs möglich, der die Verbindungen sehr schnell anlegen und trennen kann. Dennoch ist der CPLD aufgrund der nicht erreichbaren maximalen Taktfrequenz bei einer direkten Verbindung und der Problematik mit den bidirektionalen Verbindungen nicht zum Aufbau des RAM-Switchs geeignet. Darüberhinaus entsteht durch die niedrige Auslastung der Bausteine ein hoher Kostenfaktor.

Die FPIC-Bausteine sind für den Einsatz in dem RAM-Switch konzipiert und verfügt über einen speziellen Betriebsmode, der ein bidirektionales Verbinden ermöglicht. Dennoch ist auch der Einsatz dieser Bausteine eingeschränkt, da durch die Signalverzögerung ohne Zwischenspeicherung nur eine maximale Frequenz von 55MHz erreicht werden kann. Desweiteren entsteht durch die, im Vergleich zu den CPLD und QuickSwitch Bausteinen, lange Umschaltzeit eine zusätzliche Verzögerung bei der Prozesswechselzeit.

Die Quick-Switch Bausteine haben gezeigt, das mit ihnen ein RAM-Switch aufgebaut werden kann, der sehr schnell arbeitet. Dies betrifft sowohl die benötigte Umschaltzeit, die nur 10μs beträgt, als auch die maximale Taktfrequenz von 100MHz mit der direkt auf die RAM-Bänke zugegriffen werden kann. Durch den internen Aufbau der Quick-Switch Bausteine erfolgt die Realisierung der bidirektionalen Busverbindung direkt durch die Bausteine, wobei eine nahezu 100% Auslastung erreicht wird.

Aufgrund dieser Ergebnisse der Untersuchung wurde der RAM-Switch auf dem neuen FPGA-Koprozessor mit den Quick Switch Bausteinen realisiert. Die Abbildung 6.9 illustriert das genaue Verbindungsschema, daß einen einzelnen RAM-Baustein mit den drei Bussen verbindet.

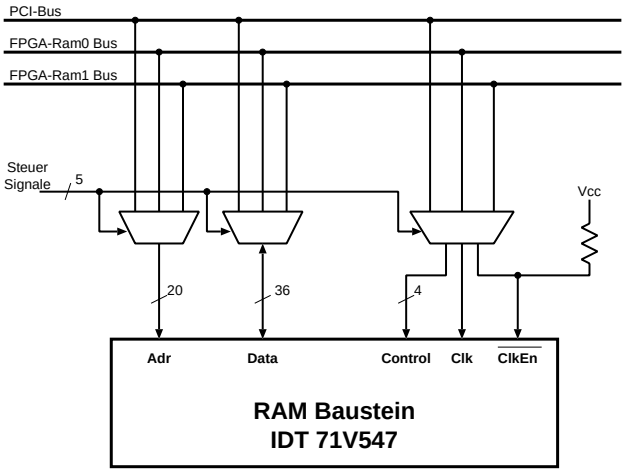


Abbildung 6.9: Funktionsschaltbild der Anbindung einer RAM-Bank an die drei RAM-Switch Busse.

Die Bussignale werden über mehrere Quick-Switch Bausteine direkt mit dem RAM-Baustein verbunden, sobald die Steuerlogik die einzelnen Switch Bausteine angesteuert. Die Steuersignale sind synchron zu-

einander und schalten den Switch zum gleichen Zeitpunkt um. Zusätzlich übernimmt die Steuerlogik die Überwachung der Steuersignale, die ein versehentliches Verbinden von mehreren gleichadressierten RAM-Bänken an einen Bus verhindert. Ausgewählt wird eine RAM-Anbindung über fünf Steuerleitungen, die zum einen die Busverbindung definieren und zum anderen einen Offsetadressierung der einzelnen RAM-Bänke ermöglichen. Dieser Adressoffset ist notwendig bei der kontrollierten Zusammenschaltung mehrerer RAM-Bänke zu einem großen RAM-Block.

Die Reduzierung der Prozesswechselzeit durch den Einsatz des RAM-Switchs wird möglich, da der RAM-Inhalt einer abgetrennten RAM-Bank nicht verändert wird. Aufgrund der eingesetzten statischen RAM-Bausteine ist bei einem Abtrennen nur darauf zu achten, dass keine 'Write'-Zugriffe auf den RAM-Baustein erfolgen. Erreicht wird dies durch die Deaktivierung des 'Clock-Enable' Signals, das im inaktiven Zustand keinen Zugriff – weder lesend noch schreibend – auf den Baustein ermöglicht. Dieser Eingang wird über einen zusätzlichen 'Pull-Up' Widerstand inaktiv geschaltet, sobald der RAM-Baustein von einem der drei Busse abgetrennt ist und garantiert somit, das die gespeicherten Daten nicht verändert werden.

6.3.3.5 RAM-Switch Einstellungen

Neben dem sicheren Abtrennen der einzelnen RAM-Bänke kann der RAM-Switch auch so konfiguriert werden, daß entweder eine RAM-Bank von mehreren FPGA-Schaltungen genutzt wird oder das eine FPGA-Schaltung durch die Schnittstelle auf ein größeres RAM zugreifen kann.

Das Nutzen einer RAM-Bank durch verschiedene Anwendungen ist eine sehr effektive Datenübertragungsart. So kann z.B. bei der Komprimierung von Daten zuerst das Histogramm von verschiedenen Datensätzen erzeugt werden, um dann in einem zweiten Schritt die Datensätze zu kodieren. Das in dem ersten Schritt erzeugte Histogramm kann in einem eigenen RAM-Block gespeichert und einfach von beiden Teilschritten genutzt werden. Im allgemeineren Sinn ist die Verwendung einer solchen geteilten RAM-Bank mit der 'Pipe'-Operation in Unix vergleichbar, wobei die RAM-Bank zur Übergabe der Daten von einer zur nächsten Anwendungen genutzt wird.

Die zweite, sehr nützliche Eigenschaft des RAM-Switchs wird durch die Zusammenschaltung von zwei oder mehr RAM-Bänken erreicht. Die Schnittstellen zu dem FPGA (RAM 0 und RAM 1) umfassen neben den Adress- und Steuerleitungen je einen 72Bit breiten Datenbus. Somit ist es möglich, durch das Zusammenschalten von zwei jeweils 36Bit breiten RAM-Bänken der FPGA-Schaltung ein 72Bit breites RAM zur Verfügung zu stellen. Die vom RAM-Switch angelegten Verbindungen schalten jeweils die Adress- und Steuerleitungen an beide RAM-Bänke an und nur die Datenleitungen werden unterschiedlich verbunden. Durch die unterschiedliche Offsetadressierung ist es ebenfalls möglich die Speichertiefe des angeschlossenen RAMs zu vergrößern.

Diese dynamische RAM-Anpassung, die für alle drei Zugriffsschnittstellen umgesetzt ist, wird erst durch den RAM-Switch möglich. Dadurch ist der Entwickler in der Lage für die verschiedensten Anwendungen und deren jeweilige RAM-Anforderungen die passendste auszuwählen, um ein Maximum an Leistungsfähigkeit zu erreichen. Unter Verwendung aller acht RAM-Bänke mit jeweils bis zu 2MByte Speicherplatz ist aus Sicht der FPGA-Schaltung ein sehr tiefer Speicher mit maximal 4Mega-Addressen (1 x 4M x 36Bit) aber auch ein breiter Speicher mit maximal 144Bit Daten (1 x 1M x 144Bit) einstellbar. Bei der Nutzung aller RAM-Bänke durch eine FPGA-Schaltung ist der Multitasking Betrieb eingeschränkt, denn aufgrund von einer eventuell entstehenden doppelten Nutzung einer RAM-Bank ist die Sicherung und Rekonstruktion des RAM-Inhalts notwendig und erhöht damit die Prozesswechelszeiten.

6.4 Messungen

Zur Bewertung der Verbesserungen, die von dem neuen FPGA-Koprozessor unterstützt werden, wurden auch hier mehrere Messungen durchgeführt. Im speziellen wurden mit diesen Messungen die Zeiten, die für die Konfiguration und den Readback des FPGA-Bausteins benötigt werden, aber auch die Auswirkungen des RAM-Switchs auf auf die Signalverfäufe ermittelt. Die im einzelnen durchgeführte Messungen sind in dem nachfolgenden Abschnitten kurz erläutert und bewertet.

Zur Durchführung der einzelnen Messungen wurden der gleiche Mes-

saufbau verwendet, der zur Ermittlung der Prozesswechselzeiten des Client-Server Betriebssystem eingesetzt wurde. Er ist in in Abschnitt 5.6.1 auf Seite 164 beschrieben ist.

6.4.1 Konfigurationsschnittstelle

Die Zeiten die zur Konfiguration und zum Readback des FPGA-Bausteins benötigt werden bestimmen maßgeblich die gesamte Prozesswechselzeit des FPGA-Koprozessors und damit auch den systematischen Overhead des FPGA-Betriebssystems. Um diese Prozesswechselzeit auf ein Minimum zu beschränken wurde auf dem neuen FPGA-Koprozessor die in Abbildung 6.3 illustrierte Anbindung der Konfigurationsschnittstelle umgesetzt.

Die theoretische Konfigurationszeit bei dem eingesetzten FPGA-Baustein XV300 ergibt sich aus der Taktrate mit der die Konfigurationsschnittstelle getaktet wird und der Datenmenge die bei der Konfiguration übertragen wird. Bei der Taktrate von 40MHz erfolgt die Konfiguration des 214KByte umfassenden Bitstroms in 5,5ms. Mit dem Versuchsaufbau wurden mehrere Messungen durchgeführt, bei denen jeweils die Zeit für die gesamte Konfigurationsfunktion gemessen wurde. Die Funktion umfasst neben der Übertragung der Daten auch das Zurücksetzen der Schnittstelle und die abschließende Kontrolle der Konfiguration. Die durchschnittliche Konfigurationszeit, die mit dem neuen FPGA-Koprozessor gemessen wurden, liegen bei nur 6,2ms.

Diese Konfigurationszeiten zeigen die Auswirkungen der optimalen Anbindung der Konfigurationsschnittstelle, denn die Ausführung der Funktion ist nur ca. 12% länger als die theoretische Konfigurationszeit. Die Verlängerung entstehen durch die Kontroll- und Steuerungszugriffe die für die Konfiguration notwendig sind. Die Verbesserung zeigt sich auch im Vergleich der schnellsten Funktionszeiten bei dem FPGA-Koprozessor microEnable II die mehr als 50% über der theoretischen Konfigurationszeit liegt (Siehe auch Abschnitt 5.8).

Durch die optimale Anbindung werden sowohl die Zeiten für die Konfiguration als auch die Zeiten für den Readback minimiert. Wie bei der Konfiguration werden auch beim Readback 214KByte Daten mit der

gleichen Frequenz übertragen, wodurch sich auch hier eine theoretische Readbackzeit von 5,5ms ergibt. Der gesamte Readback ist jedoch in drei Bereiche unterteilt. Daher erfolgt die gesamte Übertragung in drei unterschiedlich lange Datenpakete die jeweils einzeln aktiviert und vorbereitet werden müssen. Insgesamt benötigt der durchschnittliche Readback 9,1ms.

Wie zuvor bei den Konfigurationszeiten wirkt sich auch hier die Anbindung an die Konfigurationsschnittstelle positiv auf die Readbackzeiten aus. Aufgrund der zweifach unterbrochenen Übertragung entsteht jedoch eine zusätzliche Ausführungszeit, die die gesamte Readbackfunktion im Vergleich zu der theoretischen Übertragungszeit um ca. 65% verlängert.

Im Ergebnis sind die Zeiten für die Konfiguration und den Readback gegenüber dem FPGA-Koprozessor microEnable II deutlich verbessert worden. Da die Prozesswechselzeit maßgeblich durch diese beiden Zeiten bestimmt wird ergibt sich auch eine deutliche Verbesserung der gesamten Prozesswechselzeit. Unter Berücksichtigung der Softwareoptimierung wird ein vollständiger Prozesswechsel in weniger als 20ms durchgeführt. Die zusätzlichen 5ms ergeben sich durch die Einstellungen der Taktgeneratoren, das Umschalten des RAM-Switchs und weitere Steueraufgaben des FPGA-Betriebssystems.

6.4.2 RAM-Switch

Bei der Verwendung des FPGA-Koprozessor microEnable II hat das FPGA-Betriebssystem zusätzlich die Aufgabe den Inhalt der lokalen RAMs auf dem FPGA-Koprozessor zu sichern und zu rekonstruieren. Durch den RAM-Switch, der auf den neuen FPGA-Koprozessor realisiert wurde, werden die RAM-Bänke vom Bus abgetrennt und inaktiv geschaltet. Diese Vorgehensweise reduziert die Prozesswechselzeit auf die im vorherigen Abschnitt genannten 20ms und ermöglicht zusätzlich einen parallelen Zugriff des FPGA-Betriebssystems auf die derzeit nicht genutzen RAM-Bänke. In einer weiteren Messung wurden daher zum einen die Funktionalität des RAM-Switchs getestet und zum anderen wurden die Signalverformungen auf dem FPGA-Koprozessor gemessen um die ma-

ximale Taktrate des RAM-Switchs zu ermitteln.

Die Kontrolle der Funktionalität wurden unterteilt in drei Messungen die getrennt voneinander ausgeführt wurden. Im einzelnen wurden die folgenden Funktionen überprüft:

RAM-Switch Steuereinheit: In diesem ersten Funktionstest wurde die korrekte Funktionsweise der RAM-Switch Steuereinheit überprüft. Diese Steuereinheit überwacht die Steuersignale um ein verbinden von gleichadressierten RAM-Bänken an einen Bus zu verhindern.

Durch den Test mit mehreren gültigen und ungültigen Konfigurationen wurde diese Funktionalität überprüft. Die Überwachung schaltet dabei nur gültige Konfigurationen weiter auf die Steuerleitungen.

Direkter paralleler RAM-Zugriff: In einem zweiten Funktionstest wurde der direkte Zugriff des Host-Rechners auf die einzelnen an den PCI-Interfacebaustein geschalteten RAM-Bänke getestet. Jeweils eine RAM-Banke wurde über den RAM-Switch angeschlossen und mit Daten gefüllt und wieder ausgelesen.

Dieser Test hat gezeigt, daß der RAM-Switch die RAM-Bänke einzeln an den PCI-Interfacebaustein anschließen kann und das die Daten geschrieben und wieder ausgelesen werden können.

Zugriff der FPGA-Bausteine auf die RAMs: Der dritte Test hat den Zugriff des FPGA-Bausteins auf die RAM-Bänke getestet, indem auch hier jeweils Daten in das RAM geschrieben und anschließend wieder ausgelesen wurden.

Auch dieser Test hat die Funktionalität des RAM-Switchs gezeigt. Zusätzlich wurde die Grenzfrequenz gemessen, bei denen die Überprüfung der Daten Fehler anzeigte. Diese Grenzfrequenz beträgt 33MHz und ist zurückzuführen auf eine FPGA-Schaltung die schnell und nicht optimal umgesetzt wurde.

Wie die Funktionstests gezeigt haben erfüllt der RAM-Switch die an ihn gestellten Aufgaben und ermöglicht einen Zugriff von der FPGA-Schaltung aber auch von dem PCI-Interfacebaustein.

Die zweite Messung im Zusammenhang mit den RAM-Switch betrifft die Signalverformung bedingt durch die kapazitive Last der Busstruktur. Diese durchgeführte Messung geht über die in Anhang A beschriebenen Messungen hinaus, da hier zusätzlich zu der kapazitiven Last auch der Einfluß durch das Platinenlayout mit in das Ergebnis eingeht.

Zur Durchführung der Messung wurde die verbunden RAM-Bank von der FPGA-Schaltung beschrieben und das Signal direkt am RAM-Baustein gemessen. Diese Messung garantiert, daß alle Effekte, die durch den RAM-Switch und die Leitungsführung auf der Platine entstehen, in dem aufgenommenen Signal berücksichtigt werden. Mit einem Oszilloskop wurde das Taktsignal und ein Datensignal bei jeweils 14MHz und bei 80MHz aufgenommen. Die beiden Diagramme sind in Abbildung 6.10 und 6.11 dargestellt.

Wie in der Abbildung 6.10 zu sehen ist werden die Signalverläufe bei niedrigen Frequenzen nicht beeinflußt. Die Verzögerung von ca. 10ns des oben abgebildeten Datensignals entsteht durch die FPGA-Schaltung und die Ausgangstreiber des eingesetzten FPGA-Bausteins. Die Auswirkungen der kapazitiven Last sind in der zweiten Abbildung 6.11 deutlich zu sehen. Bedingt durch die Last werden die Signalverläufe geglättet, da der RAM-Switch wie ein Tiefpaß wirkt. Dennoch sind die Signalpegel eindeutig zu erkennen und die Daten können bei Einhaltung des Timings von den entsprechenden Bausteinen eingelesen werden. Messungen zur Ermittlung der Verzögerung durch den RAM-Switch wurden nicht durchgeführt, denn die Verzögerung ist durch die in Anhang A beschriebenen Messungen schon bestimmt.

Diese Messungen und Funktionstest zeigen, daß sowohl die Funktion als auch die Einsatzmöglichkeiten des RAM-Switchs gegeben sind. Die gestellten Anforderungen nach einem direkten Zugriff, einer kurzen Programmierzeit und nach einem flexiblen Einsatz wurden durch die Funktionstests aufgezeigt. Wie die weiteren Messungen der Signalverläufe zeigen ist die Funktionalität bis zu einer Taktrate von 80MHz gegeben[6]

[6] Funktionalität aufgrund der Signalpegel. Zu Beachten ist zusätzlich die Verzögerung durch die FPGA-Schaltung und den FPGA-Baustein.

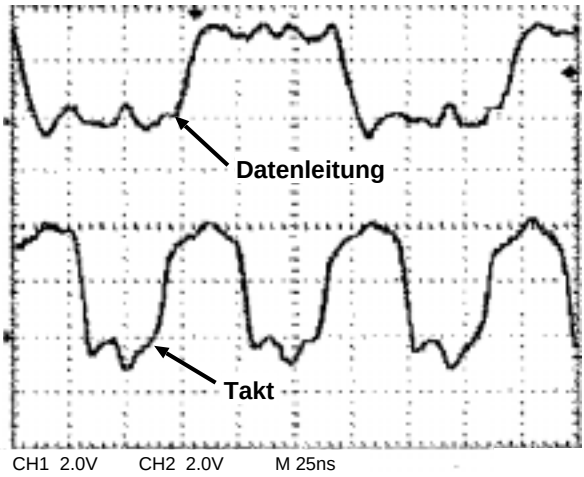


Abbildung 6.10: Signalverlauf des Takt- und eines Datensignals durch den RAM-Switch bei 14MHz.

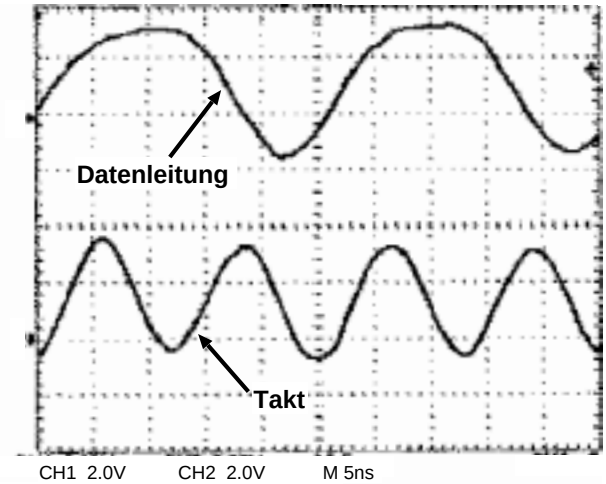


Abbildung 6.11: Signalverlauf des Takt- und eines Datensignals durch den RAM-Switch bei 80MHz.

6.5 Softwareunterstützung

Zusammen mit der Realisierung des neuen FPGA-Koprozessors wird auch die Umsetzung eines neuen Softwarekonzepts ermöglicht. Dieses neue Softwarekonzept, das im Zusammenhang mit den Neuerungen auf dem FPGA-Koprozessor entstanden ist, verfolgt primär das Ziel die Ausführung von mehreren Anwendungen auf einem Prozessor so effektiv wie möglich durchzuführen. Dieses Konzept ist speziell auf den neuen FPGA-Koprozessor abgestimmt und kann diesen daher optimal nutzen und somit die Anwendungsausführung über die eigentliche Datenverarbeitung hinaus beschleunigen. Diese ganzheitliche Sichtweise, die sowohl den FPGA-Koprozessor (Hardware) als auch das FPGA-Betriebssystem (Software) umfasst, wird komplettiert durch eine Entwicklungsumgebung, welche die Erstellung von neuen Anwendungen stark vereinfacht.

Die Entwicklungsumgebung und das Softwarekonzept basieren dabei auf der exklusiven Nutzung des FPGA-Bausteins durch jeweils eine Anwendung, der Ausnutzung des RAM-Switchs und der Umsetzung des speicherbasierten Ausführungsmodells. Eine detailierte Beschreibung dieses Ausführungsmodells und der dazu entstandenen Entwicklungsumgebung ist in den folgenden Abschnitten beschrieben.

6.5.1 Speicherbasiertes Ausführungsmodell

Das speicherbasierte Ausführungsmodell für Anwendungen wird zur Umsetzung der FPGA-Schaltung, aber auch zur Ansteuerung des FPGA-Koprozessors eingesetzt. Ziel dieses Ausführungsmodells ist zum einen die weitere Verkürzung der Ausführungszeit beim Einsatz des FPGA-Betriebssystems und zum anderen eine Vereinfachung der Entwicklung durch die Einführung von standardisierten Schnittstellen zum Datenaustausch zwischen FPGA-Schaltung und Host-Prozessor.

Die jeweiligen Verbesserungen entstehen durch die Ausnutzung der folgenden Möglichkeiten: Das parallele Ausführen von Datentransfers, die effektive Nutzung der verfügbaren Datentransferraten und ein schnellen und direkten Zugriff der FPGA-Schaltung auf die Daten.

Parallele Datentransfers: Der realisierte RAM-Switch ermöglicht es dem FPGA-Betriebssystem während der Ausführung einer Anwendung auf dem FPGA-Koprozessor die nicht genutzten RAM-Bänke über den PCI-Bus auszulesen oder zu beschreiben. Bedingt durch diesen parallelen Zugriff ist das FPGA-Betriebssystem in der Lage den notwendigen Datentransfer für die nächste auszuführende Anwendung schon im Vorfeld durchzuführen. Somit kann nach dem nächsten Prozesswechsel die Verarbeitung der Daten sofort beginnen ohne auf die Beendigung des Datentransfers zu warten. Eine detailierte Funktionsbeschreibung wird nachfolgend näher erläutert.

Dieser vorab durchgeführte Datentransfer vermindert somit die Verarbeitungszeit auf dem FPGA-Baustein und somit auch die insgesamt benötigte Zeit zur Ausführung der Anwendungen.

Effektive Datentransfers: Das speicherbasierte Ausführungsmodell ist so ausgelegt, daß der Inhalt einer RAM-Bank immer komplett übertragen wird, denn wie die Messungen der PCI-Datenübertragung in Abschnitt 5.6.2 gezeigt haben ist nur bei größeren Datenblöcken eine hohen Datenrate erreichbar. Darüberhinaus ist die Datenausgangsrate des PCI-Interfacebausteins durch die schnellen RAM-Bänke stets höher als die Dateneingangsrate von dem PCI-Bus. Dies garantiert einen optimalen und sehr effektiven Datentransfer aufgrund der langen Bursts, die über den PCI-Bus übertragen werden können.

Schneller Datenzugriff: Aus Sicht der FPGA-Schaltungen entsteht durch das speicherbasierte Ausführungsmodell eine standardisierte Schnittstelle die einen schnellen, direkten und von der Schaltung kontrollierbaren Zugriff auf die zu verarbeitenden Daten bereitstellt. Aufgrund der Realisierung des RAM-Switchs und der dazu durchgeführten Messungen ist ein Auslesen der Daten mit bis zu 400MByte/s[7] möglich. Diese hohe Datenrate ermöglicht es vor allem Datenflußanwendungen, die durch den Datentransfer über den

[7] Auslesen von nur einer RAM-Bank mit einer Taktrate von 100MHz. Die gesamte RAM-Zugriffsrate beträgt 1.6GByte/s

PCI-Bus beschrängt sind, die Daten mit einer schnelleren Taktrate zu verarbeiten und somit ihre Ausführungszeit weiter zu vermindern.

Die Funktionsweise des parallelen Zugriffs ist in der Abbildung 6.12 dargestellt. Das Beispiel zeigt eine Situation, bei der eine Anwendung

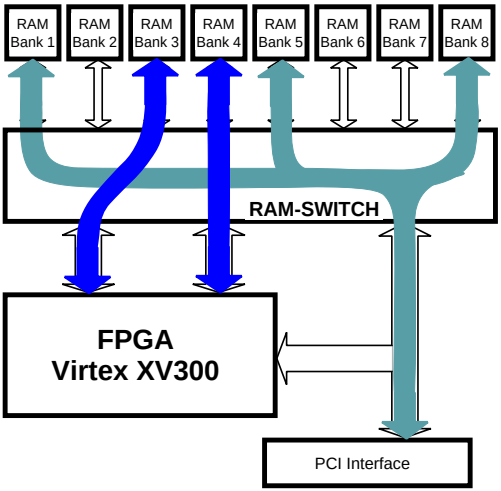


Abbildung 6.12: Beispiel eines parallelen Zugriffs auf die RAM-Bank. *In dem Beispiel arbeitet die FPGA-Schaltung auf dem RAM-Bänken drei und vier, wärend parallel dazu die RAM-Bänke eins, fünf und acht mit Daten gefüllt werden.*

auf den FPGA-Koprozessor ausgeführt wird und parallel dazu weitere RAM-Bänke vorbereitet werden. Die dort ausgeführte Anwendung verwendet jeweils eine RAM-Bank (Ram-Bank 3 und 4) an jeder Schnittstelle. Parallel dazu ist das FPGA-Betriebssystem über den PCI-Bus und den PCI-Interfacebaustein mit einem Teil der anderen RAM-Bänke (Ram-Bank 1, 5 und 8) verbunden. Diese Verbindungen dienen dazu, die Ergebnisdaten der vorherigen Anwendung aus der RAM-Bank 1 auszulesen und jeweils einen neuen Datensatz für die nächste auszuführende

Anwendung in der RAM-Bank 5 und 8 vorzubereiten.

Das nachfolgende Timingdiagramm in Abbildung 6.13 zeigt den dazugehörigen zeitlichen Ablauf, der durch das FPGA-Betriebssystem bestimmt wird. In dem Beispiel ist zu sehen, daß die erste Anwendung die

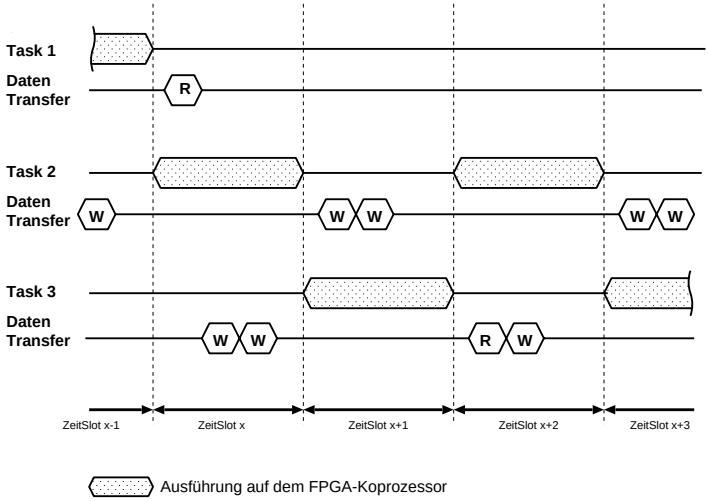


Abbildung 6.13: Timingdiagramm, das die parallelen Zugriffe wärend der Ausführung einer Anwendung zeigt.

Ausführung auf dem FPGA-Baustein beendet. Die letzten Ergebnisdaten werden aber erst gelesen, sobald die nächste Anwendung auf dem FPGA-Koprozessor ausgeführt wird. Zusätzlich werden parallel zu der Ausführung der zweiten Anwendung auch die Daten für die dritte Anwendung übertragen, so daß die dritte Anwendung diese sofort nach der Konfiguration des FPGA-Bausteins verarbeiten kann.

Die Vorteile, die sich durch diese parallele Datenübertragung ergeben, sind: eine bessere Auslastung des FPGA-Koprozessors, ein höherer Durchsatz und eine kürzere Ausführungszeit.

Durch die direkte Verbindung zwischen dem FPGA-Betriebssystem und den nicht genutzen RAM-Bänken kann die Übertragung der Daten

schon vor der eigentlichen Ausführung der Anwendung erfolgen. Dieses Vorgehen wirkt sich direkt auf die Auslastung des FPGA-Bausteins aus, denn der FPGA-Baustein kann während der gesamten Zeitscheibe Daten verarbeiten ohne zusätzlich noch Daten übertragen zu müssen. Ausgehend von einer 500ms langen Zeitscheibe, in der 4MByte Daten mit 120MByte/s übertragen werden, ergibt sich somit eine Verbesserung der Auslastung um 7%. Bei Datenflußanwendungen, wie z.B. der DTP Anwendung, ist diese Verbesserung noch deutlich höher, denn dort wird ca. die Hälfte der Ausführungszeit für die Übertragung der Daten benötigt.

Zusammen mit der Prozessorauslastung steigt auch der Durchsatz des Systems, da mehr Anwendungen innerhalb einer definierten Zeit beendet werden. Der höhere Durchsatz ist ebenfalls auf die parallele Vor- und Nachbereitung der Anwendungen zurückzuführen.

Die kürzeren Ausführungszeiten ergeben sich aufgrund der parallelen Zugriffsmöglichkeit auf die nicht verwendeten RAM-Bänke. Dies wird deutlich bei Betrachtung der ersten Anwendung, die in der Abbildung 6.13 dargestellt ist. Nach der Beendigung der Zeitscheibe stehen die Ergebnisdaten zum Auslesen bereit. In dem Beispiel werden diese Daten innerhalb der nächsten Zeitscheibe ausgelesen und die Anwendung kann beendet werden. Besteht aber kein paralleler Zugriff auf die Daten kann die Anwendung erst beendet werden, sobald sie vom FPGA-Betriebssystem erneut ausgeführt wird und ein Auslesen der Daten ermöglicht.

6.5.2 Entwicklungsumgebung

Die Entwicklungsumgebung, die zusammen mit der Hardware und dem dazugehörigen Softwarekonzept entwickelt wurde, komplettiert das gesamte System und ermöglicht eine einfache und schnelle Anwendungsentwicklung.

Aufgrund des eingesetzten speicherbasierten Ausführungsmodells sind alle Anwendungen über die gleiche standardisierte Schnittstelle mit den zu verarbeitenden Daten verbunden. Diese Schnittstelle entspricht den RAM-Zugriffen über den RAM-Switch, denn alle zu verarbeitenden Daten sind in den RAM-Bänken abgelegt. Zusätzlich zu dieser Schnittstelle

verfügt der FPGA-Baustein und somit auch jede FPGA-Schaltung über eine zusätzliche Schnittstelle zu dem externen Stecker und zu dem PCI-Interfacebaustein. Diese drei Schnittstellen sind integraler Bestandteil jeder FPGA-Schaltung und erfüllen spezifische Aufgaben. Die RAM-Schnittstelle wird eingesetzt, um große Datenmengen zwischen der Anwendung und dem FPGA-Betriebssystem effektiv auszutauschen. Die Schnittstelle zu dem PCI-Interfacebaustein dient zum Steuern der FPGA-Schaltung durch einzelne Registerzugriffe und über den externen Stecker werden weitere Hardwarekomponenten angeschlossen.

Die Aufteilung der Funktionen auf die drei Schnittstellen ermöglicht die Realisierung aller in Kapitel 4 beschriebenen Anwendungen auf dem neuen FPGA-Koprozessor. Die entwickelte Simulationsumgebung, die in der Abbildung 6.14 illustriert ist, stellt dem Schaltungsentwickler eine Testbench zur Verfügung. Diese Testbench verfügt über die oben genann-

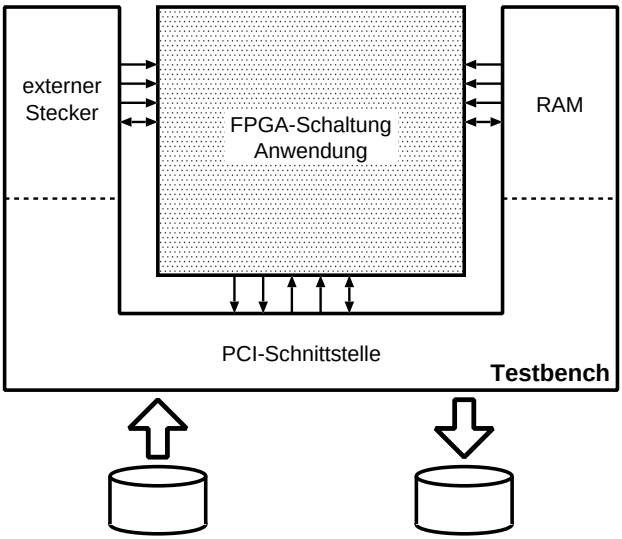


Abbildung 6.14: Blockschaltbild der Simulationsumgebung, die zur Entwicklung neuer Anwendungen eingesetzt wird.

ten drei Schnittstellen und ermöglicht eine automatisierte Simulation der gesamten FPGA-Schaltung. Die Aufgabe der umgebenden Testbench ist die Simulation der Datenzugriffe mit den entsprechenden Protokollen. Durch den Zugriff auf Dateien ist der Entwickler zusätzlich in der Lage, große Datenmengen oder komplette Anwendungen vollständig und sehr einfach zu simulieren. Eine weitere Vereinfachung dieser Simulationsumgebung ist die Zusammenstellung der Daten. Die Funktionen in der API-Schnittstelle stehen den Entwickler auch für die Generierung der Simulationsdaten zur Verfügung so das mit einem Host-Programm sowohl die Simulationsdaten als auch die spätere FPGA-Schaltung angesteuert werden kann.

Diese Entwicklungsumgebung bietet dem Anwendungsentwickler somit eine sehr einfache und vor allem schnelle Möglichkeit, neue Anwendungen zu erstellen und die die Turnaround Zeiten bei der Anwendungsentwicklung zu verkürzen.

6.6 Ergebnis

Der beschriebene neue FPGA-Koprozessor ist in der Lage, die Konfiguration und den Readback des eingesetzten FPGA-Bausteins in einer minimalen Zeit auszuführen. Ermöglicht wird dies durch den Einsatz eines speziellen FIFOs, der optimalen Anbindung des PCI-Interfacebausteins an die Konfigurationsschnittstelle und durch die Softwareoptimierungen. Die gemessene Zeit für die Konfiguration beträgt 6,2ms und liegt damit nur ca. 12% über der theoretischen Übertragungszeit für die 214KByte Konfigurationsdaten. Die optimale Realisierung der Anbindung macht sich auch bei dem Readback bemerkbar, der in nur 9,1ms ausgeführt wird. Insgesamt ergibt sich somit eine Prozesswechselzeit von weniger als 20ms. Im Vergleich zu den Messungen des Client-Server Betriebssystems in Abschnitt 5.6 erreicht der neue FPGA-Koprozessor eine um den Faktor 4 schnellere Prozesswechselzeit, bei einem $\approx$30% kleineren Konfigurationsbitstrom. Die verringerte Prozesswechselzeit reduziert auch den systematischen Overhead des gesamten FPGA-Betriebssystems und die damit verbundenen Bewertungskriterien. Die Ausführungs- und Wartezeiten verringern sich und der Durchsatz aber auch die Prozessorauslastung werden durch die kürzere Prozesswechselzeit verbessert.

Ein weiteres Ergebnis, das durch die Neuerungen auf dem FPGA-Koprozessor erreicht wird, ist die Integration der, auf dem FPGA-Koprozessor verfügbaren, RAM-Bänke in den Prozesswechsel. Die Sicherung und Rekonstruktion des lokalen RAMs auf dem FPGA-Koprozessor wurde bei dem in Kapitel 5 implementierten Client-Server Betriebssystem nicht umgesetzt, da es mit bis zu 83ms die Prozesswechselzeit verdoppelt hätte. Durch die Realisierung des RAM-Switchs auf dem neuen FPGA-Koprozessor ist die Sicherung sowie die Rekonstruktion der Daten in den RAMs nicht mehr notwendig, da der gesamte RAM-Baustein inklusive des derzeitigen RAM-Inhalts inaktiv geschaltet wird. Der Zugriff durch den RAM-Switch auf die RAM-Bänke erfolgt direkt mit einer maximalen Taktrate von bis zu 80MHz. Zudem kann der Switch innerhalb von 10μs umgeschaltet werden und ermöglicht einen parallelen Zugriff des PCI-Interfacebausteins auf die RAM-Bänke, der für das neu Ausführungsmodell benötigt wird. Bedingt durch den Aufbau des RAM-Switchs entsteht eine kapazitiven Last für die Signale, die einem Tiefpaßverhalten entspricht. Wie Messungen gezeigt haben sind die Auswirkungen auf einzelne Signale nur gering und bei Messungen bis 80MHz bleiben die Signalpegel und damit die Signale eindeutig.

Die optimale Anbindung der Konfigurationsschnittstelle, die Umsetzung der Taktgenerierung und die Realisierung des RAM-Switchs stellen die spezielle Unterstützung für das preemptive Multitasking bereit, die für die erreichte kurze Prozesswechselzeit verantwortlich sind. Zusätzlich ermöglicht der RAM-Switch ein für FPGA-Koprozessoren neues Ausführungsmodell, das den Datenaustausch über die RAM-Bänke ausführt und somit die Prozessorauslastung um weitere 7% verbessern kann.

6.7 Zusammenfassung

Dieses Kapitel beschreibt die Realisierung eines neuen FPGA-Koprozessors, der die Ausführung des preemptiven Multitaskings im besonderen Maße unterstützt. Die Verbesserungen gegenüber den anderen FPGA-Koprozessoren ergeben sich aus den Einschränkungen beim Betrieb des Client-Server Betriebssystems auf dem FPGA-Koprozessor microEna-

bleII und aus den Anforderungen, die für den Betrieb eines preemptiven Multitaskings gefordert werden. Zusammengefasst werden die einzelnen Punkte in dem Gesamtkonzept des neuen FPGA-Koprozessors. Dieses Konzept ist unterteilt in die Hardwarekonzepte, die die Verbesserungen beim Aufbau des FPGA-Koprozessors zusammenfassen und in die Softwarekonzepte, die eine effektive Nutzung des FPGA-Koprozessors ermöglichen.

Ziel der Hardwarekonzepte ist es, die Prozesswechselzeiten durch geeignete Anbindungen zu minimieren und die Anforderungen an den FPGA-Koprozessor zu erfüllen. Dazu zählt die Takterzeugung und Taktabschaltung, eine optimale Anbindung des FPGA-Betriebssystems an die Konfigurationsschnittstelle und die Einführung eines RAM-Switchs. Die einzelnen Verbesserungen werden in Detail in ihrer Funktion beschrieben und die zur Auswahl stehenden Realisierungsmöglichkeiten werden besprochen und bewertet.

Die Anbindung an die Konfigurationsschnittstelle beeinflußt direkt die benötigte Zeit für einen Prozesswechsel. Zur Minimierung der Prozesswechselzeit ist es somit notwendig, die Anbindung aber auch die Softwareansteuerung für die Konfiguration und den Readback zu optimieren. Insgesamt werden dazu vier Punkte analysiert, die einen Einfluß auf die Konfigurationszeiten besitzen. Zur abschließenden Bewertung der Verbesserung werden Zeitmessungen für die Konfiguration und den Readback durchgeführt. Es zeigt sich, daß die Konfiguration 6,2ms benötigt und somit nur 12% langsamer ist als die theoretisch errechnete Zeit. Ein Readback benötigt 9,1ms, so daß sich insgesamt eine Prozesswechselzeit unter 20ms ergibt. Im Vergleich zu dem FPGA-Koprozessor microEnableII ist das eine Verbesserung um den Faktor 4.

Die weitere Neuerung ist realisiert durch den RAM-Switch, der es ermöglicht einzelne RAM-Bänke dynamisch an die FPGA-Bausteine oder den PCI-Interfacebaustein anzuschließen. Der Aufbau und die zum Einsatz kommenden Komponenten werden im Detail beschrieben und bewertet. Zum Aufbau des RAM-Switchs werden sogenannte Quick-Switch Bausteine eingesetzt, da nur sie die geforderten Anforderungen erfüllen. Die durchgeführten Messungen an dem RAM-Switch zeigen, daß die geforderte Funktionalität bereitgestellt wird. Weitere Messungen der Si-

gnalverläufe zeigen, daß das Tiefpaßverhalten des RAM-Switchs eine Einfluß auf die Signalverläufe hat, aber die Signalpegel erhalten bleiben. Diese Messungen wurde bis zu einer Taktfrequenz von 80MHz durchgeführt.
Die drei Softwarekonzepte, die eine effektive Nutzung des neuen FPGA-Koprozessor ermöglichen, werden ebenfalls beschrieben und in ihrer Funktionsweise erläutert. Diese Konzepte bestimmen die Nutzung des FPGA-Bausteins und die damit verbundene Entwicklungsumgebung, die die Anwendungsentwicklung vereinfacht. Das weiterführende Ausführungsmodell beschreibt ein Konzept, das unter Ausnutzung der Möglichkeiten, die durch den RAM-Switch entstehen, eine weitere Verbesserung der Ausführungszeiten erreicht.

Kapitel 7

Diskussion und Ausblick

Gegenstand dieser Arbeit ist der Entwurf und die Realisierung eines Multitasking Betriebssystems, das die Ausführung von mehreren Anwendungen auf einem FPGA-Koprozessor steuert.

Die Ausführung von Anwendungen auf dem FPGA-Koprozessor beschleunigt die Datenverarbeitung und führt somit schneller zu einem Ergebnis. Zum Einsatz kommen FPGA-Bausteine, die sich durch ihre umfangreiche Programmierbarkeit, ihre hohe Verarbeitungsgeschwindigkeit aber auch durch eine dynamische Rekonfigurierbarkeit auszeichnen. Zur Beschleunigung der Ausführung werden die einzelnen Operationen parallel ausgeführt und die schnelle Verarbeitung in jedem Takt ausgenutzt. Auf diese Weise sind Beschleunigungsfaktoren zwischen 10 und 1000 erreichbar. Die Daten der Anwendung werden auf dem FPGA-Koprozessor verarbeitet und durch ein Host-Programm gesteuert.

Betriebssysteme, die die Ausführung von Anwendungen auf dem Betriebsmittel FPGA-Koprozessor steuern, verfügen nur über eine unzureichende Unterstützung der Betriebsmittel und unterstützen meist nur einzelne FPGA-Koprozessoren. Aufgrund der mangelnden Unterstützung dieser Betriebssysteme erfolgt die Steuerung der einzelnen Anwendungen bei den meisten FPGA-Koprozessoren jeweils direkt aus der einzelnen Anwendung heraus. Diese direkte Kontrolle hat zur Folge, daß eine Beschleunigung durch die Verarbeitung auf dem FPGA-Baustein immer nur einer Anwendung vorbehalten ist. Das neue Multitasking Betriebs-

system hebt diesen Nachteil auf und ermöglicht durch den preemptiven Ansatz die parallele Verarbeitung von mehreren unabhängigen Anwendungen auf einem FPGA-Koprozessor. Zusätzlich wird das realisierte Multitasking Betriebssystem unterstützt durch den Aufbau eines neuen FPGA-Koprozessors der die schnellen Prozesswechsel, die für den preemptiven Multitaskingansatz notwendig sind, im besonderen Maße unterstützt.

Preemptives Multitasking Betriebssystem

Das implementierte Multitasking Betriebssystem für FPGA-Koprozessoren basiert auf einer Client-Server Architektur und eröffnet die Nutzung des FPGA-Koprozessors durch mehrere unabhängige Anwendungen zur gleichen Zeit.

Die parallele Ausführung der einzelnen Anwendungen erfolgt, wie bei modernen Multitasking Betriebssystemen üblich, durch ein zeitliches Unterteilen der zur Verfügung stehenden Rechenzeit auf dem FPGA-Koprozessor. Das realisierte Betriebssystem arbeitet im Vergleich zu den existierenden Betriebssystemen für FPGA-Koprozessoren preemptive, d.h. die Ausführung auf dem FPGA-Bausteins wird durch das Betriebssystem unterbrochen und zu einem späteren Zeitpunkt weiter fortgesetzt. Der preemptive Ansatz bedeutet für das Betriebssystem, daß bei jedem Prozesswechsel alle Register und RAM-Inhalte des zum Einsatz kommenden FPGA-Bausteins gesichert werden müssen und das dieser gesicherte Zustand bei einer erneuten Ausführung wieder hergestellt werden muß.

Basis dieser Zustandsrekonstruktion ist die Möglichkeit, den Zustand jedes Registers und jeder RAM-Zelle individuell bestimmen und ermitteln zu können. Sowohl für die Zustandsermittlung als auch für die Zustandsrekonstruktion ist der Aufbau des jeweiligen Datenstroms notwendig, der die Zustandsinformationen enthält. Verbunden mit dem nicht bekannten internen Aufbau der FPGA-Bausteine ist auch dieser Datenstrom in den meisten Fällen nicht öffentlich. Diese Tatsache erschwert die Portierung des Betriebssystems, da zur Unterstützung eines FPGA-Koprozessors mit einem anderen FPGA-Baustein entweder die Architektur der Datenströme bekannt sein muß, oder die Zustandsbits in einem aufwendigen Verfahren einzeln ermittelt werden müssen.

Abgesehen von der Unterstützung weiterer FPGA-Bausteinfamilien ist bei der Realisierung des Multitasking Betriebssystems auf eine einfache Portierung geachtet worden. Mit dem Aufbau der Client-Server Architektur und der Verwendung der standardisierten Interprozess Kommunikation kann das Betriebssystem einfach auf andere Host-Betriebssysteme portiert werden. Zusätzlich wird durch den modularen Aufbau auch eine Unterstützung weiterer FPGA-Koprozessoren möglich. Der Nachteil, der durch die Client-Server Architektur und durch den modularen Aufbau entsteht, ist ein zusätzlicher Overhead bei der Kommunikation zwischen Client und Server. Dieser Overhead verlängert die Ausführung aller Kommandos um 60μs. Unter Berücksichtigung der wenigen notwendigen Kommandos zur Steuerung der Verarbeitung kann dieser Overhead jedoch vernachlässigt werden.

Bedingt durch die Portierung des Betriebssystems zur Unterstützung mehrerer FPGA-Koprozessoren entsteht erstmals auch eine einheitliche Programmierschnittstelle, die zur Steuerung der unterschiedlichen FPGA-Koprozessoren eingesetzt werden kann und die Portierung von ganzen Anwendungen vereinfacht.

Die Unterteilung der Rechenzeit auf dem FPGA-Koprozessor wird durch ein modifiziertes Round-Robin Schedulingverfahren gewährleistet, das unter Berücksichtigung einer dynamischen Priorität mehrere Zeitscheiben zusammenfügen kann. Die Ausführung über mehrere Zeitscheiben hinweg wurde gewählt, da die Prozesswechsel auf dem eingesetzten FPGA-Koprozessor microEnable II ca. 16% der gesamten Zeitscheibe beansprucht. Die Länge der Zeitscheiben beträgt 500ms und orientiert sich an der durchschnittlichen Ausführungszeit der Anwendungen die auf den FPGA-Koprozessoren ausgeführt werden. Sie ist so gewählt, daß Anwendungen mit einer kleinen Ausführungszeit innerhalb einer Zeitscheibe ausgeführt werden und das Anwendungen mit langen Ausführungszeiten durch den Overhead nicht unnötig verlängert werden, aber gleichzeitig die Wartezeiten der weiteren Anwendungen akzeptabel bleiben.

Eine für die untersuchten Anwendungen entscheidende Einschränkung, die durch den Einsatz des FPGA-Koprozessor microEnable II entsteht, ist die nicht durchgeführte Sicherung und Rekonstruktion des

Inhalts, der auf dem FPGA-Koprozessor verfügbaren RAM-Bausteine. 12 der 15 untersuchten Anwendungen verwenden das verfügbare RAM, das an den FPGA-Baustein angeschlossen ist. Bedingt durch den häufigen Einsatz dieses RAMs ist es integraler Bestandteil der Anwendung und muß bei einem Prozesswechsel mit gesichert werden. Die Sicherung und anschließende Rekonstruktion des RAM-Inhalts benötigt auf dem verwendeten FPGA-Koprozessor bis zu 83ms und würde somit die Prozesswechselzeit verdoppeln. Aus diesem Grund wird bei dem umgesetzten Client-Server Betriebssystem die Sicherung des RAM-Inhalts nicht durchgeführt und schränkt daher die auszuführenden Anwendungen stark ein.

Die durchgeführten Messungen der Prozesswechselzeiten, der Ausführungszeiten und der Zeitscheibenlängen haben gezeigt, daß das Multitasking Betriebssystem in der Lage ist, mehrere unabhängige Anwendungen gleichzeitig ausführen zu können. Eine detaillierte Analyse der zu verarbeitenden Anwendungen, des zum Einsatz kommenden FPGA-Koprozessors und des Ablaufs bei der Ausführung der Anwendungen deckt mehrere Verbesserungspotentiale auf. Das Ziel der Verbesserungen ist die Reduzierung der Prozesswechselzeiten, die sich direkt auf den systematischen Overhead und damit auf den Leistungsverlust bedingt durch das Betriebssystem auswirken. Umgesetzt sind diese Verbesserungen in einem neuen FPGA-Koprozessor der basierend auf dieser Analyse und den Erfahrungen das preemptive Multitasking Betriebssystem im besonderen Maße unterstützt.

Multitasking FPGA-Koprozessor

Dieser neue FPGA-Koprozessor ist speziell für den Einsatz des preemptiven Multitasking Betriebssystems konzipiert worden und hat die Minimierung der Prozesswechselzeit zum Ziel. Aufgrund der zuvor durchgeführten Analyse und den Erfahrungen bei der Anwendungsentwicklung wurden drei Bereiche ausgewählt, die das größte Verbesserungspotential aufweisen. Diese drei Bereiche sind: die Takterzeugung, die Anbindung der Konfigurationsschnittstelle und die fehlende Sicherung des RAM-Inhalts.

Im Bereich der Takterzeugung besteht durch den Einsatz des FPGA-Koprozessors microEnable II eine Einschränkung, die es nicht ermöglicht den Takt gesteuert und präzise anzuhalten, um den Zustand der FPGA-Schaltung zu sichern. Durch die Umsetzung der, an die Taktgenerierung gestellten Anforderungen wird diese Einschränkung beseitigt. Mit der realisierten Steuerung auf dem FPGA-Koprozessor erfolgt die Kontrolle des Taktes über eine Steuerleitung, die durch das FPGA-Betriebssystem und die FPGA-Schaltung selbst aktiviert wird. Die maximale Frequenz, die präzise abgeschaltet werden kann, beträgt 70MHz und wird durch die zusätzlich benötigte Steuerlogik bestimmt. Durch den Einsatz eines schnelleren Logikbausteins kann diese maximal abschaltbare Frequenz bis auf 100MHz erhöht werden.

Im Gegensatz zu der Realisierung der Takterzeugung wirken sich die Verbesserungen bei der Anbindung der Konfigurationsschnittstelle direkt auf die Prozesswechselzeiten aus. Durch den Einsatz eines kleineren FPGA-Bausteins, der besseren Anbindung über ein FIFO, die Verwendung einer schnellen parallelen Schnittstelle und weiteren Softwareoptimierungen wird die benötigte Prozesswechselzeit in Vergleich zu der zuvor gemessenen Zeit um den Faktor 4 auf weniger als 20ms verringert. Zurückzuführen ist diese Verbesserung maßgeblich auf eine effektivere Ausnutzung des PCI-Busses und auf die Softwareoptimierungen. Diese gemessene Zeit für die Konfiguration, die nur 12% über der theoretischen Datenübertragungszeit liegt, zeigt, daß dort keine weiteren Verbesserungen möglich sind. Anders verhält es sich bei dem Readback, der aufgrund von sechs einzeln nacheinanderfolgenden Übertragungen rund 65% länger benötigt als die theoretische Übertragungszeit.

Die dritte Verbesserung dient zur weiteren Verringerung der Prozesswechselzeiten und zur Umsetzung der Sicherung und der Rekonstruktion der häufig verwendeten RAM-Bausteine. Durch die Einführung eines RAM-Switchs und acht einzelnen RAM-Bänken ist eine Sicherung des Inhalts nicht länger notwendig, denn die RAM-Bänke werden zusammen mit der Anwendung einfach deaktiviert und bei der erneuten Ausführung der Anwendung zusammen mit der FPGA-Schaltung wieder aktiviert, wobei der Inhalt während der inaktiven Zeit nicht verändert wird. Durch diese Vorgehensweise bei einem Prozesswechsel wird die benötigte Zeit

zur Sicherung und Rekonstruktion des RAM-Inhalts auf die Zeit zum Umgeschalten des RAM-Switchs reduziert. Dieses Umschalten wird in nur 10μs ausgeführt. Um entstehende Leistungsverluste durch diesen RAM-Switch zu minimieren wurde er mit sogenannten Quick-Switchs aufgebaut. Durch den Einsatz dieser Quick-Switchs erfolgt der Zugriff auf die RAM-Bänke direkt und es entstehen keine zusätzlichen Waitstates bei den Zugriffen. Darüberhinaus ermöglichen die eingesetzten Bausteine ein Umschalten in nur 10μs und die Signalverformung durch das Tiefpaßverhalten des Switchs hat selbst bei einer gemessenen Frequenz von 80MHz nur eine geringe Auswirkung. Nachteilig wirkt sich der RAM-Switch, aber auch die acht RAM-Bausteine, bei der Realisierung und den Kosten für den FPGA-Koprozessor aus. Mit über 40 Bausteinen benötigt der RAM-Switch die größte Fläche auf der Platine.

Ausblick

Obgleich der neue FPGA-Koprozessor mit seinen spezifischen Verbesserungen die Prozesswechselzeit um einen weiteren Faktor 4 verringern konnte, bestehen weitere Optimierungsmöglichkeiten, die vor allem den Einfluß auf das Host-Betriebssystem minimieren. Zwei interessante Umsetzungen, die den Rahmen dieser Arbeit gesprengt hätten, sind zum einen die automatische Ermittlung und Rekonstruktion des FPGA-Schaltungszustands während der Konfiguration bzw. während des Readbacks und zum anderen die Ausführung des FPGA-Betriebssystems durch einen eigenen Mikroprozessor direkt auf der FPGA-Koprozessorkarte.

Die Idee der automatischen Zustandsrekonstruktion kann, wie in der nachfolgenden Abbildung 7.1 dargestellt, in die Anbindung der Konfigurationsschnittstelle integriert werden. Bei einem Readback werden die jeweiligen Zustände der Register und RAM-Zellen aus dem Bitstrom ermittelt und in einem eigenen Zustandsspeicher abgelegt. Dieser Zustandsspeicher speichert die jeweiligen Zustände aller ausgeführten Anwendungen und kann so bei der nächsten Konfiguration die ermittelten Zustandsdaten direkt in dem Konfigurationsbitstrom, der zu dem FPGA-Baustein übertragen wird, verändern und den Zustand somit rekonstruieren.

Unterstützt wird diese automatische Zustandsrekonstruktion im spe-

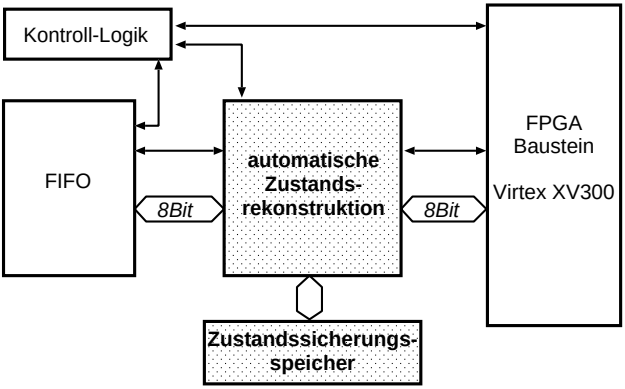


Abbildung 7.1: Integration der automatischen Zustandsrekonstruktion auf dem FPGA-Koprozessor.

ziellen durch die bekannte Bitstromarchitektur des Virtex-Bausteins von Xilinx. Die Zustandsrekonstruktion erfolgt nach dem gleichen Prinzip, das in den Softwarefunktionen in Abschnitt 3.6 realisiert ist. Die Umsetzung dieser automatischen Zustandsrekonstruktion hat durch eine weitere Registerstufe eine zu vernachlässigende negative Auswirkung auf die gesamte Prozesswechselzeit, aber durch die automatisch ausgeführte Zustandsrekonstruktion muß dieser Schritt, der ≈5ms benötigt, nicht mehr im FPGA-Betriebssystems ausgeführt werden und verringert somit die Auswirkungen auf die übrigen, von dem Host-Prozessor ausgeführten, Anwendungen.

Die zweite Erweiterungsmöglichkeit verringert ebenfalls den Einfluß auf das Host-Betriebssystem und die dort auszuführenden Anwendungen. Durch die Verarbeitung des FPGA-Betriebssystems auf einem eigenen Mikroprozessor, der auf dem FPGA-Koprozessor integriert ist, wird der Einfluß auf ein Minimum reduziert. Dieser Mikroprozessor übernimmt alle Aufgaben die zur Durchführung der Prozesswechel und der Anwendungen notwendig sind und arbeitet unabhängig von den Host-Prozessor.

Die gesamte Steuerung der Anwendung wird zusammen mit dem Konfigurationsbitstrom und den zu verarbeitenden Daten an den lokalen Mikroprozessor übermittelt und dort in den zugewiesenen Zeitscheiben verarbeitet. Nach Beendigung der Anwendung werden die Ergebnisdaten zurück an den Host-Prozessor übertragen und dort weiterverarbeitet oder abgespeichert.

Die Integration dieses lokalen Mikroprozessors kann sehr einfach erfolgen, denn der auf dem neuen FPGA-Koprozessor eingesetzte PCI-Interfacebaustein von der Firma PLX verfügt über einen integrierten Mikroprozessor. Dieser eignet sich sehr gut für die Ausführung des FPGA-Betriebssystems, da er direkt Daten mit dem Host-Rechner austauschen kann und die Kontrolle über die Verbindungen zu der Konfigurationsschnittstelle aber auch zu der Steuereinheit und dem RAM-Switch besitzt.

Durch eine Weiterentwicklung der Place&Route-Werkzeuge, die von den FPGA-Herstellern entwickelt werden, und einer einsetzbaren partiellen Nutzung der Logikzellen wird auch die Weiterentwicklung für eine echte parallele Verarbeitung von Anwendungen durch das FPGA-Betriebssystem interessant. Diese Entwicklung ist jedoch derzeit nicht abzusehen, so daß das realisierte FPGA-Betriebssystem, das während einer Zeitscheibe nur eine Anwendung ausführt, die beste einsatzfähige Möglichkeit ist, um die FPGA-Schaltung auszuführen.

Kapitel 8

Zusammenfassung

In der vorliegenden Arbeit wurde die Realisierung eines preemptiven Betriebssystems für FPGA-Koprozessoren vorgestellt, das die Ausführung von mehreren unabhängigen Anwendungen auf dem FPGA-Koprozessor ermöglicht. Unterteilt ist die Arbeit in die Realisierung des Betriebssystems und in den Aufbau eines neuen FPGA-Koprozessors, der das Betriebssystem im Besonderen unterstützt.

Das realisierte Betriebssystem ermöglicht erstmals die gleichzeitige Ausführung von mehreren Anwendungen, die den FPGA-Koprozessor zur Beschleunigung einsetzen. Im Gegensatz zu den bestehenden Betriebssystemen für FPGA-Koprozessor arbeitet dieses neue Betriebssystem preemptiv und kann die Anwendungen auf den FPGA-Baustein in deren Ausführung unterbrechen und zu einem späteren Zeitpunkt weiter fortsetzen. Durch diese preemptive Ausführung entstehen die gleichen Vorteile, die von den modernen Multitasking Betriebssystem wie z.B. LINUX und Windows NT bekannt sind. Der Aufbau des Betriebssystems erfolgt mit einer Client-Server Architektur, um die Portabilität zu unterstützen. Die gewählte Realisierung orientiert sich dabei an einer Anwendungsanalyse und ermöglicht die Unterstützung mehrerer FPGA-Prozessoren. Ausgeführt wird das Betriebssystem auf einem FPGA-Koprozessor, der aufgrund seines allgemeinen Aufbaus nicht al-

le gestellten Anforderungen erfüllt. Dennoch können mehrere Anwendungen gleichzeitig, unterteilt in einzelne Zeitscheiben, auf den FPGA-Baustein ausgeführt werden, wobei eine Prozesswechselzeiten von 80ms erreicht wird. Daraus ergibt sich bedingt durch das Betriebssystem eine Verlängerung der Ausführungszeiten um maximal 20%.

Zur weiteren Verringerung des Einflusses auf die Anwendungen werden die Aufgaben des Betriebssystems analysiert. Basierend auf den Ergebnissen dieser Analyse und den Erfahrungen wird ein neuer FPGA-Koprozessor konzipiert, der das Betriebssystem im besonderen Maße unterstützt. Dieser neue Koprozessor reduziert durch eine bessere Anbindung der Konfigurationsschnittstelle die benötigte Prozesswechselzeit um einen Faktor 4 auf unter 20ms. Zudem erfolgt aufgrund der Realisierung eines RAM-Switchs ein Teil der Zustandssicherung direkt auf dem Koprozessor. Das zugrundeliegende Gesamtkonzept des neuen FPGA-Koprozessors reduziert somit die benötigte Prozesswechselzeit und ermöglicht die Durchführung eines neuen speicherbasierten Ausführungsmodells, welches eine zusätzlich Verbesserung der Ausführungszeiten erlaubt.

Anhang A

Evaluierung des RAM-Switchs

Die für die Realisierung des RAM-Switch eingesetzten Quick-Switch Bausteine wurde zuvor durch eine durchgeführte Messung auf ihre Eignung überprüft. Ziel der durchgeführten Messung ist die experimentelle Untersuchung der Signalverformung, bei der durch die Busstruktur entstehenden kapazitiven Last. Darüberhinaus wird der Meßaufbau auch zur Verifikation der RAM-Switch Eigenschaften eingesetzt.

Der gesamte Meßaufbau, der in Abbildung A.1 graphisch dargestellt ist, entspricht der Verschaltung eines Bussignals in dem realisierten Bus. Die eingesetzten Quick-Switches von der Firma Pericom (PI3B34X245) verfügen über 32 schaltbare Verbindungen, die durch ein Steuersignal verbunden bzw. getrennt werden. Zur Durchführung der Messungen wurden diese Bausteine verwendet, da die 3-zu-1 Multiplexer zum Zeitpunkt der Messung nicht verfügbar waren. Der eingsetzte Pericom Baustein verfügt über die gleichen elektrischen Kenndaten wie der 3-zu-1 Multiplexer und kann daher für die Messungen verwendet werden.

Als Signalquelle wurde ein Signalgenerator eingesetzt. Das Signal dieses Generators ist entsprechend verbunden mit acht Eingängen verteilt auf zwei Quick-Switch Bausteine. Der erste Baustein, an dem nur ein Ein-

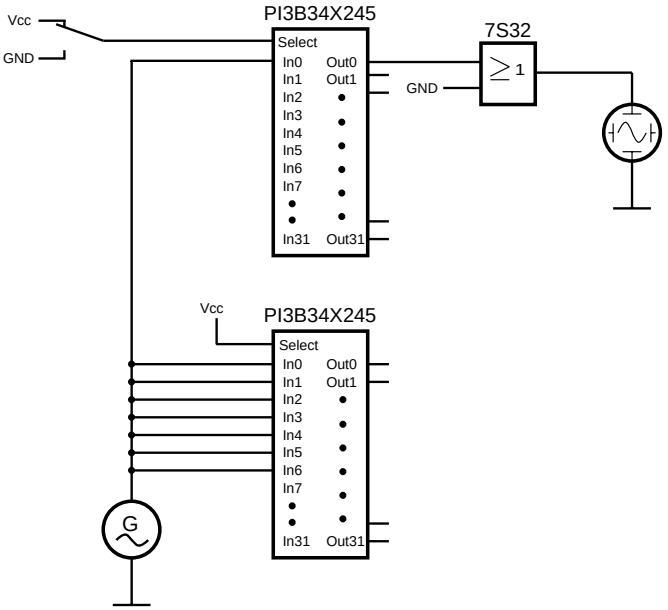


Abbildung A.1: Aufbau der Schaltung, die zur Evaluierung der Quick-Switch Bausteine und der Eigenschaften des RAM-Switchs eingesetzt wird.

gang verwendet wurde, entspricht dem an- bzw. abgeschalteten RAM-Baustein und leitet das Signal im durchgeschalteten Zustand weiter. Bei dem zweiten Baustein wurden sieben Eingänge belegt, die entsprechend der RAM-Switch Verschaltung die kapazitive Last bilden. Daher sind die Verbindungen dieses zweiten Bausteins permanent inaktiv geschaltet. Zur Realisierung des Eingangs eines RAM-Bausteins wurde ein weiteres ODER-Gatter (7S23) an den Ausgang des ersten Quick-Switch Bausteins angehängt, das gleichzeitig zur Signalerkennung eingesetzt wird.

Wie die nachfolgenden Meßkurven in Abbildung A.2 und A.4 zeigen, entsteht durch den RAM-Switch und die eingesetzten Quick-Switch

Bausteine nur eine minimale Verzögerung der Signale von 1-2ns. Desweiteren ist zu erkennen, daß sich die Signalverformung des Eingangssignals durch die entstehende kapazitive Last der abgeschalteten Bausteine nur minimal auswirkt. Durch den zusätzlichen Widerstand (5Ω), der durch die geschaltete Quick-Switch Verbindung entsteht, wird das Signal am Ausgang des Quick-Switchs zusätzlich bedämpft und geglättet. Bedingt durch die minimale Signalveränderungen und durch die zusätzliche Bedämpfung, ist das Signal von dem nachgeschalteten ODER-Baustein ohne weitere Probleme zu erkennen. Siehe dazu auch Abbildung A.4.

In den Abbildungen A.3 und A.5 ist das Ausgangssignal des RAM-Switchs zu sehen, das von dem Eingangssignal abgetrennt ist. Sowohl bei 50MHz als auch bei 100MHz ist am Ausgang nur ein stabiles LOW Signal mit einem geringen Rauschen zu beobachten.

Die gemessenen Signalverläufe, die sich durch den Einsatz des exemplarisch aufgebauten RAM-Switch ergeben, zeigen, daß die einzelnen RAM-Bänke durch den RAM-Switch sicher abgetrennt werden können, und daß auch bei abgeschalteten Verbindungen keine Veränderung der RAM Inhalte zu erwarten ist. Dieses Verhalten ist aufgrund der gleichen elektrischen Kenndaten auch bei den später eingesetzten 3-zu-1 Multiplexern zu erwarten.

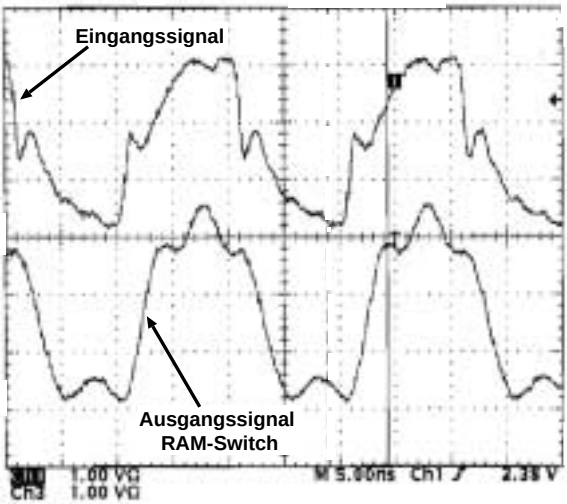


Abbildung A.2: Signalverläufe bei 50MHz und einer angesteuerten und durchgeschalteter Verbindung.

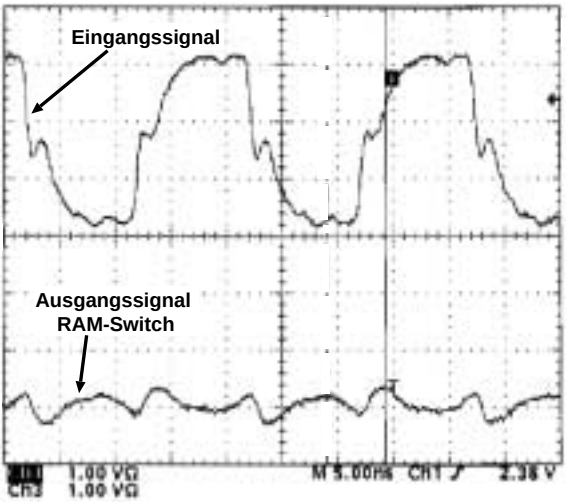


Abbildung A.3: Signalverläufe bei einer getrennten Verbindung eines 50MHz Signals.

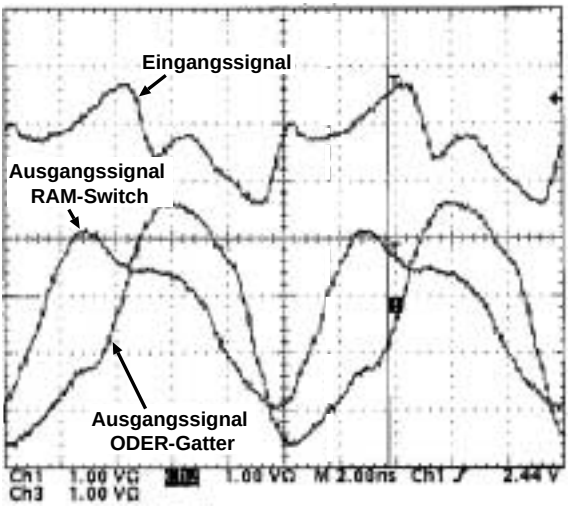


Abbildung A.4: Signalverläufe bei 100MHz und einer angesteuerten und durchgeschalteter Verbindung.

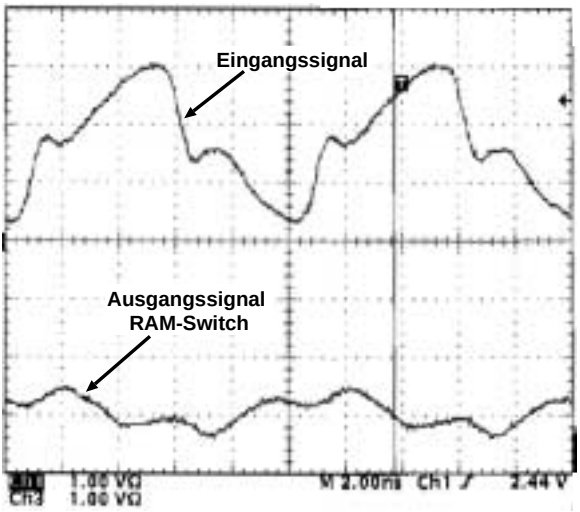


Abbildung A.5: Signalverläufe bei einer getrennten Verbindung eines 100MHz Signals.

Anhang B

Bilder des neuen FPGA-Koprozessors

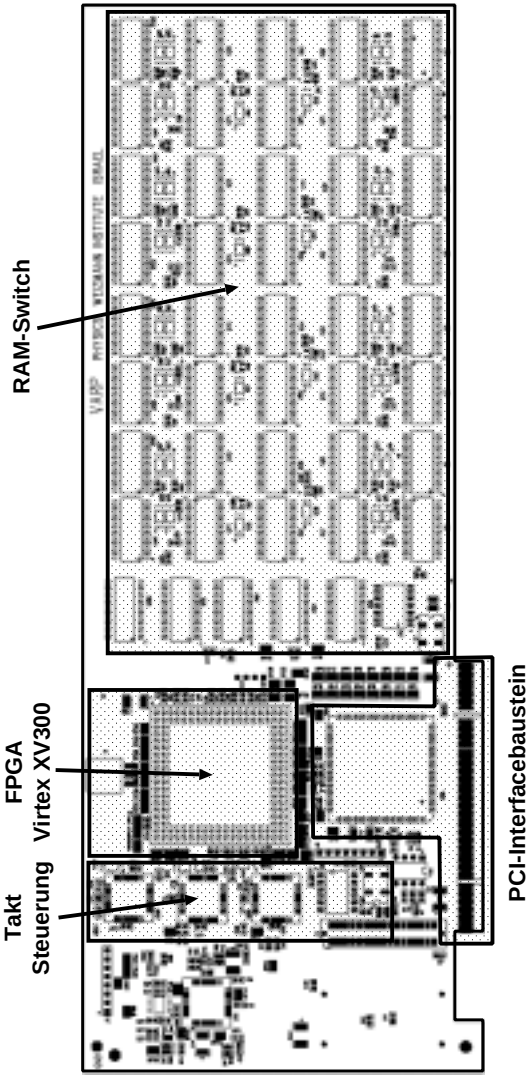


Abbildung B.1: FPGA-Koprozessor mit den Funktionsblöcken auf der Bestückungsseite.

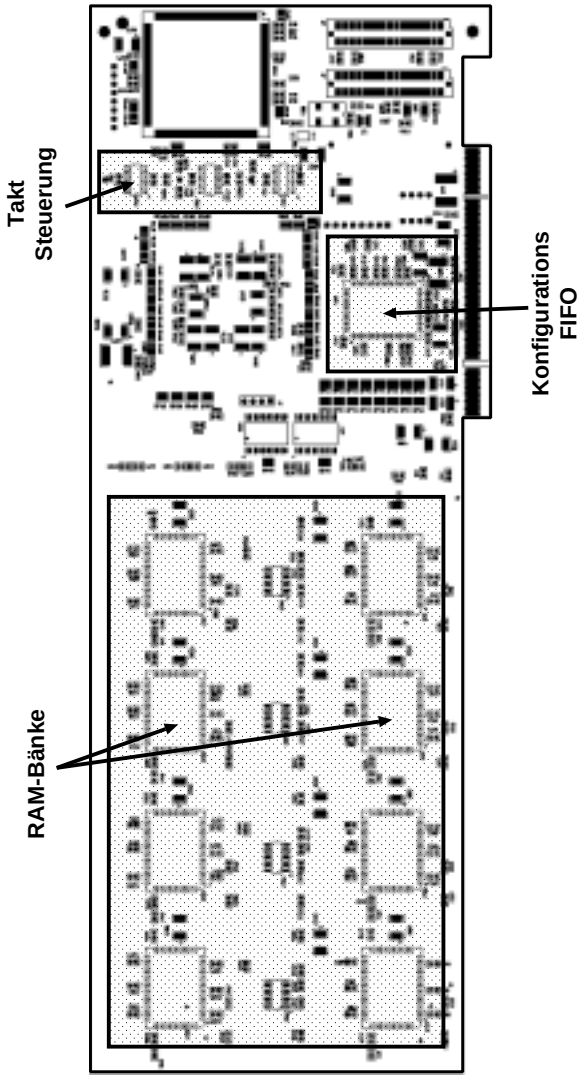


Abbildung B.2: FPGA-Koprozessor mit den Funktionsblöcken auf der Lötseite.

Abbildung B.3: Foto des neuen FPGA-Koprozessors.

Anhang C

Glossar

ALU: Arithmetic Logic Unit

API: Application Programmers Interface

ASIC: Application Specific Integrated Circuit

Block-RAM: Internes, festverdrahtetes RAM in modernen FPGAs

CHDL: 'C++'-based Hardware Description Language

CPLD: Complex Programmable Logic Device

CPCI: Compact PCI

CRC: Cyclic Redundancy Check

DD: Device Driver

DLL: Delay Locked Loop

DMA: Direct Memory Access

DSP: Digital Signal Processor

FCFS: First-Come-First-Served – Scheduling-Strategie

FIFO: First-In-First-Out – Speicher

FPGA: Field Programmable Logic Device

FPGA-Schaltung: Logische Funktion die auf einem FPGA-Baustein ausgeführt wird

FPIC: Field Programmable Inter Connect

FSM: Finite State Machine

H-OS: Host Operating System

HEP: High Energy Physics

HMU: Hardware Management Unit

IPC: Interprocess Communication

ISA: Industry Standard Architecture

JTAG: Joint Test Action Group

LIB: Softwarebibliothek

LINUX: Freies UNIX Betriebssystem

LUT: Look Up Table

MIMD: Multiple-Instructions-Multiple-Date

MLFS: Multilevel Feedback Sheduling – Scheduling-Strategie

NAND: Invertierte UND-Verknüpfung

NOR: Invertierte ODER-Verknüpfung

PAL: Programmable Array Logic

PCI: Peripheral Component Interconnect

PLL: Phase Locked Loop

PMC-PCI: PCI Mezzanine Card

PPC: Parallel Pipelined C

RR: Round-Robin – Scheduling-Strategie

RAM: Random Access Memory

ROM: Read Only Memory

RT-Modell: Register-Transfer Modell

SBUS: Sun BUS

SDRAM: Synchrone dynamic RAM

SIMD: Single Instruction Multiple Data

SJF: Shortest Jop First – Scheduling-Strategie

SLU: Swappable Logic Unit

SRAM: Statisches RAM

TSS: Task-Status-Segment

VHDL: Very High Speed Hardware Description Language

VME: VersaModule EuroCard Bus

Literaturverzeichnis

[Act99] Actel Corp., Sunnyvale, CA 94089, USA. *Actel Data Book*, 1999. `http://www.actel.com`.

[AFA+99] J. Carlos Alves, J. Canas Ferreira, C. Albuquerque, José F. Oliveira, J. Soeiro Ferreira, and J. Silvia Matos. FAFNER – Accelerating Nesting Problems with FPGAs. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 168–176, Los Alamitos, California, April 1999.

[AK96] A. Auer and R. Kimmelmann. *Schaltungstest mit Boundary Scan*. Hüthig Verlag, 1996. ISBN 3-7785-2519-0.

[Alt00] Altera Corporation, 101 Innovation Drive, San Jose, California 95134, USA. *Altera Data Book*, 2000. `http://www.altera.com`.

[Atm99a] Atmel Corp., San Jose, California 95131, USA. *Atmel AT40K Data Book*, 1999. `http://www.atmel.com/atmel/products/prod98.htm`.

[Atm99b] Atmel Corp., San Jose, California 95131, USA. *Atmel FSlip Data Book*, 1999. `http://www.atmel.com/atmel/products/prod93.htm`.

[Atm00] Atmel Corp., San Jose, California 95131, USA. *Atmel Data Book*, 2000. `http://www.atmel.com`.

[BAK96] Duncan A. Buell, Jeffrey M. Arnold, and Walter J. Kleinfelder. *Splash 2 – FPGAs in a Custom Computing Machine.* IEEE Computer Society Press, 1996. ISBN 0-8186-7413-X.

[BBD+95] M. Beck, H. Böhme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner. *Linux-Kernel-Programmierung.* Addison-Wesley Publishing Company, third edition, 1995. ISBN 3-89319-939-X.

[BDH+97] Jim Burns, Adam Donlin, Jonathan Hogg, Santam Singh, and Mark de Wit. A Dynamical Reconfiguration Run-Time System. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 66–75, Los Alamitos, California, April 1997.

[BH98] Peter Bellows and Brad Hutchings. JHDL - An HDL for Reconfigurable Systems. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 175–184, Los Alamitos, California, April 1998.

[Bra94] Barry E. Bray. *The Intel Microprocessors.* Macmillan Publishing Company, third edition, 1994. ISBN 0-02-314250-2.

[Bre96] Gordon Brebner. A Virtual Hardware Operating System for the Xilinx XC6200. In *6th International Workshop Field-Programmable Logic – FPL96*, pages 327–336, Springer-Verlag Berlin, September 1996.

[Bre97a] G. Brebner. The Swappable Logic Unit: A Paradigm for Virtual Hardware. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 77–86, Los Alamitos, California, April 1997.

[Bre97b] Gordon Brebner. Automatic Identification of Swappable Logic Units in XC6200 Circuitry. In *7th International Workshop Field-Programmable Logic – FPL97*, pages 173–182, Springer-Verlag Berlin, September 1997.

[Bre98a] Gordon Brebner. Circlets: Circuits as Applets. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 300–301, Los Alamitos, California, April 1998.

[Bre98b] Gordon Brebner. Field Programmable Logic: Catalyst for New Computing Paradigms. In *8th International Workshop Field-Programmable Logic – FPL98*, pages 49–58, Springer-Verlag Berlin, September 1998.

[Car00] Carl Carmichael. VIRTEX FPGA Series – Configuration and Readback. Technical Report XAPP138, Xilinx Inc., 2000.

[CKW95] Stephen Churcher, Tom Kean, and Bill Wilkie. The XC6200 FastMap Processor Interface. In *5th International Workshop Field-Programmable Logic – FPL95*, pages 36–43, Springer-Verlag Berlin, September 1995.

[Cyp] Cypress ICS525 Clock Generator Data Sheet. Internet Webside. `http://www.cypress.com`.

[DeH94] Andre DeHon. Dpga-Coupled Microprocessors: Commodity ICs for the Early 21th Century. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 31–39, Los Alamitos, California, April 1994.

[DG00] Klaus R. Dittrich and Stella Gatziu. *Aktive Datenbanken.* DPunkt Verlag, 2000. ISBN 3-932588-19-3.

[Eic96] Erich Eich. Betriebssysteme. Vorlesungsscript, 1996.

[Eil00] Lars Eilenbrecht. *Apache Web-Server.* MITP-Verlag, 2000. ISBN 3-8266-0612-4.

[GN96] Paul Graham and Brent Nelson. Genetic Algorithms In Software and In Hardware. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 216–225, Los Alamitos, California, April 1996.

[Gos91] A. Goscinski. *Distributed Operating Systems – The Logical Design*. Addison-Wesley Publishing Company, 1991. ISBN 0-201-41704-9.

[Han98] HandleC. Internet Webside, 1998. `http://www.celoxica.com`.

[Han99] HandleC, Gerat Britain. *HandleC Manual*, 1999. `http://www.celoxica.com`.

[Hez] S. Hezel. Verkehrszeichenerkennung. interner unveröffentlichter Report.

[HFHP97] Scott Hauck, Thomas W. Fry, Matthew M. Hosler, and Jeffrey P.Kao. The Chimaera Reconfigurable Functional Unit. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 87–96, Los Alamitos, California, April 1997.

[HH95] J.D. Hadley and B.L. Hutchings. Design Methodologies for Partially Reconfigured Systems. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 78–84, Los Alamitos, California, April 1995.

[HKL+95] H.Högl, A. Kugel, J. Ludvig, R. Männer, K.H. Noffz, and R. Lay. Enable++: A Second Generation FPGA Processor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 45–53, Los Alamitos, California, April 1995.

[HLA98] Rhett D Hudson, David I. Lehn, and Peter M. Athanas. A Run-Time Reconfigurable Engine for Image Interpolation. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 88–95, Los Alamitos, California, April 1998.

[HLS98] Scott Hauck, Zhiyuan Li, and Eric Schwabe. Configuration Compression for the Xilinx XC6200 FPGA. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 138–146, Los Alamitos, California, April 1998.

[HM99] S. Hezel and R. Männer. Schnelle Berechnung von 2D-FIR Filteroperationen mittels FPGA-Koprozessor microEnable. In *Mustererkennung 1999, Proc. des 21 DAGM Symposiums Bonn*, pages 250–257, September 1999.

[HP96] John L. Hennessy and David A. Patterson. *Computer Architecture – A Qualtitative Approach.* Morgan Kaufmann Publishers, second edition, 1996. ISBN 1-55860-329-8.

[HR98a] G. Haug and W. Rosenstiel. Reconfigurable Hardware as Shared Resources for Parallel Threads. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 320–321, Los Alamitos, California, April 1998.

[HR98b] Gunter Haug and Wolfgang Rosenstiel. Reconfigurable Hardware as Shared Resource in Multipurpose Computers. In *8th International Workshop Field-Programmable Logic – FPL98*, pages 149–158, Springer-Verlag Berlin, September 1998.

[Inc99] Intel Inc. *Intel Pentium II Data Book.* Intel Inc., 1999. `http://www.intel.com`.

[JTY+98] Jack Jean, Karen Tomko, Vikram Yavagal, Robert Cook, and Jignesh Shah. Dynamic Recofiguration to Support Concurrent Applications. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 302–303, Los Alamitos, California, April 1998.

[Jun00] Jungo, Israel. *WinDriver Manual*, 2000. `http://www.jungo.com`.

[KBH+98] K. Kornmesser, O. Brosch, S. Hezel, P. Dillinger, A. Kugel, M. Müller, C. Hinkelbein, H. Singpiel, H. Simmler, and R. Männer. Simulating FPGA-Coprocessors using FPGA Development System CHDL. In *Proc. PACT 98, Workshop on Reconfigurable Computing*, pages 78–82, Paris, 1998.

[KD98] Tom Kean and Ann Duncan. DES Key Breaking, Encryption and Decryption on the XC6216. In *IEEE Symposium on*

FPGAs for Custom Computing Machines, pages 310–311, Los Alamitos, California, April 1998.

[Kel97] Steve Kelem. Mapping a Real-Time video Algorithm to a Context-Switching FPGA. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 236–237, Los Alamitos, California, April 1997.

[Kel00] Steve Kelem. VIRTEX Configuration Architecture Advances Users' Guide. Technical Report XAPP151, Xilinx Inc., 2000.

[KJ97] J. Keller and W. J.Paul. *Hardware-Design*. Teubner Verlag, 1997. ISBN 3-8154-2304-X.

[KLL+98] A. Kugel, K. Kornmesser R. Lay, J. Ludvig, R. Männer, K.H. Noffz, S. Rühl, M. Sessler, H. Simmler, and H. Singpiel. 50kHz Pattern Recognition on the Large FPGA Processor Enable++. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 262–263, Los Alamitos, California, April 1998.

[LCH00] Zhiyuan Li, Katherine Compton, and Scott Hauck. Configuration Caching Management Techniques for Reconfigurable Computing. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages –, Los Alamitos, California, April 2000.

[Lie00] Gerhard Lienemann. *TCP/IP-Grundlagen*. Heise Verlag, second edition, 2000. ISBN 3-88229-180-X.

[Lin90] Wen C. Lin. *Handbook of Digital System Design*. CRC Press, 1990. ISBN 0-8493-4274-4.

[LM95] Eric Lemoine and David Merceron. Run Time Reconfiguration of FPGA for Scanning Genomic DataBases. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 90–98, Los Alamitos, California, April 1995.

[LMSS00] L. Levinson, R. Männer, H. Simmler, and M. Sessler. Preemptive Multitasking on FPGAs. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 301–302, Los Alamitos, California, April 2000.

[Luc98a] Lucent Technologies Inc., 555 Union Boulevard, Allentown, PA 18103, USA. *Lucent Data Book*, 1998. `http://www.lucent.com`.

[Luc98b] Lucent Technologies Inc., 555 Union Boulevard, Allentown, PA 18103, USA. *Lucent Orca 3C Data Book*, 1998. `http://www.lucent.com`.

[MMF98] Oskar Mencer, Martin Morf, and Michael J. Flynn. PAM-Blox: A High Performance FPGA Design for Adaptive Computing. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 167–174, Los Alamitos, California, April 1998.

[MS97] Laurent Moll and Mark Shand. System Performace Measurement on PCI Pamette. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 125–133, Los Alamitos, California, April 1997.

[MS98] Donald MacVicar and Satnam Singh. Acceleration DTP with Reconfigurable Computing Engines . In *8th International Workshop Field-Programmable Logic – FPL98*, pages 391–395, Springer-Verlag Berlin, September 1998.

[MSS00] R. Männer, M. Sessler, and H. Simmler. Pattern Recognition and Reconstuction on a FPGA-Koprocessor Board. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 325–326, Los Alamitos, California, April 2000.

[Mue00] Matthias Müller. interner unveröffentlichter Report, 2000. Performace Messung an zwei PCI Systemen.

[NL99] K.-H. Noffz and R. Lay. *microEnable*. Mannheim, Germany, 1999. `http://www.silicon-software.com`.

[Nof96] Klaus-Henning Noffz. *Ein FPGA-Prozessor als 2nd Level-Trigger für Atlas.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 1996.

[NS98] Jürgen Nehmer and Peter Sturm. *Systemsoftware.* DPunkt Verlag Heidelberg, 1998. ISBN 3-920993-74-8.

[Nye92] Adrian Nye. *Xlib Programing Manual.* O'Reilly & Associates, third edition, 1992. ISBN 1-56592-002-3.

[Opt] The OptiMagic Webside. Internet Webside. `http://www.optimagic.com`.

[Pal96] Samir Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis.* Book News Inc., 1996. ISBN 0134516753.

[Pat00] Cameron Patterson. High Performace DEC Encryption in Virtex FPGAs using JBits. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages –, Los Alamitos, California, April 2000.

[Per98] Douglas L. Perry. *VHDL.* McGraw-Hill, second edition, 1998. ISBN 0-07-049436-3.

[Per00] Pericom Semiconductor Inc., 2380 Bering Drive, San Jose, California 94131, USA. *Pericom 3.3V 32-Bit BusSwitch*, 2000. `http://www.pericom.com`.

[PLX00] PLX Technology Inc., 870 Maude Ave., Sunnyvale, California 94085, USA. *PLX IOP480 PCI-Interfacebaustein*, 2000. `http://www.plxtech.com`.

[PT97] David Pellerin and Douglas Taylor. *VHDL Made Easy.* Prentice Hall, 1997. ISBN 0-13-650763-8.

[PW87] Franklin P. Prossner and David E. Winkel. *The Art of Digital Design.* Prentice-Hall, second edition, 1987. ISBN 0-13-046673-5.

[RH97] Michael Rencher and Brad L. Hutchings. Automated Target Recognition on Splash 2. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 192–200, Los Alamitos, California, April 1997.

[Ric97] Jeffry Richter. *Windows – Programmierung für Experten.* Microsoft Press, third edition, 1997. ISBN 3-86063-389-9.

[RPG+99] D. Rusling, O. Pomerantz, S. Goldt, S. van der Meer, S. Burkett, M. Welsh, I. Bowman, S. Siddiqi, and M.C. Tanuan. *LINUX Programming – White Paper.* Coliolis Open Press, 1999. ISBN 1-57610-473-1.

[Rub98] Alessandro Rubini. *Linux Gerätetreiber.* O'Reilly Verlag, 1998. ISBN 3-89721-122-X.

[Rus99] David A. Rusling. The Linux Kernel, 1999.

[SA95] Tom Shanley and Don Anderson. *PCI System Architecture.* MindShare Inc., third edition, 1995. ISBN 0-201-40993-3.

[SBM00] H. Simmler, E. Bindewald, and R. Männer. Acceleeration of Protein Energy Calculation by FPGAs. In *Proc. Of the Int. Conf. on Mathem. and Engineeing Techniques in Medicine and Biological Science, METMBS00*, pages 177–183, June 2000.

[Sch97] Herman Schmit. Incremental Reconfiguration for Pipelined Applications. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 47–55, Los Alamitos, California, April 1997.

[SG94] Abraham Silberschatz and Peter B. Galvin. *Operating System Concepts.* Addison-Wesley Publishing Company, fourth edition, 1994. ISBN 0-201-50480-4.

[Sha97] Mark Shand. PCI Pamette V1. Technical report, DEC, Systems Research Center, Palo Alto, USA, 1997. `http://www.research.digital.com/SRC/pamette`.

[SHW+98] S. Sezer, J. Heron, R. Woods, R. Turner, and A Marshall. Fast partial reconfiguration for FCCMs. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 318–319, Los Alamitos, California, April 1998.

[Sim98] Harald Simmler. Experimente des Authors mit dem AT40K Baustein von Atmel, 1998.

[Sin00] Holger Singpiel. *Der Atlas LVL2 Trigger mit FPGA-Prozessoren.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2000.

[SJV95] Brian Schoner, Chris Jones, and John Villasenor. Issues in Wireless Video Coding using Run-time-reconfigurable FPGAs. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 85–89, Los Alamitos, California, April 1995.

[SKS+00] S. Singpiel, A. Kugel, H. Simmler, R. Männer, A.C. Castanon Vieira, F. Galvez-Durand, and J.M.S. de Alcantara. Implementation of Cryptographic Applications on a Reconfigurable FPGA Coprozessor microEnable. In *Proc. SBCCI2000, Symposium on Integrated Curcuits and System Design*, pages 359–362, Manaus, Amazonas, Brazil, September 2000.

[SLC98] N. Shirazi, W. Luk, and P.Y.K. Cheung. Run-Time Management of Dynamically Reconfigurable Designs. In *8th International Workshop Field-Programmable Logic – FPL98*, pages 59–68, Springer-Verlag Berlin, September 1998.

[SLM00] H. Simmler, L. Levinson, and R. Männer. Multitasking on FPGA-Coprocessors. In *10th International Workshop Field-Programmable Logic – FPL00*, pages 121–130, Springer-Verlag Berlin, September 2000.

[SS98] Satnam Singh and Robert Slous. Accelerating Adobe Photoshop with Reconfigurable Logic. In *IEEE Symposium on*

FPGAs for Custom Computing Machines, pages 236–244, Los Alamitos, California, April 1998.

[Sta] HAL300. Internet Webside. `http://www.starbridgesystems.com`.

[Ste90] Richard W. Stevens. *UNIX Network Programming*. Prentice Hall, forth edition, 1990. ISBN 0-13-49876-1.

[SV98] Stephen M. Scalera and Jóse R. Vázquez. The Design and Implementation of a Context Switching FPGA. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 78–85, Los Alamitos, California, April 1998.

[Tan92] A.S. Tannenbaum. *Modern Operating Systems*. Prentice-Hall, 1992.

[TCJW97] Steve Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. A Time-Multiplexed FPGA. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 22–28, Los Alamitos, California, April 1997.

[TG99] Andrew S. Tanenbaum and James Goodman. *Computerarchitektur*. Prentice-Hall, fourth edition, 1999. ISBN 3-8272-9573-4.

[tH95] Klaus ten Hagen. *Abstrakte Modellierung digitaler Schaltungen*. Springer Verlag, 1995. ISBN 3-540-59143-5.

[Tim] Time Logic - DeChyper. Internet Webside. `http://www.timelogic.com`.

[TW97] A.S. Tannenbaum and A.S. Woodhull. *Operating Systems: Design and Implementation*. Prentice-Hall, zweite edition, 1997.

[VHM+99] B. Vettermann, J. Hesser, R. Männer, H. Singpiel, and A. Kugel. Implementation of Algorithmically Optimized Volume Rendering on FPGA-Hardware. In *IEEE Vizualization 99*, San Francisco, USA, 1999.

[Vir97] Virtual Computer Corporation. VCC H.O.T. II, 1997. http://www.vcc.com.

[Wan98] Markus Wannemacher. *Das FPGA-Kochbuch*. International Thomson Publishing GmbH, Bonn, 1998. ISBN 3-8266-2712-1.

[WC96] Ralph D. Wittig and Paul Chow. Onechip: An Fpga Processor With Reconfigurable Logic. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 126–135, Los Alamitos, California, April 1996.

[Wie95] Michael Wielsch. *UNIX*. Data Becker, 1995. ISBN 3-8158-1166-X.

[WV97] John Woodfill and Brian Von Herzen. Read-Time Stereo Vision on the PARTS Reconfigurable Computer. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 201–209, Los Alamitos, California, April 1997.

[Xil96] Xilinx Inc., 2100 Logic Drive, San Jose, California 95124, USA. *XC6200 FPGA*, 1996. http://www.xilinx.com.

[Xil99a] Xilinx Inc., 2100 Logic Drive, San Jose, California 95124, USA. *The Programmable Logic Data Book*, 1999. http://www.xilinx.com.

[Xil99b] Xilinx Inc., 2100 Logic Drive, San Jose, California 95124, USA. *Xilinx Virtex FPGA*, 1999. http://www.xilinx.com.

[Xil00] Xilinx. Using the Virtex Delay-Locked Loop. Technical Report XAPP132, Xilinx Inc., 2000.

[Zoz97] Ralf Zoz. *Eine Hochsprachen-Programmierumgebung für FPGA-Prozessoren*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 1997.