Masterarbeit

Dynamische partielle Rekonfiguration von SoCs in einem Netzwerk zur Beschleunigung von verteilten Berechnungen

Frank Opitz

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Frank Opitz

Dynamische partielle Rekonfiguration von SoCs in einem Netzwerk zur Beschleunigung von verteilten Berechnungen

Masterarbeit eingereicht im Studiengang Master of Science Informatik
am Department Informatik
in der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor : Prof. Dr.-Ing. Bernd Schwarz
Zweitgutachter : Prof. Dr. Jürgen Reichardt

Abgegeben am 28. November 2011

Frank Opitz

Thema der Bachelorarbeit

Dynamische partielle Rekonfiguration von SoCs in einem Netzwerk zur Beschleunigung von verteilten Berechnungen

Stichworte

Verteiltes Rechnen, dynamische partielle Rekonfiguration, SoC, FPGA, ICAP, TEMAC, MicroBlaze, Ethernet, LwIP, RTOS

Kurzzusammenfassung

Die Berechnung komplexer Algorithmen oder von großen Datenmengen, wie sie beispielsweise bei der Untersuchung von Funksignalen auftreten, erfordern eine hohe Rechenleistung. Diese Leistung wird zum einen durch Supercomputer erreicht, zum anderen durch den Zusammenschluss vieler „normaler“ Computer zu einem Distributed Computing Netzwerk. Diese Computer haben durch ihren Aufbau sowie der darauf laufenden Software eine geringen Wirkungsgrad im Bezug auf das Berechnen der unterschiedlichen Aufgaben. Die Verwendung von FPGAs mit einer angepassten Hardware fand hier bisher keine Verwendung, da die Konfiguration dieser bereits beim Einschalten zu erfolgen hatte. Unter der Verwendung der dynamischen partiellen Rekonfiguration wird ein Distributed Computing Netzwerk entworfen, welches die Berechnungen auf FPGAs ausführt. Hierzu wird in dieser Arbeit ein SoC-Client sowie ein Server für den Aufbau eines Netzwerkes und ein Beispiel Projekt entwickelt. Der SoC-Client basiert dabei auf einem Xilinx FPGA und dem MicroBlaze Mikroprozessor. Dieser ist über ein Ethernet Interface an den Server gebunden und empfängt die Rekonfigurationsdateien und Datensätze, welche zu berechnen sind. Der rekonfigurierbare Bereich wird dabei über ein Interface als Bus-Master an das System angeschlossen, so dass dieser Daten eigenständig aus dem Speicher lesen und in den Speicher schreiben kann. Die Topologie des Netzwerkes basiert auf der Client-Broker-Server Architektur, um die vorhanden SoCs auf die Projekte zu verteilen. Die Projekte stellen hierbei die einzelnen Konfiguration und die dazugehörigen Datensätze bereit. Dieses wurde mit einer Bildverarbeitung aus einem autonomen Fahrzeug erfolgreich getestet.

Frank Opitz

Title of the paper

Dynamic Partial Reconfiguration from SoCs in a distributed computing network

Keywords

distributed computing, dynamic partial reconfiguration, SoC, FPGA, ICAP, TEMAC, MicroBlaze, Ethernet, LwIP, RTOS

Abstract

The computation of complex algorithms or large amounts of data, such as those occur in the study of radio signals, require a high computing power. This performance is achieved firstly through supercomputers or by the merge of many other "normal" computers to a distributed computing network. These computers have in their structure and the software running on the computer a low efficiency in terms of calculating the different tasks. The use of FPGAs with an adjusted hardware has not been used for distributed computing, since the configuration of the FPGAs has to be done when the system is switched on. By using the dynamic partial reconfiguration a distributed computing network is designed, which performs computations on FPGAs. For this purpose a SoC client and a server for the construction of a network and a sample project are developed in this work. The SoC client is based on a Xilinx FPGA and a MicroBlaze Microprocessor. It is also connected to the server with a ethernet interface and receives reconfiguration data and records, which are calculated. The reconfigurable area is connected via an interface as a bus master to the system so that this can independently read data from the memory and write onto the memory. The topology of the network is based on the broker-client-server architecture to distribute the available SoCs on the projects. The projects provide the individual configurations and the associated records. This system was successfully tested with a image processing of an autonomous vehicle.

Inhaltsverzeichnis

1 Einleitung **1**

2 Arbeiten im Bereich des reconfigurable Computings **6**
2.1 Analyse von Verschlüsselungs-Algorithmen mit einem FPGA Cluster 6
2.2 Verwendung der partiellen Rekonfiguration zur Verbesserung von Bildverarbeitungsalgorithmen 9
2.3 Die partielle Rekonfiguration als Unterstützung eines SoC basierten biometrischen Authentifizierungssystems 11
2.4 Simulation einer Partiellen Rekonfiguration in einer mobilen Agenten Umgebung 16

3 Technologieübersicht zur Realisierung des SoC-Clienten mit einem Xilinx FPGA **20**
3.1 ML605 Entwicklungsboard mit Virtex 6 FPGA 20
3.2 MicroBlaze MicroProzessor zur Steuerung des Systems 20
3.3 Erstellen eines DPR-Systems und Verwaltung von FPGA Konfiguration mit PlanAhead 22
3.4 Tri-Mode Ethernet Media Access Controller 24
3.5 Hardware Internal Configuration Access Point (HWICAP) 25
3.6 System Advanced Configuration Environment (SYSACE) 26
3.7 Multi-Port Memory Controller 27
3.8 Real-time Operating System mit POSIX Standard 27
3.9 IP Stack zur softwareseitigen Anbindung des Ethernetcontrollers 28
3.10 Motorola S-Record zur Speicherung von ausführbarem Code im Flash 30

4 Architektur des FPGA basierten Distributed Computing Systems **32**
4.1 SoC-Client zur Berechnung der Datensätze im Distributed Computing System . 33
4.2 Bereitstellen der Daten über den Projekt-Client 37
4.3 Broker zur Verteilung der SoC-Clienten und der Datensätze 40

5 Implementierung des SoC-Clients **43**
5.1 Statisches Subsystem mit MicroBlaze zur Kommunikation mit dem Broker . . 43
5.2 Interface zur Kommunikation mit dem dynamischen Subsystem 47
5.3 Definition des dynamischen Bereiches 52
5.4 Software Client des statischen SoC Systems 53
5.4.1 Dateisystem zur Speicherung der PRM-Dateien im SoC-Client 57
5.4.2 Speicherung der Ein-und Ausgangsdateien im SoC-Client 60
5.5 Initialisierung der SoC Konfiguration und der Client Software 61

6 Broker zur Verteilung der Ressourcen implementiert mit Java **65**
6.1 Verwaltung der Clients in der Client-Factory 65
6.2 Project-Factory zur Verwaltung von Projekten 67
6.3 Scheduling der SoC-Clients und der Projekte 72

7 Beispiel Projekt zur Verifizierung von Bildverarbeitungsalgorithmen eines autonomen Fahrzeuges **74**
7.1 Einbindung der Projekt IPs 75
7.2 Projekt Software für die Server-Kommunikation und Daten-Vorverarbeitung . . 76
7.3 Messtechnische Analyse der Funktion und des Zeitverhaltens 78

8 Zusammenfassung und Ausblick **81**
8.1 Zusammenfassung 81
8.2 Ausblick 82

Literatur **83**

Abbildungsverzeichnis **87**

A Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform **i**

B Ressourcenbedarf des statischen SoC-Clients **iii**

C Messung zum Datendurchsatz der XILMFS und XILFATFS Dateisysteme **iv**

D Hardware Konfiguration des MicroBlaze Systems **vii**

E Software Konfiguration des MicroBlaze Systems **xiii**

F Auszug aus dem PLB-Master Interface zum Lesen und Schreiben der Daten **xv**

G DVD mit Soft- und Hardware Projekten **xvii**

1 Einleitung

Die stetig wachsende Komplexität von zu berechnenden Daten erfordert den Bau sogenannter Supercomputer. Der zur Zeit schnellste Supercomputer, mit einer Rechenleistung von 2507 Tera FLOPS, ist der Tianhe-1A, welcher im National Supercomputer Center in China für die Berechnung von chemischen und physikalischen Prozessen verwendet wird, beispielsweise der Simulation von Flugzeugkomponenten ([NVIDIA (2010)],[Wikipedia (2010)]). Die Entwicklung dieses Supercomputers ist von 200 Wissenschaftlern innerhalb von zwei Jahren vollzogen worden und kostete etwa 88 Millionen USD [Raman (2010)]. Diese Kosten sowie die jährlichen Energiekosten, von etwa 2,7 Millionen USD, sind vom Träger des Projektes zu tragen. Für mittelständische Firmen sowie Universitäten sind Projektkosten dieser Größenordnung nicht zu finanzieren, so dass hierfür komplexe Probleme die Rechnungen auf mehrere Rechner verteilt werden, wodurch ein virtueller Supercomputer entsteht.

Diese verteilten Berechnungen verbinden einzelne Rechner über eine bestehende Netzwerk Infrastruktur zu einem „Distributed Computing System“ (DCS). Zumeist bestehen derartige Systeme aus Freiwilligen, die die Rechenleistung ihrer PCs verschiedenen Projekten zur Verfügung stellen, wie es beim „Berkeley Open Infrastructure for Network Computing“ (BOINC) der Fall ist. Diese Plattform dient der Verwaltung der Ressourcen, wie CPUs und Speicher, und stellt sie wissenschaftlichen und kommerziellen Projekten zur Verfügung. Das BOINC ist aus dem SETI@Home Projekt entstanden, welches in den ersten Versionen des Projektes die wissenschaftliche Arbeit sowie die Verwaltung der Ressourcen durchgeführt hat. Die Aufteilung des eigentlichen Projektes in BOINC und den wissenschaftlichen Anteil, sorgte für einen Anstieg der interessierten Anwender, da die Art der Projekte stetig gewachsen ist [California (2010)]. Folgende Projekte geben einen kurzen Einblick in die Vielseitigkeit der wissenschaftlichen Themen:

- ClimatePrediction.net:
  Dieses Projekt umfasst die Berechnung von Wettermodellen der nächsten 50 bis 100 Jahre, um Aussagen über die globale Erwärmung zu treffen. Das ursprüngliche Projekt diente der Eingrenzung des Parameterraums, um Wettermodelle für die Zukunft erstellen zu können, und wurde mit dem HadSM3-Modell durchgeführt. Dieses Modell erlaubt eine detaillierten Simulation der Atmosphäre, beinhaltet jedoch ein vereinfachtes Ozean Modell. Berechnet wurde das Modell in drei Schritten:

  1. Mit den gewählten Parametern wird berechnet, ob eine stabile Wetterlage erreicht wird.
  2. Die Werte des Ozeans werden verändert und der CO_2 Gehalt der Luft auf einen vorindustriellen Wert gesetzt.
  3. Der CO_2 Gehalt der Luft wird verdoppelt.

  Bei der Berechnung jedes Schrittes wird überprüft, ob mit diesen Parametern ein konstantes Klima erreicht werden kann. Ist dies nicht der Fall, so wird die Berechnung abgebrochen und die Berechnung mit neuen Parametern gestartet. Das zweite Experiment von ClimatePrediction.net startete 2005 mit einem neuen Klimamodell. Für dieses Modell wurden die Parameter gesucht, die auf das Wetter der Jahre 1950 - 2000 passen. 2006 startete das eigentliche Projekt, welches mit den zu berechneten Parametern eine Vorausberechnung für die Jahre 2000 bis 2100 erstellt [Oxford (2010)].

- Einstein@home:
  Nach Gravitationswellen, welche entstehen, wenn stark verdichtete, schnell rotierenden Sterne die Raumzeit um sich herum deformieren, sucht dieses Projekt. Die These dazu wurde von Einstein in der allgemeinen Relativitätstheorie aufgestellt. Der Beweis, dass diese existieren, fehlt aber bis heute. Ausgewertet werden hier Datensätze, die bei der Messung mit einem Michelson-Interferometer entstehen. Hier werden mit Lasern die Längen von rechtwinklig angeordneten Vakuumröhren gemessen, was bis auf eine Genauigkeit von ca. 10^{-18} m erfolgt [Max Planck (2010)].

- FightAIDS@Home:
  FightAIDS@Home ist ein Projekt aus dem pharmazeutischen Bereich. Hier wird nach einem Medikament gesucht, welches zur Bekämpfung des HI-Virus am besten geeignet ist. Der HI-Virus unterliegt einer starken Mutation, welche simuliert wird, um für viele Mutationen ein passendes Medikament zu finden. Genauer wird ein Molekül gesucht, welches viele verschiedene HIV Proteasen, welche an die menschlichen Zellen andocken, blocken kann [Olson-Laboratory (2010)].

Ist die Bearbeitung von Daten in einer Echtzeit Umgebung erforderlich, so werden FPGAs zur Berechnung eingesetzt. Diese haben die Eigenschaft die implementierten Algorithmen, anders als in einer CPU, nebenläufig ablaufen zu lassen. So wird ein höherer Datendurchsatz bei einer geringen Frequenz erreicht. In Tabelle 1 ist der Vergleich des Datendurchsatzes von einem Virtex2 FPGA und einem Itanium 2 Prozessor dargestellt. Gezeigt wird, dass bei einer um den Faktor zehn geringeren Frequenz die vierfache Anzahl an Berechnungen auf dem FPGA durchgeführt werden.

Tabelle 1: *Vergleich eines Virtex 2 FPGAs mit einem Itanium 2 Microprozessor [Xilinx (2006b)].*

	Microprozessor Itanium 2	FPGA Virtex 2VP100
Technologie	0,13 Micron	0,13 Micron
Frequenz	1,6 GHz	180 MHz
Durchsatz des internen Speichers	102 GBytes/s	7,5 TBytes/s
Recheneinheiten	5 FPU(2MACs + 1FPU) + 6 MMU + 6 Integer Unit	212 FPU oder 300+ Integer Units u.v.m.
Verbrauch	130 Watt	15 Watt
Performanz (Spitze)	8 GFLOPs	38 GFLOPs
Performanz (Mittel)	2 GFLOPs	19 GFLOPs
Durchsatz von I/O-Ports	6,4 GBytes/s	67 GBytes/s

Bisherige FPGA-basierte Systeme waren durch die Größe des FPGAs begrenzt und wurden beim Systemstart konfiguriert. Hier wurde die komplette Konfiguration beim Start in den FPGA geschrieben, so dass die Größe des FPGAs die Anzahl und Komplexität der Module bestimmt. Überschreitet die Anzahl der benötigten Ressourcen die Kapazität des FPGAs, so wurde die Anzahl der FPGAs erhöht oder eine komplette Rekonfiguration des FPGAs vorgenommen, wobei die einzelnen Konfigurationen unterschiedliche Module integriert hatten. Die Rekonfiguration des FPGAs erfolgte, je nach Größe des FPGAs und der Module, in mehreren Millisekunden, in

denen der FPGA reaktionsunfähig war, was besonders in sicherheitskritischen Systemen nicht akzeptabel ist. Zur Verkürzung der Reaktionsunfähigkeit wird mit der partiellen Rekonfiguration nur ein Teil des FPGAs rekonfiguriert. In dieser Zeit kann der FPGA aber weiterhin nicht verwendet werden. Die Weiterentwicklung der partiellen Rekonfiguration, die „Dynamische Partielle Rekonfiguration“ (DPR), erlaubt es, dass sicherheitskritische Module ihre Arbeit fortsetzen, während Teile des FPGAs rekonfiguriert werden. Dazu findet eine Unterteilung des FPGAs in dynamische Bereiche, sogenannte „Partial Reconfiguration Regions“ (PRR), und statische Bereiche statt. Diese statischen Bereiche sind in der Lage über einen „Internal Configuration Acces Point“ (ICAP) den FPGA selbst zu rekonfigurieren [Xilinx (2008a)].

Das Projekt „dynamische partielle Rekonfiguration von SoCs in einem Netzwerk zur Beschleunigung von verteilten Berechnungen“ zielt auf eine Kombination aus dem „Distributed Computing“ und der „Dynamischen partiellen Rekonfiguration“ ab. Angestrebt wird ein Netzwerk aus „System on Chips“ (SoC), welches zur Berechnung komplexer Daten verwendet wird. So ist keine größere Anzahl an FPGAs für ein Projekt anzuschaffen wie beispielsweise beim COPACOBANA-Projekt, das bis zu 120 FPGAs in einem System vereint. Diese werden über einen PC angesteuert und dienten in der eigentlichen Entwicklung dem Bruteforce von DES verschlüsselten Daten [Kumar u. a. (2006)]. Die in dieser Masterarbeit zur Berechnung verwendeten FPGAs werden durch die Teilnehmer des Netzwerkes bereitgestellt und von einer zentralen Instanz verwaltet, welche die FPGAs nach Bedarf zur Verfügung stellt (vgl. Abbildung 1).

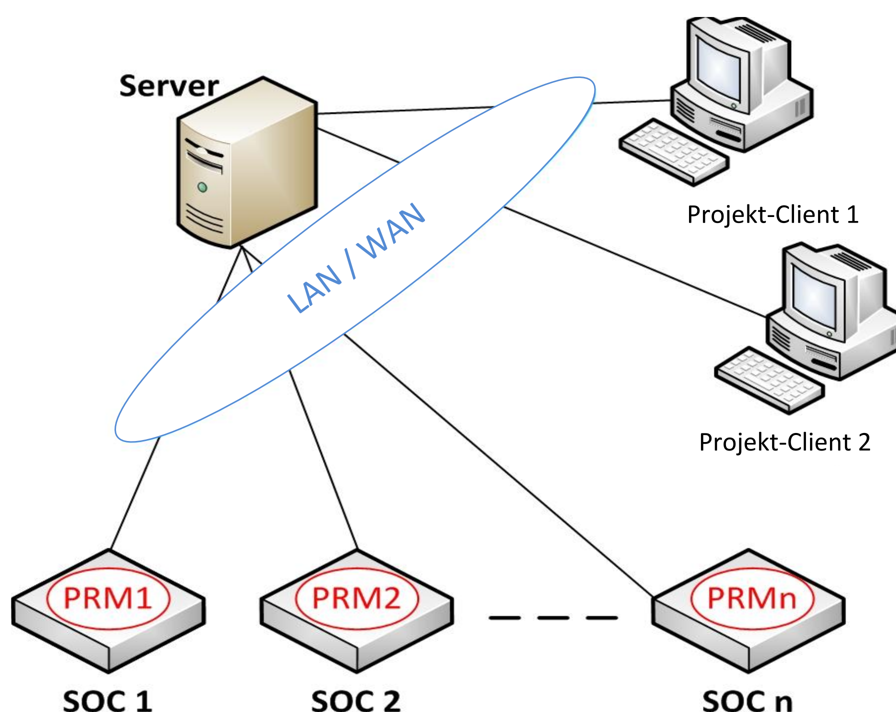


Abb. 1: *Architektur des Distributed Computing Systems mit den SoC-Clienten sowie den zu bearbeitenden Projekt und der Kommunikation über einen Server, welcher die Verteilung der Rechenleistung vornimmt.*

Im Verlauf dieser Masterarbeit wurde ein Prototyp der „FPGA basierten Distributed Computing Plattform“ entwickelt. Hierzu war der Schwerpunkt auf den Entwurft des FPGAs zu legen. Dieser stellt die Kommunikation mit dem Netzwerk sowie die Rekonfiguration bereit. Dazu wurde ein Sever entworfen, der die Verwaltung der FPGAs und der Projekte übernimmt. Zum Testen der Implementierung wurde ein Beispiel Projekt verwendet, welches Datensätze und Rekonfiguraitonsmodule bereitstellt. Konkret waren folgenden Aufgaben zu erledigen:

- SoC-Client: Der Design-Flow eines PR-Systems war zu durchdringen und die Unterschiede zu einem Standard-SoC aufzuzeigen. Weiter war das Erstellen des dynamischen und des statischen System aufzuzeigen, um eine Ergänzung zu den Ausarbeitungen zu erhalten, die ausschließlich eine mögliche Anwendung beinhalten.
  - statisches System (vgl. Abbildung 2): Ein System zur Steuerung des SoCs sowie der Kommunikation mit dem Server war zu entwerfen. Als Basis hatte hier das Xilinx EDK zu dienen und die bereitgestellten IPs waren zu durchdringen.
  - dynamisches System: Die Anbindung des dynamischen an das statische System waren so zu entwerfen, dass eine Kompatibilität zu einer großen Anzahl an Projekten erreicht wird. Ebenfalls war eine Effiziente Datenübertragung innerhalb des SoCs zu evaluieren.
- Server:
  - Netzwerk Architektur: Bei der Entwicklung des System war eine geeignete Architektur des Systems zu entwerfen. Hierbei war die Einschränkung des SoCs, bezüglich der Anzahl der Verbindungen, sowie eine optimale Erreichbarkeit und Datendurchsatz zu berücksichtigen.
  - Implementierung: Die Implementierung des Servers erfolgte auf Basis der entworfenen Architektur und der Programmiersprache Java.
- Projekt-Client: Es war ein Beispiel Projekt zu finden und die Module an das entwickelte Interface anzupassen. Weiter war ein Projekt-Client zu entwerfen, welcher die Module sowie Datensätze bereitstellt und die jeweiligen Antworten empfängt und evaluiert.

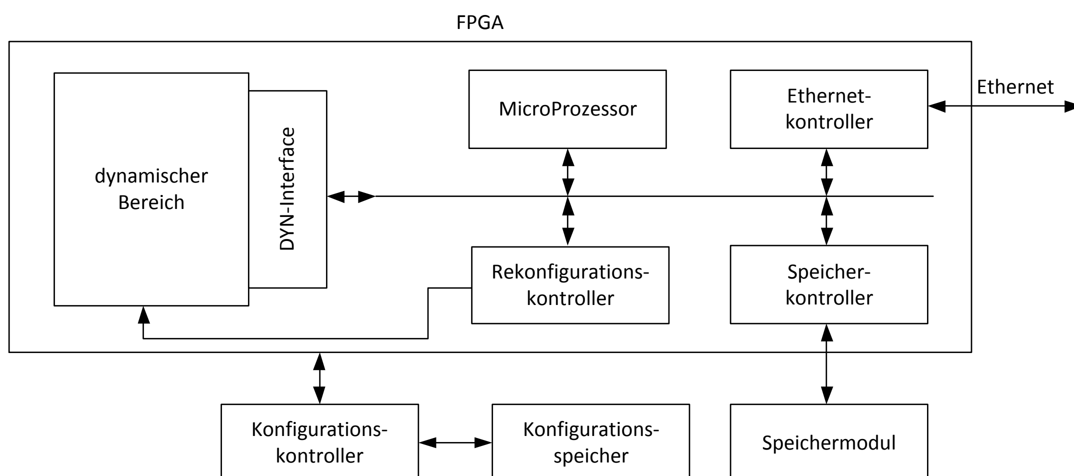


Abb. 2: *Systemaufbau des SoC-Clients mit den Modulen zur Ansteuerung des Speichers, des dynamischen Bereiches sowie der Kommunikation zum Server*

Die Arbeit ist wie folgt aufgebaut:

- Kapitel 2 stellt aktuellen Arbeiten vor, die sich im Schwerpunkt mit FPGAs beschäftigen. Hierbei werden Projekte aus den Bereichen der partiellen Rekonfiguration, dem verteilten Berechnen und der Verwendung von FPGAs in einem Cluster vorgestellt.
- Kapitel 3 gibt eine Übersicht, über die angewandten Technologien. Beschrieben werden hier beispielsweise die IP-Module, sowie Entwicklungssoftware und das verwendete Real Time Operating System.

- In Kapitel 4 wird der geplante Aufbau des Systems dargestellt, welches auf einer mehrstufigen Architektur basiert, in der FPGA-basierte SoCs die Verarbeitung von den zur Verfügung gestellten Projektdaten übernehmen. Weiter wird das zur Kommunikation zwischen dem Server und dem SoC-Client verwendete Overlay Protokoll beschrieben.
- Kapitel 5 stellt den Aufbau des statischen Systems mit Komponenten zur Kommunikation mit dem Server und zum Speichern der Daten vor. Es wird auf den Anschluss des dynamischen Systems eingegangen, welches aus Projekt 1 übernommen wurde. Zur Analyse des Systems wird der Ressourcenbedarf der einzelnen Komponenten sowie des Gesamtsystems dargestellt. Weiter wird die Konfiguration des Board Support Packages erläutert und die Funktion der erstellten Software beschrieben, wobei in diesem Projekt die Funktion zur Übertragung der PRM sowie die Rekonfiguration realisiert wurde.
- Kapitel 6 beschreibt den Aufbau des erstellten Servers sowie die Verwaltung der ankommenden Verbindugen durch einen SoC- sowie einen Projekt- Manager. Die implementierten Funktionen zum Versenden der PRM werden ebenfalls vorgestellt.
- Kapitel 7 stellt ein Beispiel Projekt vor. Das Projekt entstand aus der Fahrspurerkennung eines autonomen Fahrzeuges und berechnet die Ausgabe von verschiedenen Filtern, welche so für den eigentlichen Einsatz getestet werden.

2 Arbeiten im Bereich des reconfigurable Computings

Dieses Kapitel stellt vier Arbeiten vor, die ihren Schwerpunkt auf die Verwendung von FPGAs für die Implementierung von Algorithmen setzen, bzw. den Aufbau von FPGAs als Basis für rekonfigurierbare Systeme nutzen. Vorgestellt werden Projekte aus dem Bereich FPGA-Cluster, zur partiellen Rekonfiguration und zum Distributed Computing mit FPGAs:

- Analyse von Verschlüsselungs-Algorithmen mit einem FPGA Cluster:
  In diesem Projekt werden bis zu 120 FPGAs zu einem Cluster zusammengesetzt, um die Sicherheit von Verschlüsselungs-Algorithmen zu testen. Diese werden in einem System vereint, um eine hohe Verfügbarkeit zu erhalten. Die Kosten für das System betragen dabei ca. 10.000 €. Durch den Zusammenschluss der FPGAs wird ein bis zu den Faktor 650 effizienteres System, verglichen mit einer reinen Software Lösung, geschaffen.
- Verwendung der partiellen Rekonfiguration zur Verbesserung von Bildverarbeitungsalgorithmen:
  Der Einsatz der partiellen Rekonfiguration in einem Projekt zur Analyse von Bilddaten der zellularen Mikroskopie. Die Verarbeitungspipeline wird dabei, basierend auf einer Software-Pipeline, erzeugt. Die Bilddatenverarbeitung erfolgt dabei offline, nach dem Speichern der kompletten Daten, um die partielle Rekonfigruation zu erproben.
- Die partielle Rekonfiguration als Unterstützung eines SoC basierten biometrischen Authentifizierungssystems:
  Ebenfalls mit Bildverarbeitung und der partiellen Rekonfigruation beschäftigt sich dieses Projekt. Hier werden Fingerabdrücke untersucht und bei einem vorhandenen Vergleichsabdruck die Authentizität des Anwenders bestätigt. Die Verarbeitung der Bilddaten erfolgt ebenfalls in einer Pipeline, wobei die einzelnen Stufen nacheinander in die PRR geladen werden und die Bilddaten jedesmal zwischen zu speichern sind. So wird Echtzeit in dem Sinne erreicht, dass der Anwender keine Rechenzeit bemerkt.
- Simulation einer partiellen Rekonfiguration in einer mobilen Agenten Umgebung :
  Dieses Projekt befasst sich mit der Verteilung von Aufgaben auf einen Schwarm von Drohnen. Hierbei wird die partielle Rekonfigruation simuliert, da den Authoren die aktuellen Beschränkungen der PR zu groß sind. So werden die FPGAs mit vorher synthetisierten möglichen Konfigurationen komplett rekonfiguriert, wobei die Zustände der einzelnen Drohnen vollständig zu sichern sind.

2.1 Analyse von Verschlüsselungs-Algorithmen mit einem FPGA Cluster

In [Guneysu u. a. (2008)] wird die Rechenkraft von gekoppelten FPGAs zum Testen von Verschlüsselungs-Algorithmen verwendet. Bei diesen Algorithmen hängt die Sicherheit des Verfahrens zumeist mit der Länge des Schlüssels zusammen, wobei diese anhand der geschätzten Rechenkraft des Angreifers gewählt wird. So wird das Verfahren, bei einem starken Anstieg der Rechenkraft des Angreifers, schnell unsicher. Beispielsweise wird bei einer Steigerung der Perfomance im Berechnen von Schlüsseln um den Faktor 1000 die effektive Schlüssellänge um 10 Bit ($1000 \approx 2^{10}$) gekürzt. Das Untersuchen von aktuellen Verschlüsselungsverfahren wird unter der Verwendung von parallelen Berechnungen durchgeführt, die mehr als 2^{40} Operationen durchführen. Diese parallele Berechnung wird durch den Aufbau der Verschlüsselungsalgorithmen, die auf unabhängige Operationen setzen, ausgeführt. Die Verwendung von spezieller

Hardware zur Untersuchung der Algorithmen war bis vor einiger Zeit Regierungen vorbehalten, da das Entwickeln und Herstellen der Hardware hohe Kosten verursacht. Diese werden unter der Verwendung von rekonfigurierbarer Hardware so weit gesenkt, dass die spezifische Hardware auch außerhalb von Regierungen verwendet werden kann.

Bisherige Implementierungen von Clustern zur Kryptoanalyse setzen beispielsweise auf die Verwendung von Distributed Computing Systemen wie dem BOINC Netzwerk. Diese haben, gegenüber lokalen Clustern den Nachteil, dass der Erfolg des Projektes stark von der Anzahl der angemeldeten Anwender abhängt. Im Gegensatz dazu bieten Supercomputer durchgehend die gleiche Leistung, sie machen diese Eigenschaft aber durch ihre hohen Kosten in der Anschaffung und Wartung aber wieder weg. Um die jeweiligen positiven Eigenschaften, wie niedrige Kosten und hohe Verfügbarkeit, zu bündeln, wurde der „Cost-Optimized Parallel Code Breaker“ (COPACOBANA) entwickelt. Dieser besteht aus bis zu 120 FPGAs, welche über einen Bus verbunden sind, der einen Datendurchsatz von 1,6 Gbps bereitstellt.

Unter der Verwendung dieser Plattform werden aktuelle Sicherheitssysteme untersucht. Beispielsweise der e-Ausweis oder Norton Diskreet, welches zum Verschlüsseln von Festplatten und Dateien verwendet wird. Hierzu wird die geringe Länge der Schlüssel für diese Systeme ausgenutzt, Systeme wie beispielsweise AES mit mehr als 128 Bit Schlüsseln, sind auch für das COPACOBANA Projekt nicht in annehmbarer Zeit zu berechnen.

Der Aufbau des Systems erfolgte anhand der Annahmen, dass die Operationen parallel durchgeführt werden können, dass nur geringe Kommunikation benötigt wird und dass kaum Eingriffe von außerhalb notwendig sind, so dass ein konventioneller PC zur Initialisierung des Systems ausreicht. Weiter sind die Algorithmen so zu implementieren, dass der FPGA-interne Speicher ausreicht. Auf Basis dieser Annahmen wurden Platinen entwickelt, die eine optimale Kosten-Perfomance bieten, da bereits verfügbare Platinen einen starken Overhead an Peripherie mitbringen, welche nicht benötigt wird (vgl. Abbildung 3). Die Anbindung an einen PC erfolgt zur Konfiguration des Systems sowie zur Übertragung von Datensätzen. Zur Kommunikation kann wahlweise ein USB oder Ethernet Modul verwendet werden, wobei mehrere COPACOBANA Systeme an einen PC angeschlossen werden können.

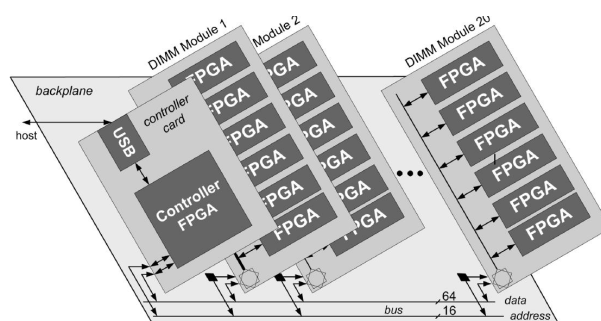


Abb. 3: *Hardware Architektur des COPACOBANA Systems [Guneysu u. a. (2008)]*

Verwendet wird COPACOBANA unter anderem zur vollständigen Untersuchung von Schlüssel-Räumen zum Entschlüsseln von symetrischen Verschlüsselungen. Die Längen der jeweiligen Schlüssel wird zumeist so gewählt, dass einerseits eine kostengünstige Implementierung erfolgen kann, andererseits Brute-Force Attacken ineffizient macht. So wurde 1977 der DES mit 56 Bit Schlüsseln zum ersten industriellen Standard, da die hohen Entwicklungskosten zu der Zeit einen Brute-Force Angriff unwirtschaftlich machten. 1977 sollte eine Brute-Force Maschine, die den Angriff in einem Tag durchführen sollte, 20 Millionen $ kosten [Diffie und Hell-

man (1977)]. 1998 wurde dann bereits eine Maschine gebaut, die 56 Stunden brauchte um den Schlüsselraum zu durchlaufen und 250.000$ kostete [Foundation (1998)].

COPACOBANA verwendet eine angepasste DES-Engine der Ucl Crypto Group [Rouvroy u. a. (2003)]. Basierend auf einer 21 Stufigen Pipeline wird mit jedem Takt einen Schlüssel getestet. Vier dieser Module werden auf einem FPGA zusammengefasst, die bei der Berechnung den gleichen Text verwenden und den Schlüsselraum teilen (Abbildung 4).

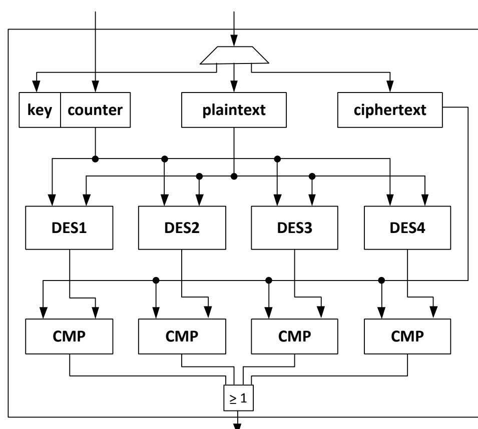


Abb. 4: *Anordnung der DES-Module in einem FPGA mit geteiltem Schlüsselraum und Datensätzen [Guneysu u. a. (2008)]*

Seit der ersten Vorstellung des COPACOBANA Systems wurde eine Steigerung in der Perfomance um 36% erreicht, so dass aktuell ein System-Takt von 136 MHz verwendet wird. Mit dieser verbesserten Leistung wird ein Schlüsselraum von 2^{42} Schlüsseln in $2^{40} \times 7,35$ ns $= 135$ min überprüft. Bei einem voll ausgerüstetem COPACOBANA System werden 120 FPGAs eingesetzt, so dass $4 \times 120 = 480$ Schlüssel alle 7,35 ns überprüft werden, was eine Anzahl von 65 Milliarden Schlüsseln pro Sekunde ergibt. Bei einem Schlüsselraum von 2^{52} Bit werden 6,4 Tage zur Berechnung benötigt. Im Vergleich mit einer Software Lösung, die auf gleichwertigen Standard-PCs implementiert wird, ergibt sich eine 650 fache schnellere Berechnung der Schlüssel auf dem COPACOBANA System. Hierzu kommt der Energie Verbrauch, der um den Faktor 8,7 niedriger ist als auf der PC basierten Implementierung.

Fazit

Die Effizienz dieses Projektes zeigt den Vorteil von Hardware Implementierungen bestimmter Algorithmen und diente als Basis der Idee zum „Distributed SoC Network“, da die Kosten für ein COPACOBANA, von ca. 10.000 €, nicht von jedem Projekt getragen werden können. Zur Steigerung des Durchsatzes wurde hier das Erstellen eines Clusters gegenüber einer verteilten Lösung bevorzugt. Dies hat den Vorteil, dass immer eine konstante Anzahl an FPGAs dem Projekt zur Verfügung steht. Diese Entscheidung ist für jedes Projekt zu treffen. So ist es beispielsweise für einen schnellen Test von Hardware-Modulen oder Systemen, die kurzzeitig eine hohe Anzahl an FPGAs benötigen, sinnvoller diesen in einem verteilten System durchzuführen, ohne die hohen Anschaffungskosten in Kauf zu nehmen.

2.2 Verwendung der partiellen Rekonfiguration zur Verbesserung von Bildverarbeitungsalgorithmen

In [Raikovich und Feher (2010)] wird die Verwendung der dynamischen PR Technologie bei der Verarbeitung von Bilddaten im medizinischen Bereich vorgestellt. So eigenen sich Algorithmen, die eine große Datenmenge bei serielle zu verarbeiten haben, besonders für den Einsatz im FPGA. Hierunter fallen beispielsweise die Verarbeitung von Bilddaten bei einer zellularen Mikroskopie, da hier eine große Menge an Daten anfällt. Diese erfordern zum einen die menschliche Intelligenz zur qualitativen Kontrolle, zum anderen automatische Werkzeuge zur quantitativen Untersuchung der Bilddaten. Hierzu wurde eine Bilderverarbeitungssoftware entwickelt, welche in der Lage ist, auf Matlab basierende Software-Bildverarbeitungs Pipelines zu erzeugen. Die Ergebnisse dieser Pipelines werden von Biologen untersucht und gegebenenfalls die Parameter angepasst und die verbesserten Pipelines anschließend zur Verarbeitung der Bilddaten verwendet. Das Ziel dieser Arbeit ist es die verbesserten Bildverarbeitungsalgorithmen in einem rekonfigurierbarem Hardware-System einzusetzen.

Die entwickelte Software umfasst Bibliotheken zur Erzeugung von beispielsweise Mittelwerts- oder Median-Filter Stufen. Das Testen der DPR Pipeline erfolgt auf Basis eines Zellen Erkennungs-Algorithmus. Dieser enthält folgende Pipeline Stufen:

1. Median Filter: Das Median Filter dient dem Entfernen von Störungen im Bild. In der hier verwendeten Implementierung, wird der Median in 3 Stufen ermittelt, hierbei sortiert die erste Stufe die jeweiligen Zeilen, die zweite Stufe die Spalten. In der dritten Stufe wird die Diagonale Sortiert, wodurch am Ausgang dieser Stufe der Median des Bildausschnittes anliegt.

2. Berechnung des Histogramms: Das Histogramm wird mit einem 256×23 Bit RAM Berechnet. Dessen Größe setzt sich aus den 8 Bit Grauwerten des Bildes zusammen sowie der maximalen Größe von 2048×2048 des Bildes. Der Speicher wird zu Beginn jedes Bildes auf 0 gesetzt und bei jedem neuen Pixel die Anzahl des als Adresse verwendeten Farbwertes inkrementiert.

3. Errechnen eines Schwellenwertes: Der Schwellwert des Bildes zum Trennen des Hintergrundes vom Vordergrund wird mit der Otsu Methode berechnet [Greensted (2010)]. Dieser berechnet den Schwellwert auf Basis der Varianz der Grauwerte im Bild. Ein Nachteil dieser Methode ist die Verwendung von Division, wodurch die Berechnung dieser Werte nach 16 Clock Zyklen erfolgt.

4. Binarisation: Die Binarisation erfolgt mit dem vorher errechneten Schwellenwert. Jedes Pixel, dessen Intensität niedriger als die des Schwellwertes ist, wird als schwarzes Pixel weiterverarbeitet. Alle Pixel mit einer Intensität über dem Schwellwert werden als weißes Pixel gesehen.

5. Kanten Erkennung: Zur Kantenerkennung wird ein FIR Filter eingesetzt, der die gewichteten Summen der Eingangswerte berechnet. Die jeweiligen Gewichte werden so gewählt, dass ein Hochpass-Filter entsteht. Dies erfolgt durch das Wählen einer positiven Gewichtung für die inneren Werte und einer negativen für die äußeren.

Die einzelnen Stufen werden über FIFO Elemente verbunden, um die unterschiedlichen Laufzeiten innerhalb einer Pipeline Stufe auszugleichen (vgl. Abbildung 5).

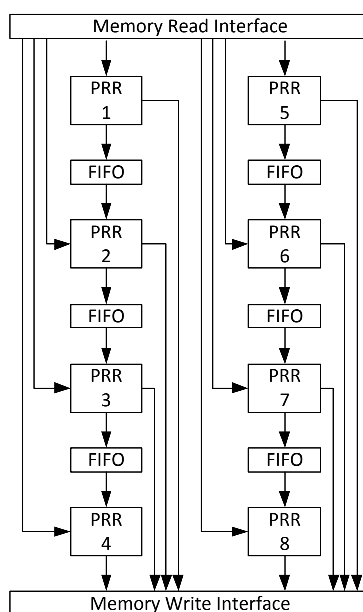


Abb. 5: *Rekonfigurierbare Pipeline zur Bilddatenverarbeitung [Raikovich und Feher (2010)]*

Aufgrund von Testergebnissen, die zeigten, dass das Place-and-Route fehlschlägt, wenn mehr als 75% der Logic-Ressourcen verwendet werden, ist die Größe jeder PRR ist auf 120CLBs, zuzgülich 4 Block-RAMs and 16 DSP48E Slices festgelegt. Diese Größe erlaubt das Anlegen von acht PRMs im Virtex 5, mit Reserve für den statischen Part, der die Ansteuerung eines Speichers sowie die Verbindung zu externen Systemen übernimmt. Eine Vergleichs-Implementierung in Software ergab, dass die Berechnung in Hardware bis zum Faktor 71 schneller ist als auf einem herkömmlichen PC, einzig die Verwendung von Dividieren führt zur einer reduzierung des Datendurchsatzes [2].

Tabelle 2: *Ressourcenbedarf der einzelnen PRMs sowie die Berechnungsdauer im Vergleich zu einem herkömmlichen PC auf einem ML501 Entwicklungsboard mit einem XC5VSX50T FPGA [Raikovich und Feher (2010)]*

PRM	Ressourcenbedarf (%)			Berechnungsdauer (ms)	
	CLB	RAMB16	DSP48E	FPGA mit 125 MHz	PC mit 2,8 GHz
Median	70	25	0	2,1	150
Otsu / Schwellwert	56	25	75	2,15	2
Binarisierung / FIR	43	25	57	2,1	60

Fazit

Die Verarbeitung der aufgenommen Bilddaten erfolgt nachdem diese komplett in den Speicher geladen worden. Diese werden dann ausgelesen und in der Pipeline verarbeitet. Aus diesem System ergibt sich, dass keine Echtzeitverarbeitung der Daten stattfindet und so keine FiFos zwischen den Modulen notwendig erscheinen. Weiter werden die Filter vor der Inbetriebnahme des Systems synthetisiert und es besteht nicht die Notwendingkeit das System bei der Konfiguration weiter laufen zu lassen, so dass hier die PRR einzig zur späteren Anpassung des Systems dient bzw. ein Standardsystem die zu verwendenen Filter je nach Konfigruation lädt.

2.3 Die partielle Rekonfiguration als Unterstützung eines SoC basierten biometrischen Authentifizierungssystems

In System welches zur Bimoterischen Authentifizierung die dynamische partielle Rekonfiguration einsetzt wird in [Fons und Fons (2011)] vorgestellt. Der Einsatz sowie die Akzeptanz solcher Systeme ist stark abhängig von der Bedienbarkeit, durch den Anwender sowie der Trefferquote, der Antwort-Zeit und der Systemkosten. Unter den biometrischen Erkennungs-Systemen ist die Verwendung der Fingerabdrücke das am längsten und meisten verwendete Verfahren. Es wurde zuerst bei der Verbrechensbekämpfung eingesetzt. Inzwischen sind diese Systeme weiter verbreitet und werden beispielsweise bei der Authentifizierung für Laptops oder Zugangskontrollen für Gebäude verwendet, wobei die eindeutige und schnelle Erkennung von Personen weiterhin ein Forschungsgebiet darstellt, da ein Algorithmus, der einen Nutzer ohne Fehler erkennt, nicht existiert.

Ein Weg die aktuellen beschränkungen durch die vorhandenen Algorithmen zu umgehen, ist das Erweitern der Systeme mit verbesserten Algorithmen, die direkt Einfluss auf Rechenleistung oder die Fehlerrate haben. Das Ziel der vorgestellten Arbeit ist es, eine System-Architektur zu entwerfen, welche die Anforderungen an die hohe Rechenleistung sowie eine kleine Fehler Rate erfüllt. Hierzu werden FPGAs mit Hardware-Software Co-Design Techniken sowie der dynamischen partiellen Rekonfiguration verwendet.

Der Prozess zur Authentifizierung erfolgt in zwei Phasen (vgl. Abbildung 6):

- Enrolment: In der Registrierungsphase werden die Fingerabdrücke des Anwenders zum Vergleich genommen. Der Abdruck wird verarbeitet und die extrahierten Merkmale zum gespeichert.
- Authentication: Während einer Authentifizierung erfolgt ein erneutes Nehmen der Fingerabdrücke. Dieser neue Abdruck wird ebenfalls verarbeitet und die Ergebnisse mit den gespeicherten Merkmalen verglichen. Stimmen dieser überein, wird der Zugriff gewährt.

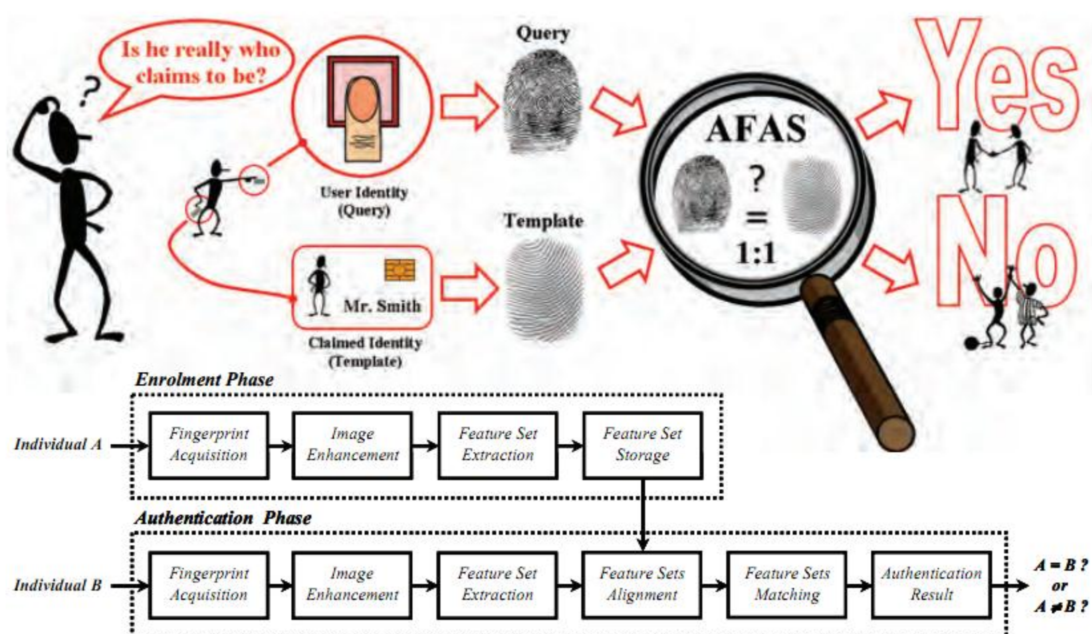


Abb. 6: *Ablauf der Zugriffsgewährung mit der Enrolment Phase zum Speichern der Merkmale sowie der Authentication zum Gewähren des Zugriffs bei einer Anfrage [Fons und Fons (2011)]*

Der dargestellte Ablauf der Registrierung zeigt, dass die Registrierung und die Authentifizierung bis zur Merkmalsextraktion die gleiche Funktionalität besitzen. Das Ziel des Authentifizierungssystems ist es, beide Phasen zu vereinen, so dass das System jede Aufgabe in der Applikation erfüllen kann. Hierzu werden bereits vorhandene Algorithmen eingesetzt und auf die Zielplattform angepasst. Diese beruhen auf dem Erkennen von Minutien durch die Endungen und Verzweigungen der Papillarleisten und sich zur Erkennung eignen, da sie unverändlich und für jeden Menschen einzigartig sind [Wikipedia (2011)].

In Abbildung 7 ist der Verlauf einer Fingerabdruck Analyse dargestellt:

- Bildverbesserung (A-G): In diesem Abschnitt wird zuerst der Abdruck vom Hintergrund getrennt (A) und das Bild normalisiert (B). Noch auftretende Störungen werden anschließend beseitigt (C) und die Richtung der Papillarleisten bestimmt (D). Die Ausrichtung der Papillarleisten wird in einem weiteren Schritt verbessert (E) und das Bild binarisiert (F) bevor es geglättet (G) an die Merkmalsextraktion übergeben wird.
- Merkmalsextraktion (H-I): Die hervorgehobenen Papillarleisten werden so verdünnt, dass nur ein Pixel pro Papillarleisten bleibt, um die Erkennung von Minutien zu vereinfachen (H). Im nächsten Schritt werden die Hauptmerkmale wie endende oder sich spaltende Papillarleisten markiert (I) und zusammen mit der Richtung der Papillarleisten als Merkmal genutzt.
- Merkmalsvergleich (J): Die beiden Bilder werden auf ähnliche Regionen untersucht und bei einer Übereinstimmung wird der Fokus zur weiteren Untersuchung auf diesen Bereich gelegt.
- Ergebnis (K): Stimmen die zu vergleichenden Bereiche überein, wird ein positives Ergebnis geliefert und der Zugriff gewährt.

Abb. 7: *Ergebnisse der einzelnen Bildverarbeitungsschritte zur Merkmalserkennung der gespeicherten Bildes (blau) und der Anfrage (rot) zum Merkmalsvergleich [Fons und Fons (2011)]*

Die implementierten Algorithmen werden mit Abdrücken getestet, die von der Fingerprint Verification Competition (FCV2004) zur Verfügung gestellt werden. Die Datenbank des FCV2004 enthält 110 verschiedene Finger, von denen jeweils acht Abdrücke genommen sind, so dass

insgesamt 880 Abdrücke zur Verfügung stehen. Diese Abdrücke wurden in zwei Gruppen unterteilt, wobei die erste Gruppe mit zehn Fingern zur Kalibierung des Systems und die übrigen 800 Abdrücke zum Testen genutzt werden. Auf Basis dieser Abdrücke erfolgte der Test der Systeme nach den Kriterien der FCV2004:

- „False Acceptance Rate“: Die Berechnung der Fehler Akzeptanz Rate erfolgte durch das Testen eines Abdruckes jedes Fingers gegen die übrigen Beispielabdrücke. Dies erfolgt, um zu überprüfen wieviele nicht berechtigte Zugriffe gewährt werden.
- „False Rejection Rate“: Zur Berechnung der Fehler Ablehnungs Rate wird ein Abdruck des Fingers gegen die restlichen sieben Abdrücke des Fingers geprüft. Wobei jede Ablehnung der getesteten Abdrücke ein Fehler des Systems ist.

Nach der Initialisierung des Systems wurde eine Fehler Rate von 4% ermittelt. Bei einer Teilnahme des Systems an dem FCV2004 Wettbewerb wäre mit der erreichten Fehlerquote ein fünfter Platz erreicht worden.

Zum Testen der Leistungsfähigkeit des Algorithmus wurde dieser auf einem Standard-PC und Embedded Hard- und Softcore Prozessoren getestet (vgl. Tabelle 3). Die Ergebnisse zeigen, dass in einer reinen Softwareumgebung einzig der Standard-PC die Anforderungen an ein Echtzeitsystem erfüllt. Die Embedded Prozessoren erfüllen diese Anforderungen durch ihre geringe Frequenz nicht. Die starke Abweichung der Altera Excalibur Plattform wird dabei durch die ausgeschalteten Daten-Caches erklärt, welches zur einer Verlangsamung bei der Ausführung des Programms führt.

Tabelle 3: *Eigenschaften der verwendeten Systeme zum Test der Software Implementierung des Analyse Algorithmusses sowie der Verarbeitungsdauer zu Registrierung und Authentifizierung [Fons und Fons (2011)]*

	Standard PC Plattform	Embedded System Plattform 1	Embedded System Plattform 2	Embedded System Plattform 3
Plattform	Acer Aspire 9420	Altera Excalibur EPXA10	Xilinx Spartan 3AN	Xilinx Virtex 4 ML401
Familie	MPU Intel Core 2 Duo	SiPC EPXA10F1020C1	FPGA XC3S700AN	FPGA XC4VLX25
Prozessor	Intel Core 2 DUO T5600	ARM 922T	MicroBlaze	MicroBlaze
Takt	1,83 GHz	200 MHz	66,667 MHz	100 MHz
Registrierung	40.790 ms	14598,674 ms	3240,934 ms	2256,663 ms
Authentifizierung	3274,38 ms	295748,055 ms	213962,035 ms	140767,219 ms

Das Hauptthema der vorgestellten Arbeit befasst sich mit dem Erstellen eines FPGA basierten eingebetteten Systems, welches die Möglichkeit zur dynamischen partiellen Rekonfiguration besitzt. Der Aufbau dieses Systems erfolgte auf Basis von fünf Design Aspekten:

1. General-Purpose Mikroprozessor: Der Einsatz von günstigen Prozessoren, die eine mittlere Leistung besitzen, stellt die Grundlage vieler eingebetteter Systeme. Diese besitzen die Eigenschaft, dass viele Bibliotheken zur Verfügung stehen und Software eine geringe Entwicklungsdauer hat. Ein Nachteil der CPUs ist die langsame Datenverarbeitung, da meist mehrere Takte für eine Operation gebraucht werden.

2. Programmierbare Logik: Reicht Software allein nicht aus, um die gegebenen maximalen Zeiten einzuhalten, werden Hardwarebeschleuniger eingesetzt. Hier sind FPGAs und CPLDs eine sehr flexible Lösung im Gegensatz zu ASICs. Diese sind inzwischen in der Lage sehr komplexe Systeme zu beinhalten und besitzen oft spezielle interne Module, welche die Ausführung der Anwendung weiter verbessern. Beispiele für diese Module sind DSP Elemente oder Speicher Blöcke.

3. Hardware/Software Co-Design: Die Aspekte des Hardware/Software Co-Design vereinen die Vorteile der Software und der Hardwarebeschleuniger, wie Flexibilität der Software und den hohen Datendurchsatz der Hardware.

4. Partiell rekonfigurierbare FPGAs: Komplexe Systeme haben einen hohen Bedarf an FPGA internen Ressourcen und stoßen, vorallem bei kleinen FPGAs, oft an die Grenzen der Auslastung. Die Verwendung von größeren oder mehreren FPGAs kommt aus Kostengründen oft nicht in Frage, so dass das System eine Möglichkeit zur Wiederverwendung der Ressourcen bieten muss.

5. „System-on-programmable Chip“ (SoPC): Der Einsatz von programmierbarer Logik mit einem internen Mikroprozessor verringert die Kosten und den Energieverbrauch der eingebetteten Systeme. Die Ingetration weitere Elemente in einen Chip verringert diese weiter. So werden beispielsweise Speichercontroller und Timer auf einem Chip zusammengefasst. Zusätzlich bietet dieses Schutz gegen Angreifer, da keine Daten über freie Leitungen übertragen werden.

Die Implementierung des Systems zur Fingerabdruck Analyse erfolgt dabei auf einem ML401 Board mit einem Virtex 4 FPGA [Xilinx (2006a)]. Der Fingerabdruck-Sensor ist an die I/O-Pins des Entwicklungsboard angeschlossen und die Kommunikation mit einem PC erfolgt über ein RS-232 Interface (vgl. Abbildung 8). Aufgeteilt ist das System in ein dynamisches und ein statisches Subsystem. Das statische System umfasst den MicroBlaze Mikrocontroller sowie die Peripherie Elemente, der dynamische Bereich (PRR) die Bildverarbeitungsfilter für die Fingerabdruck Erkennung. Im Virtex 4 sind nur flüchtige Speicher enthalten, so dass zum Speichern der Konfigruation der externe Plattform-Flash-Speicher verwendet wird. Der ebenfalls verfügbare Linear-Flash enthält die acht Dateien zur Rekonfiguration (PRM), welche den Filterstufen in Abbildung 7 entsprechen, sowie den Programm-Code und die vergleichsmuster zur Authentifizerung. Während der Einschalt-Phase des Systems lädt der, in der Konfiguration enthaltene Bootloader das auszuführende Program aus dem Linear-Flash und schreibt es in den SDRAM Speicher des ML401. Nach dem Kopieren wird die Start-Adresse des SDRAM-Speichers aufgerufen und das Programm zur Analyse gestartet. Die Kommunikation zwischen der PRR und dem statischen System erfolgt über Bus-Makros, welche bei der Rekonfiguration die Übertragung von Pseudzuständen verhindern, wobei die Kommunikation zwischen den Bereichen, während der Rekonfiguration, unterbrochen wird. Um eine geringe Rekonfigruationszeit zu erhalten ist die PRR so ausgelegt, dass gerade ausreichend Ressourcen für das Modul mit dem größten Bedarf bereit stehen. Gestartet wird die Rekonfiguration über den MicroBlaze, welche dem „Internal Configruation Access Point“ (ICAP) das zu konfigurierende Modul mitteilt. Das ICAP liest die Daten direkt aus dem Speicher, so dass der MicroBlaze keinen Datentransfer zu vollziehen hat. Der Abschluss der Rekonfiguration wird dem Task über ein Interrupt mitgeteilt.

Tabelle 4 zeigt die Ergebnisse der Messung mit der Software-Only Implementierung und einer mit Hardware-Beschleuniger. Die Software Lösung wird auf dem rekonfigurierbaren System etwas verlangsamt ausgeführt, da die Instruktions-Puffer von 32 kByte auf 8 kByte reduziert

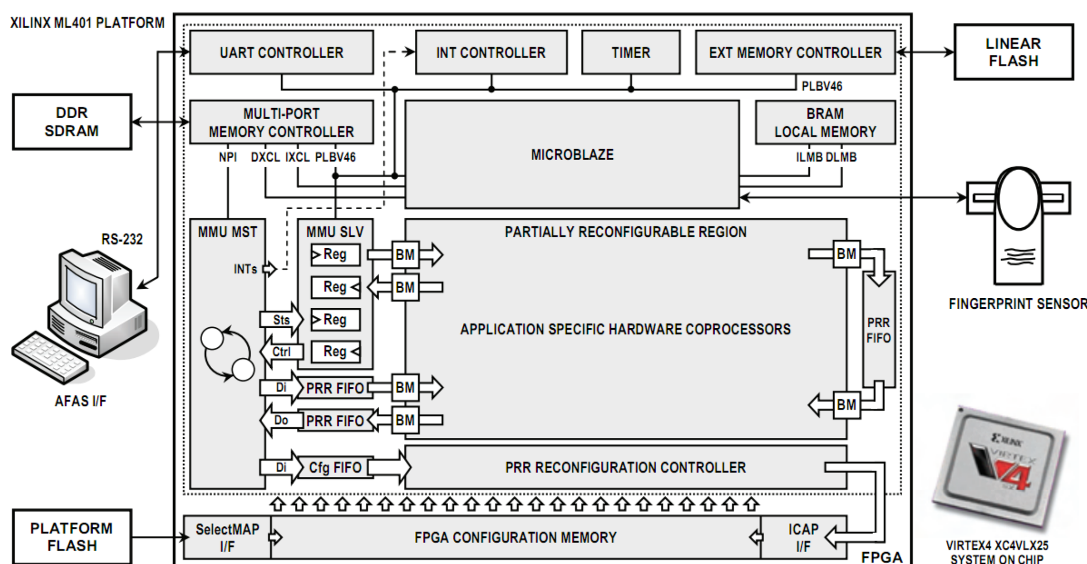


Abb. 8: *Fingerabdruck Authentifizierungssystem basierend auf einer SoC Plattform mit der Fähigkeit zur dynamischen partiellen Rekonfigruation [Fons und Fons (2011)]*

wurden. Die Implementierung mit Hardware-Beschleuniger Modulen reduziert die Verarbeitungsdauer der Authentifizierung um 70%, wobei die benötigte Zeit bereits die Rekonfiguration enthält.

Tabelle 4: *Verarbeitungsdauer der Registriegung und Authentifizierung im rekonfigurierbaren System in einer Software-Only Implementierung und mit Unterstützung durch Hardware-Beschleunigern (vgl. Abbildung 7). [Fons und Fons (2011)]*

Aufgabe	Software	HW/SW
Registrierung	2433,543 ms	21,932 ms
Authentifizierung	142693,451 ms	205.025 ms

Fazit

Das SoC-System zum Abgleich der Fingerabdrücke enthält eine PRR, in die die jeweiligen Filter nacheinander geladen werden. Die Rekonfiguration erfolgt dabei nachdem der Filter das komplette Bild verarbeitet hat. In dieser Zeit steht das restliche System still, da keine weiteren Aufgaben für das System vorgesehen sind. Bei einer Implementierung auf einem größeren FPGA, beispielsweise dem Virtex6, könnte die Anzahl der PRRs erhöht werden ([Xilinx (2010g)],[Xilinx (2010f)]). So ist das System in der Lage den nachfolgenden Filter zu konfigurieren, während die vorherige Stufe bereits Daten verarbeitet um so die Gesamt-Verarbeitungsdauer weiter zu verkürzen. An der HAW-Hamburg wird in einer Firmen kooperation ein System zur Fingerabdrucks basierten Authentifizierung entwickelt. Dieses wird in der ersten Phase nur eine Vorverarbeitung der Daten vollziehen, um die Authentifizierungszeit zu verkürzen, aber gleichzeitig einen kostengünstigen FPGA einsetzen zu können.

2.4 Simulation einer Partiellen Rekonfiguration in einer mobilen Agenten Umgebung

Der Artikel [Kearney und Jasiunas (2006)] befasst sich mit der Verteilung von IP-Modulen innerhalb eines Schwarms von Drohnen oder der Portierung eines Drohnen-Zustandes auf eine andere, wenn diese ausgetauscht werden. Hierbei wird insbesondere auf kleine „Unpiloted Airborne Vehicles“ (Drohnen, UAV) eingegangen, die in der Ausstattung und der Energieversorgung sehr eingeschränkt sind.

Der Einsatz von Drohnen wird in den nächsten 20 Jahren einen Großteil der Flugzeug-Industrie beschäftigen und der Forschungsetat wird in den Jahren 2003-2013 auf 5,5 Milliarden Euro geschätzt [SpaceDaily (2004)]. Die Datenverarbeitung von Kameras und kleinen RF-Sensoren ist in den meisten Fällen die Aufgabe dieser Drohnen. Die Verarbeitung der Daten erfolgt oft in FPGAs, da diese die Daten parallel und mit einem determiniertem Zeitverhalten abarbeiten, was mit einer CPU, auf der auch ein OS beziehungsweise kritische Applikationen laufen, nicht möglich ist.

Damit eine größere Genauigkeit der Daten erreicht wird, sind mehrere Drohnen gleichzeitig zu verwenden, so wird z.B. der Fehler einer Ortung um mehr als 80% verringert. Die Verwendung eines Drohnen Schwarms wird in dem Artikel in zwei Gruppen unterschieden.

- „Single Mission“:
  Es sind für die Mission N Aufgaben zu erfüllen und es stehen N Drohnen zur Verfügung. Hierbei wird angenommen, das die Aufgaben so verteilt sind, dass die Drohnen gleichzeitig zur Basis zurückkehren. Hierbei ist es das Ziel, die Zeit der Ausführung zu maximieren.
- „Continuous Mission“:
  In diesem Szenario existieren mindestens $N + 1$ Drohnen für N Aufgaben. Hierdurch können Drohnen mit geringem Energievorrat durch eine voll aufgeladene Drohne ersetzt werden. Bei Aufgaben, die die Drohnen in unterschiedliche Zustände versetzen, sind die Zustände auf die übernehmende Drohne zu übertragen, so dass eine Mobilität der einzelnen Applikationen vorauszusetzen ist.

Diese Mobilität wird z.B. über das Agenten Paradigma erreicht [Wooldridge und Jennings (1995)]. Insbesondere die mobilen Agenten sind in der Lage während des Betriebes zwischen Host-Systemen zu migrieren, wobei jeweils eine spezielle Umgebung zur Ausführung der Agenten verwendet wird. Diese Umgebung stellt hier das OS welches zusätzlich die Verwaltung der Hardware (HW) Ressourcen im FPGA zu übernehmen hat und die IP-Module der Agenten an die freien Ressourcen des FPGAs anzupassen. Dies ist auf Grund, der aktuellen ausprägung der FPGAs, beispielsweise durch festgelegte Bereiche zur PR, jedoch noch nicht realisierbar. Aus diesem Grund wurde die Partielle Rekonfiguration simuliert, welches mit Hilfe eines „Checkpointing“ Verfahrens implementiert wird.

Bei der simulierten Partiellen Rekonfiguration wird jeweils der komplette FPGA mit der neuen Konfiguration überschrieben, welche vorher synthetisiert wurde und als Bitstream bereit liegt. Hierbei gehen alle Zustände des Systems verloren. Um dies zu verhindern, wird das „Checkpointing“ eingesetzt. Dieses sichert die Zustände der einzelnen IP-Module und der Software und wird in zwei unterschiedlichen Varianten eingesetzt.

- „Cooperative“:
  Bei dieser Variante teilt das OS den Tasks mit, dass eine Rekonfiguration nötig ist und diese Sichern darauf hin die eigenen Zustände. Ein Vorteil dieses Verfahrens ist, dass

während des normalen Betriebes keine Rechenzeit für die Sicherung der Zustände verwendet wird. Nachteilig ist, dass nach der Nachricht und der Sicherung aller Zustände eine lange Zeitspanne verstreicht, in welcher die Zustände erst gesichert werden.

- „Preemptive“:
  Die einzelnen Tasks sichern periodisch ihre Zustände, so dass sie jeweils bei dem letzten „Checkpoint“ nach einer Rekonfiguration wieder starten. Der Vorteil dieser Variante liegt darin, dass eine Rekonfiguration durchgeführt werden kann, sobald diese angefordert wird. Nachteilig wirkt sich die zur Sicherung benötigte Rechenzeit auf die Effektivität aus, da immer wieder eine Sicherung durchgeführt wird, welche nicht immer gebraucht wird.

In dem vorgestellten Projekt wird das „Preemptive Checkpointing“ eingesetzt. Dazu wurden die Applikationen in logische Blöcke unterteilt, welche als atomare Einheiten behandelt werden. Diese atomaren Einheiten werden dann als Zustände in einem Automaten betrachtet und die Variablen bei jedem Zustandsübergang gesichert, so dass jederzeit eine Rekonfiguration stattfinden kann.

Zusätzlich zu der Kommunikation zwischen den einzelnen Drohnen und der Migration der Agenten behandelt das Projekt die Verteilung von FPGA externen Ressourcen wie Speicher. Um diese an die einzelnen Module zu verteilen, sind ein Netzwerk, ein Arbiter und eine Strategie zur Zuteilung der Ressourcen entscheidend (vgl. Abbildung 9).

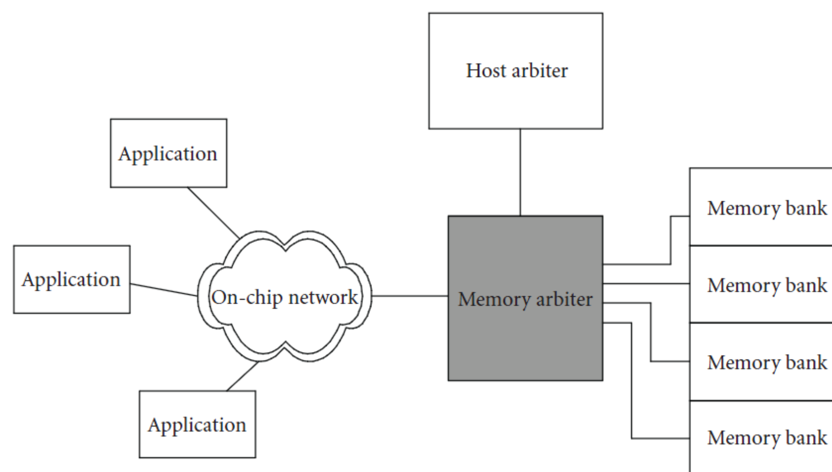


Abb. 9: *SoC Netzwerkarchitektur bestehend aus dem Netzwerk mit den HW-Module sowie dem Memory-Arbiter. Der Host-Arbiter wird durch einen GPP implementiert. [Kearney und Jasiunas (2006)]*

Bei der Wahl der Netzwerk Architektur wurde auf folgende Punkte besonderer Wert gelegt.

- Implementierung:
  Wie einfach lässt sich die jeweilige Architektur implementieren?
- Routing Kosten:
  Wieviele Pfade sind notwendig, um eine neue Applikation an das vorhandene System anzukoppeln?
- Nebenlaufigkeit:
  Wie weit wird Nebenläufigkeit unterstützt?

- Latenz:
  Wie verhält sich die Latenz bei verschiedenen Größen des Netzwerks?
- Skalierbarkeit:
  Wie skaliert das Netzwerk bei einer großen Anzahl an Applikationen?

Aufgrund der hohen Anforderung von Drohnen an die Nebenläufigkeit sowie die Latenz wird hier die „Star“ Topologie verwendet (vgl. Tabelle 5). Die geringe Skalierbarkeit ist vernachlässigbar, da nur eine kleine Anzahl an Modulen auszutauschen ist. Die Verteilung des zur Verfügung stehen Speichers erfolgt statisch, d.h. jede Applikation bekommt einen definierten Speicherbereich und es ist somit keine dynamische Strategie erforderlich. Der Arbiter selbst ist nach dem Round-Robin Prinzip aufgebaut und verbindet Applikationen mit dem Speicher sowie anderen HW-Ressourcen.

Tabelle 5: *Eigenschaften von Netzwerk Topologien in der Implementierung. (+ = Gut, - = schlecht, +/- = neutral) [Kearney und Jasiunas (2006)]*

	Einfachheit der Implementation	Routing Kosten	Nebenläufigkeit	Latenz	Skalierbarkeit
Bus	++	−	−−	−	−−
Star	++	−−	++	++	−−
Mesh	−−	−−	+	+	+
Ring	++	+	−	+/−	+/−
Tree	+/−	++	+/−	+	+
Fat tree	−	−	+	+	+

Bei der Verteilung der Applikationen zwischen den Drohnen wird das Agenten Paradigma verwendet. Unterschieden wird zwischen statischen Agenten, die z.B. eine Ressource wie eine Kamera darstellen, und mobilen Agenten, welche die austauschbaren Applikationen repräsentieren. Anders als andere Paradigmen erlauben es Agenten, dass nicht nur die Ausführung zwischen den Host-Plattformen wechselt, sondern auch der jeweilige Zustand, wobei für jeden Agenten spezielle Regeln festgelegt werden können. Die einzelnen Agenten sind hier über ein einzigartiges Tupel im jeweiligen Netzwerk adressierbar. Dieses Tupel besteht aus:

- „sequence number“ und „home node“ : eine einzigartige Identifikationsnummer und dem eigentlichen Heimatknoten
- „class“ : der Typ des Agenten
- „current node“ : der Knoten auf dem der Agent aktuell ausgeführt wird
- „ability list“ : eine Liste an Fähigkeiten, die der Agent besitzt

Damit die einzelnen Agenten zwischen den Drohnen migrieren können, stellt die Netzwerkumgebung folgende Dienste bereit:

- Finden von anderen Agenten
- Kommunikation mit anderen Agenten
- Abfragen von Informationen über anderen Drohnen
- Migration der Agenten zwischen den Drohnen

Damit die Agenten die anderen adressieren können, erstellt jede Drohne periodisch einen Snapshot mit ihren aktuellen Agenten. Dieser wird anschließend den anderen Drohnen mitgeteilt.

Die Migration der Agenten ist ein teurer Prozess, im Bezug auf Energie sowie Durchsatz um die Downtime des Agenten gering zu halten. Daher benötigt der Agent vorher Informationen über das Zielsystem wie freie FPGA Ressourcen oder den aktuellen Energieverbrauch. Sind diese Information vorhanden und der Agenten möchte migrieren, läuft dieser Prozess wie folgt ab:

1. Der Agent schickt eine Anfrage an die Ziel-Drohne.
2. Die Ziel Drohne erstellt eine Instanz des Agenten im „Sleep Mode“.
3. Die neue Instanz bekommt eine temprorära Adresse, welche der anfragenden Instanz zugeschickt wird.
4. Der Agent stoppt seine Berechnungen und schickt die Daten des erreichten Checkpoints an das Ziel
5. Die neue Instanz übernimmt die Zustände der alten und beginnt die Arbeit, während die alte Instanz abgeschaltet wird.

Fazit

Die Autoren dieses Projektes haben sich entschieden, die eigentliche PR durch ein statisches Verfahren zu simulieren, in dem jede Belegung der FPGAs synthetisiert wurde. Findet dann eine Rekonfiguration statt, so wird der komplette FPGA rekonfiguriert. Dieses Verfahren ist für die Praxis ungeeignet, da jeweils die Zustände aller Module gesichert werden müssen und auch sicherheitskritische Module, die auf dem FPGA laufen, kurzzeitig abgeschaltet und neu gestartet werden.

Vermieden werden könnte dies durch das Einsetzen der PR-Technologie. Durch die festgelegten Bereiche wäre zwar ein Verlust der Flexibilität der Module, welche während der Synthese auf einen Bereich fest zu legen sind und ein zusätzlicher Verbrauch der FPGA Ressourcen in Kauf zu nehmen, dies wird aber durch die Schnelligkeit der Rekonfiguration sowie die Praxistauglichkeit wieder ausgeglichen.

3 Technologieübersicht zur Realisierung des SoC-Clienten mit einem Xilinx FPGA

Die Berechnungen für die FPGA-basierte Distributed Computing Plattform erfolgt in einem „System on Chip“ (SoC) mit dynamischem Bereich. Zum Erstellen dieses SoCs werden verschiedene Software-Tools und „Intellectual Property“ (IP) Module verwendet. Diese werden in den folgenden Abschnitten erläutert.

3.1 ML605 Entwicklungsboard mit Virtex 6 FPGA

Die Durchführung dieses Projektes, erfolgt mit dem ML605 Entwicklungsboard [Xilinx (2011j)]. Dieses enthält einen Virtex6 XC6VLX240T-1FFG1156 FPGA aus der LXT Reihe, welche in ihrer Anzahl an Logik Blöcken und IO-Ports im mittleren Feld der Virtex6 Familie liegen [Xilinx (2010g)]. Als Speicher bietet das Board einen 512 MB DDR3 Speicher, welcher einen Datendurchsatz von ca. 1,6 GB/s aufweist, sowie einen 32 MB BPI Linear Flash, der auch zur Konfiguration des FPGA verwendet werden kann. Als Peripherie stehen dem Anwender beispielsweise ein Tri-Speed Ethernet Port, ein USB zu UART Brücke, sowie eine PCI-Express Schnittstelle zur Verfügung (vgl. Abbildung 10). Der Virtex 6 FPGA stellt dem Anwender folgende Attributed zur Verfügung [Xilinx (2010a)]:

- 37.680 Slices, welche aus je vier LUTs sowie acht FlipFlops bestehen
- Maximal 720 IO-Ports
- 3.650 kB an Distributed RAM
- 768 DSP48E1 Elemente stehen zur Verfügung, die aus jeweils einem 25 Bit x 18 Bit Multiplizierer, einem Addierer und einem Akkumulator bestehen.
- 416 Block RAM mit einer Tiefe von 32 kByte oder 832 Block Ram mit einer Tiefe von 18 kByte mit einer Gesamtgröße von 14.976 kByte

3.2 MicroBlaze MicroProzessor zur Steuerung des Systems

Der MicroBlaze ist ein Mikroprozessor mit einer 32-bit RISC Harvard Architektur und steht für Xilinx FPGAs als Softcore bereit. Bei der Konfiguration des MicroBlazes steht dem Anwender eine Vielzahl an Konfigurationsmöglichkeiten zur Verfügung. So steht beispielsweise eine 5-stufige Pipeline für einen hohen Datendurchsatz oder eine 3-stufige für einen geringeren Ressourcenbedarf zur Auswahl. Weitere Konfigurationen sind in Abbildung 11 dargestellt [Xilinx (2011g)].

Zur Konfiguration des MicroBlaze stehen zum einen vordefinierte Konfigurationen bereit, zum anderen ist eine exakte Anpassung an die Bedürfnisse des Anwenders möglich. Beispielsweise stehen folgende Konfigurationen dem Anwender zur Verfügung :

- Minimum Area: Diese Konfiguration dient dem Einatz in kleinen FPGAs. Dem Anwender stehen keine Caches und Interrupts zur Verfügung, ebenso wird das Debug-Modul deaktiviert. Der MicroBlaze selber wird in dieser Konfiguration als 3-stufige Pipeline implementiert, wodurch der Datendurchsatz der CPU auf ein Minimum reduziert wird. Der Ressourcenbedarf des MicroBlaze wird in dieser Konfiguration auf Slices beschränkt, so dass kein Bedarf an BRAM und DSP Elementen besteht.

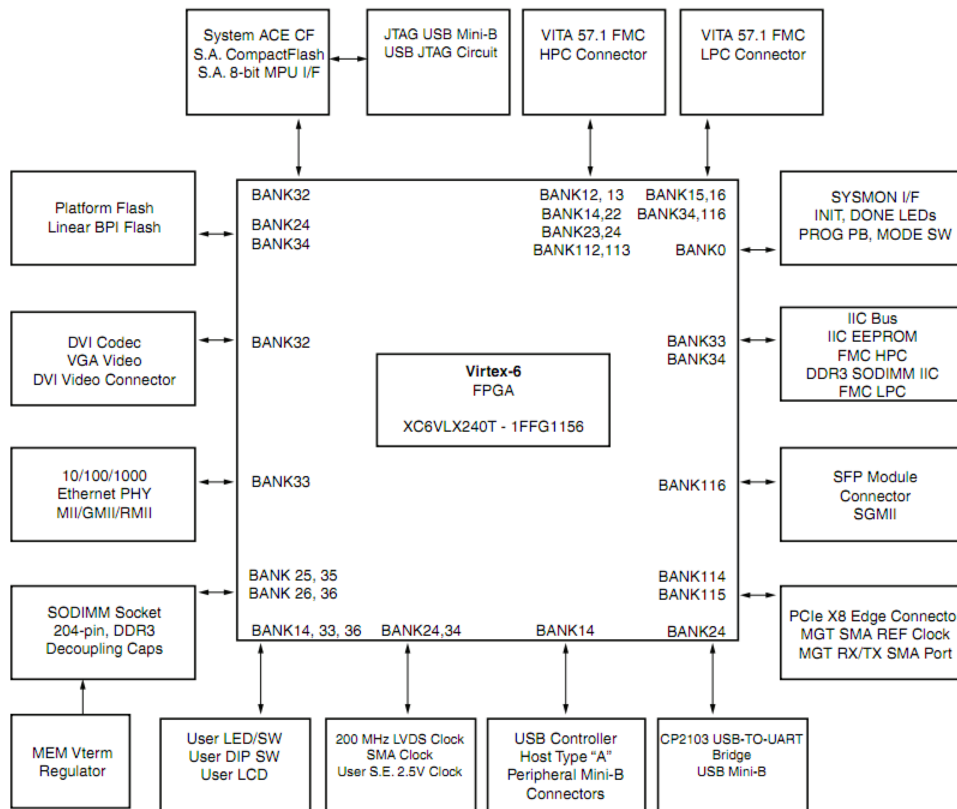


Abb. 10: *Blockschaltbild des ML605 Entwicklungsboard mit den verwendeten Anschlüssen der Peripherie im FPGA [Xilinx (2011h)]*

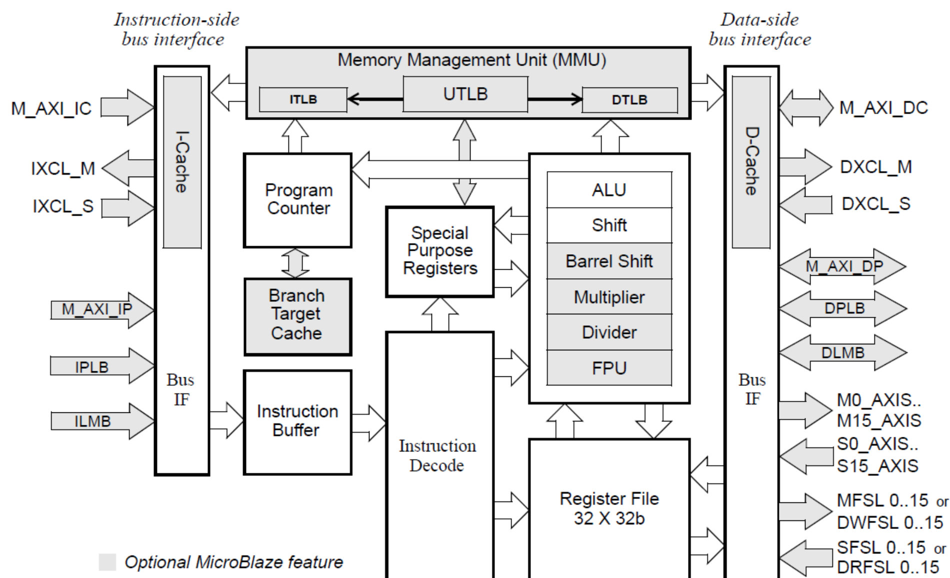


Abb. 11: *Standardmäßig vorhandene (weiß) und optionale (grau) Komponenten des MicroBlaze Mikroprozessors [Xilinx (2011g)]*

- Maximum Perfomance: In der Maximum Perfomance Konfiguration wird der MicroBlaze auf einen hohen Datendurchsatz ausgelegt. Dies erfolgt durch das Aktivieren von Daten und Instruktions Caches sowie dem Abschalten der Interrupt Logik. Zur weiteren Optimierung werden die Hardware Multiplizierer und Dividierer aktiviert. Zur Implementierung dieser Konfiguration werden 37 BRAM und 6 DSP Elemente verwendet, was in

kleineren FPGAs, wie dem Spartan-3A zu einer kompletten Auslastung der Ressourcen führt.

- Maximum Frequenzy: In dieser Konfiguration werden alle optionale Module deaktiviert und die 3-stufige Pipeline des MicroBlaze verwendet. Dies erfolgt um die höchste Frequenz zu verwenden. Diese Konfiguration ist am besten für kleine FPGAs geeignet, die einen hohen Datendurchsatz zu erreichen haben, aber keine Ressourcen für die Maximum Perfomance Konfiguration besitzen.

3.3 Erstellen eines DPR-Systems und Verwaltung von FPGA Konfiguration mit PlanAhead

Das Erstellen eines DPR-Systems erfolgt auf Basis des in Abbildung 12 dargestellten Ablaufs [Xilinx (2010d)]:

1. Spezifikation: In der Spezifikationsphase sind die Daten für das Gesamtsystem festzulegen. Hierunter fallen die Schnittstellen zur Peripheriegeräten und das Timingverhalten des Systems. So sind beispielsweise maximale Reaktionszeiten für Echtzeitsysteme zu definieren.
2. Partitionierung: Während der Partitionierung wird entschieden, welche Komponenten als dynamisch und welche statisch zu implementieren sind. Dies erfolgt unter anderem durch die Verfügbarkeit der Module. So sind die Module statisch zu implementieren, die im laufenden Betrieb durchgängig verfügbar sein müssen.
3. Schnittstellen: Die Schnittstellen zwischen dem statischem und dem dynamischem System sind so zu entwerfen, dass alle Konfigrationenen dies verwenden können. Sind unterschiedliche Steuersignale zu implementieren, so ist es ratsam deren Dekodierung im dynamischen System zu implementieren und ggf. die PRR zu vergrößern. Dies hat den Vorteil, dass keine Kenntnisse der PRM im statischen System erforderlich sind und bei einer Änderung der PRM keine Anpassung zu erfolgen hat.
4. Entwicklung und Simulation: Die Entwicklung erfolg auf Basis der Spezifikation, getrennt in den statischen und den dynamischen Bereich. Die jeweiligen funktionalen Simulationen sind ebenfalls seperat zu vollziehen und erst bei einer gegebenen Funktionalität die Teil-Systeme zu verbinden, um eine bessere Übersicht zu erhalten und die Anzahl der Fehlerquellen zu verringern.
5. Implementierung und Test: Die Implementierung erfolgt auf Basis der simulierten Teil-Module mit der PlanAhead Software.

Das Tool Plan Ahead dient dem Implementieren von Netzlisten und unterstützt die Analyse der Timing Pfade und bietet eine grafische Oberfläche zum Setzen der Module innerhalb des FPGAs. Bei der Implementierung werden unterschiedliche Strategien zum Konfigurieren erstellt, wie dem Setzen von Optionen bei der Implementierung oder dem Laden verschiedener BMM Files, welche zur Initialisierung von Block-Ram genutzt werden. Zur Implementierung von PR-Projekten werden für jede PRM eigene „Runs“ erstellt, welche die zu verwendende Netzliste enthält. Nach der Implementierung des ersten Bitfiles wird das statische System den anderen „Runs“ zur Verfügung gestellt, wodurch anschließend das dynamische System mit den Verbindungen des statischen Systems implementiert wird. Beispielsweise sind die folgenden

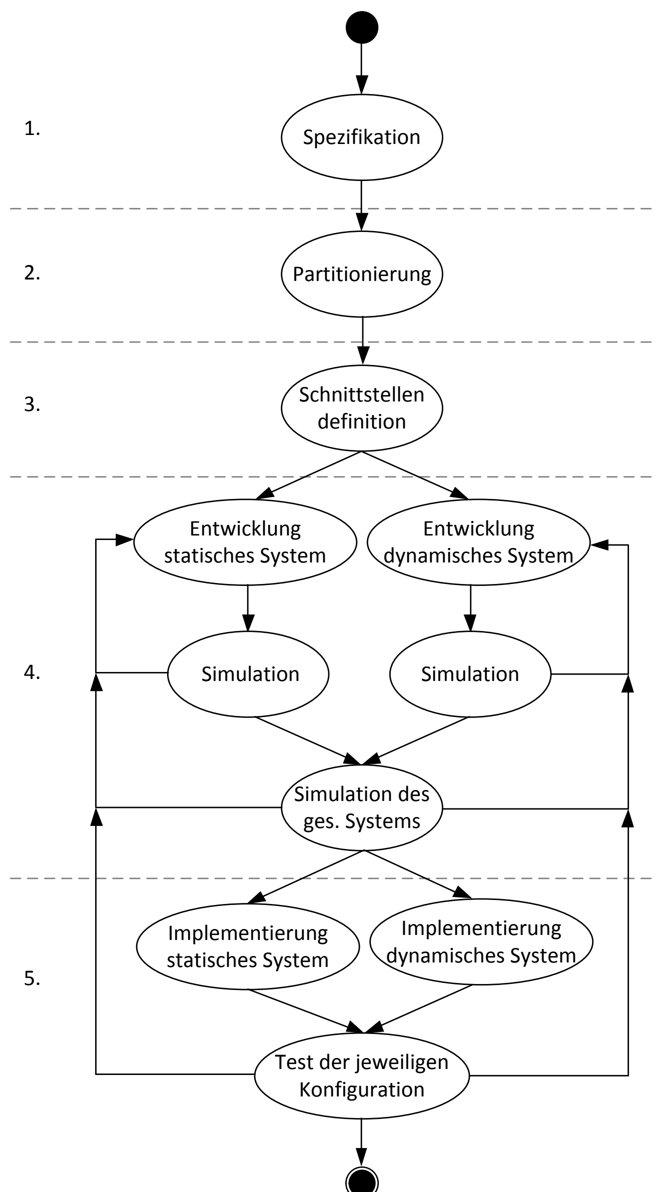


Abb. 12: *Entwurfsablauf zum Erstellen eines dynamisch partiell rekonfigurierbaren Systems*

„Runs" für ein Bildverarbeitungssystem, mit einer Binarisierung und einem Median-Filter, zu erstellen:

1. Statisches System mit Binarisierung:
   - Static_Logic: Action: Implement
     Das statische System wird in der Konfiguration erstellt und in den übrigen „Runs" verwendet.
   - PRR: Configuration: Binär-Filter, Action: Implement
     Hier wird direkt die Binarisierung Implementiert ohne eine Black-Box zu erstellen. Dies erfolgt nach dem Erstellen des statischen Systems, so dass die verwendeten Signale bereits festgelegt sind.
2. Median-Filter:
   - Static_Logic: Action: Import
     Das statische System wird aus dem vorherigen „Run" verwendet.

- PRR: Configuration: Median-Filter, Action: Implement
  Der Filter wird auf Basis des bereits erstellen statischen Systems implementiert und an die definierten Schnittstellen angepasst.

3.4 Tri-Mode Ethernet Media Access Controller

Der Tri-Mode Ethernet Media Access Controller (XPS LL TEMAC) ist ein IP zur Anbindung des SoCs an ein Ethernet Netzwerk, welcher die Übertragungsraten 10, 100, und 1000 Mb/S unterstützt. Das IP ist in einer Hard- und Softcore-Variante verfügbar. Untersützt werden die Hardcores vom Virtex-6, Virtex-5 FXT, LXT, SXT und der Virtex-4 FXT (vgl. Abbildung 13). Das IP arbeitet im Full Duplex Modus und untersützt, sowohl die Berechnung von TCP- und UDP- Checksums in Hardware, als auch VLAN-Frames. Die Anbindung an den MicroBlaze erfolgt über ein PLB Interface wobei für RX und TX jeweils 2 - 32 kBit FiFos verwendet werden. Die in diesen FiFos gespeicherten Daten werden anschließend über das *XPS_ LL_ FIFO* der Software zur Verfügung gestellt. Bei der Verwendung eines „Multi Port Memory Controllers“ (MPMC) kann anstelle des *XPS_ LL_ FIFO* eine SDMA Schnittstelle verwendet werden, welche die empfangenen Daten direkt in den Speicher schreibt (vgl. Abbildung 14). Verwendet wird der in den Virtex 6 integrierte V6-TEMAC, welcher mit dem auf dem ML605 verbautem Marvell Alaska PHY device (88E1111) kommuniziert [Xilinx (2010b)][Xilinx (2011h)].

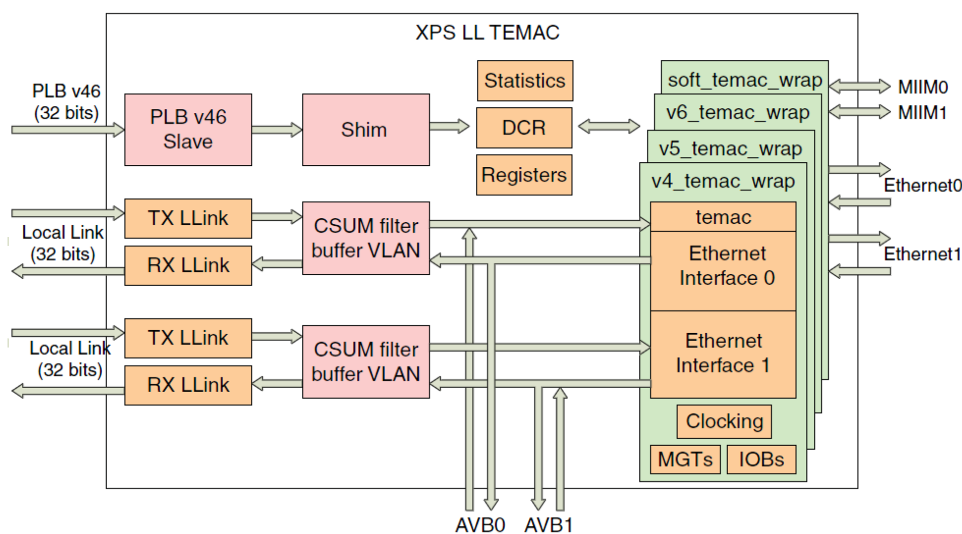


Abb. 13: *XPS_LL_TEMAC IP mit Registern für die Ethernet Hard- bzw. Soft-IPs [Xilinx (2010b)]*

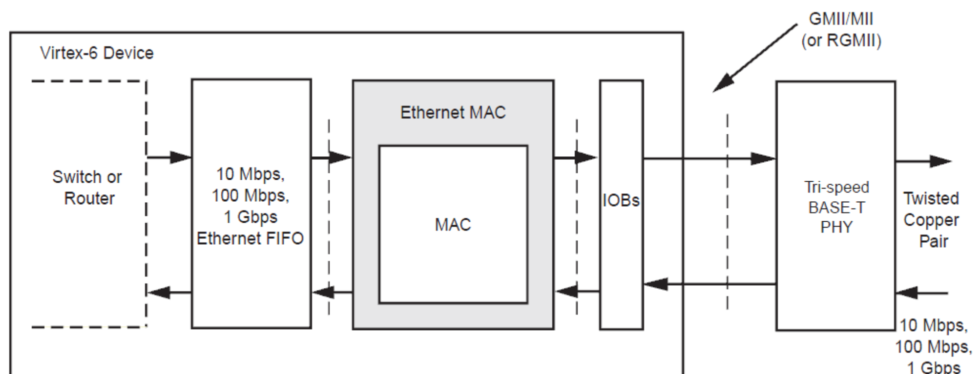


Abb. 14: *Datenpfad des TEMAC IPs [Xilinx (2011i)]*

3.5 Hardware Internal Configuration Access Point (HWICAP)

Der „Hardware Internal Configuration Access Point“ (HWICAP) ist die FPGA interne Schnittstelle um den FPGA Konfigurationsspeicher zu lesen und zu beschreiben. Durch die Anbindung des HWICAP sind Teile des FPGAs während des Betriebes veränderbar, um die Funktionalität zu erweitern oder zu ändern. Das Schreiben der Daten in den Konfigurationsspeicher erfolgt mit einer maximalen Frequenz von 100 MHz[1], so dass bei einer Busbreite von 32 Bit eine Konfiguration mit einem Datendurchsatz von 3200 MB/s erfolgt. Wobei diese Begrenzung für den Takteingang des HWICAP Modul greift. Der angeschlossene Bus kann mit einer höheren Frequenz betrieben werden. Bei der Anbindung des HWICAP an den MicroBlaze wird das HW-IP mit einem XPS-HWICAP Wrapper versehen. Es koppelt die Signale des PLB von der Funktionalität und übergibt die zu schreibenden Daten über ein „IP Interconnect“ (IPIC) an das HWICAP (vgl. Abbildung 15). Es schreibt die Daten mit der an dem *ICAP_CLK* Pin angelegten Frequenz in den FPGA. Bei der Verwendung des HWICAP im Interrupt Modus, welcher beispielsweise einen leeren FiFo anzeigt, ist der *IP2INTC_lprt* an den Interruptcontroller des MB Systems anzuschließen [Xilinx (2010i)].

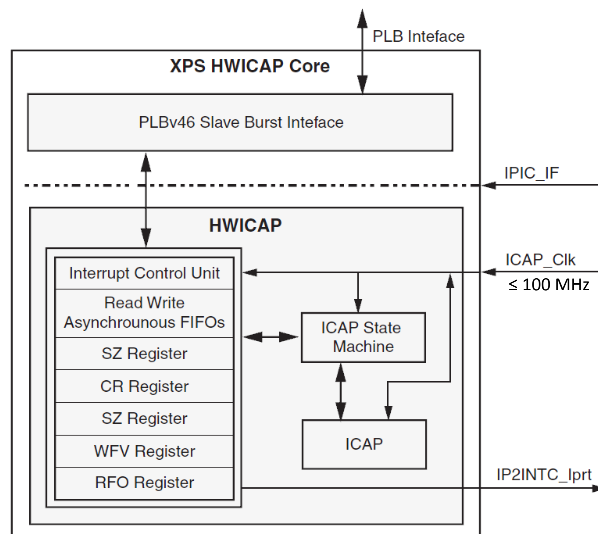


Abb. 15: *Interne Register und Interfaces des XPS HWICAP Modules [Xilinx (2010i)]*

Die Verwendung des *XPS HWICAP* erfolgt in drei Schritten. Der erste Schritt dient der Initialisierung des IPs. Danach wird der Header der Reconfigurationsdatei gelesen und die Anzahl, der zu schreibenden Bytes, extrahiert. Im dritten Schritt werden die Rekonfiguriatonsdaten aus der Datei gelesen und in an das *XPS HWICAP* übergeben, welches den FPGA anschließend rekonfiguriert (vgl. Listing 1)[Xilinx (2010e)].

Listing 1: *Ablauf der Initialisierung zum Verwenden des XPS HWICAP*

```
0


int main (void) {

        //Variablen Deklaration
5


        //1. Schritt
        ConfigPtr = XHwIcap_LookupConfig(XPAR_XPS_HWICAP_0_DEVICE_ID);
        /*Nach der HWICAP LookupConfig*/
10
```

[1] 100 MHz werden in den FPGAs der Reihen Virtex-4, Virtex-5 und Virtex-6 erreicht. Bei Spartan 6 FPGAs ist eine Konfiguration mit maximal 20 MHz durchführbar [Xilinx (2010i)].

```
        Status = XHwIcap_CfgInitialize(&HwIcap, ConfigPtr,
        ConfigPtr->BaseAddress);
        /*HWICAP Initialisiert*/

15
        //2. Schritt



20      /* Datei öffnen */
        stream = sysace_fopen(filename, "r")

        /* Lesen der Datei aus dem SystemAce */
        numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
25                                  stream);

        /* Bitstream Header lesen */
        bit_header = XHwIcap_ReadHeader(systemACE_Buffer,0);

30      /* Schließen der Datei */
        rc = sysace_fclose (stream);

        /* Erneutes Öffnen der Datei zum extrahieren der PRM Daten */
        stream = sysace_fopen(filename, "r")
35

        //3. Schritt

        /* Lesen von Bytes, deren Anzahl der größe des Headers entspricht */
40      numCharsRead = sysace_fread(systemACE_Buffer, 1, bit_header.HeaderLength,
                                    stream);

        /* Lesen des ersten Sektors der PRM Datei */
        numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
45                                  stream);

        SectorNumber = 1;

        /* Die Daten ans HWICAP übergeben und die restlichen Daten lesen */
50      ace_buf_count = 0;
        for (i=0; i<bit_header.BitstreamLength; i+=4)
        {

                /* Convert 4 chars into an integer */
55              data3 = systemACE_Buffer[ace_buf_count++];
                data2 = systemACE_Buffer[ace_buf_count++];
                data1 = systemACE_Buffer[ace_buf_count++];
                data0 = systemACE_Buffer[ace_buf_count++];
                word[0] = ((data3 << 24) | (data2 << 16) | (data1 << 8) | (data0));
60              i+=4;

                XHwIcap_DeviceWrite(&HwIcap, word, 2);

                /* Wurden alle Daten geschrieben, nächsten Sektor lesen */
65              if (ace_buf_count == XSA_CF_SECTOR_SIZE)
                {
                        numCharsRead = sysace_fread(systemACE_Buffer, 1, XSA_CF_SECTOR_SIZE,
                                                    stream);
                        ace_buf_count = 0;
70                      SectorNumber++;
                }
        }

        /* Datei schließen */
75      rc = sysace_fclose (stream);
        return 0;
}
```

3.6 System Advanced Configuration Environment (SYSACE)

Xilinx stellt mit dem „System Advanced Configuration Environment“ (SYSACE) eine Konfigurationsmethode für die FPGAs zur Verfügung, die mit einer Compact-Flash Karte funktioniert und so keine JTAG Verbindung zu einem externen Gerät benötigt. Hierzu wird das System ACE

Controller Device zwischen dem FPGA und einem CF-Slot angeschlossen (vgl. Abbildung 16). Beim Anlegen der Betreibsspannung an den FPGA und dem System ACE Controller werden die Konfigurationsdaten aus der CF-Karte gelesen und über die JTAG Ports des FPGAs geschrieben [Xilinx (2008d)]. Mit Hilfe des *XPS SYSACE* Interface Controller ist ein MicroBlaze System in der Lage den Hardware Controller anzusprechen, um Daten auf die CF-Karte zu schreiben oder von ihr zu lesen. Diese stehen dann, anders als beispielsweise bei DRAM basierten Speichern, auch nach einem Neustart zur Verfügung [Xilinx (2010j)].

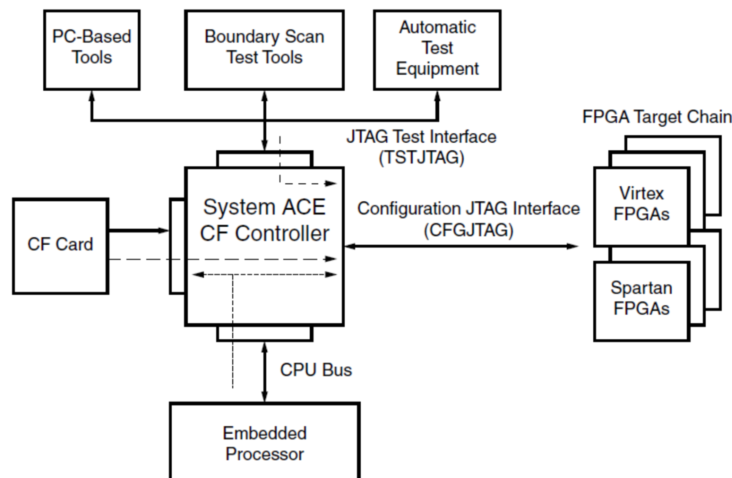


Abb. 16: *FPGA Konfigurationsschaltung mit System ACE und Option zur Konfiguration des FPGAs über einen externen JTAG Anschluss [Xilinx (2008d)]*

3.7 Multi-Port Memory Controller

Der „Multi-Port Memory Controller“ (MPMC) dient der Ansteuerung von externen Speicher Elementen. Unterstützt werden dabei SDRAM, DDR, DDR2, DDR3 und LPDDR Speicher Module. Die Anbindung an den MicroBlaze erfolgt wahlweise über den PLB, den „Fast Simplex Link“ (FSL) oder über den „Xilinx Cache Link“ (XCL). Ist es erforderlich, dass IPs einen direkten Zugriff auf Speicher (DMA) erhalten, so stehen hier das „Native Port Interface“ (NPI) und ein Soft DMA Controller zur Verfügung, welcher einen Full-Duplex Zugriff auf den Speicher ermöglicht. Insgesamt stehen dem Anwender acht Ports zur Verfügung, welche einzeln auf eine Datenbreite von 4 - 64 Bit konfigurierbar sind [Xilinx (2010c)].

3.8 Real-time Operating System mit POSIX Standard

In das EDK ist der Xilinx Xilkernel integriert, welcher ein leichtgewichtiger Kernel mit einer „Portable Operating System Interface“ (POSIX) API ist [Xilinx (2011l)]. Diese API wurde von der IEEE in Zusammenarbeit mit der Open Group erstellt und dient als Schnittstelle zwischen dem Betriebssystem und den Programmen und umfasst einige Basis-Definitionen, wie beispielsweise Pfadnamen, sowie System-Schnittstellen und Hilfsprogramme. Erstellt wurde sie, um eine Portierbarkeit von Programmen zwischen Betriebssystemen zu ermöglichen und wird beispielsweise von QNX und BSD verwendet [IEEE (2009)].

Der Xilkernel lässt sich in der Größe und Funktionalität durch das An- und Abschalten von Features einstellen. So lassen sich beispielsweise Semaphore dem System hinzufügen oder das

Scheduling von Round-Robin (RR) auf prioritätsbasiert umstellen. Der Kernel selbst ist durch eine Parametervalidierung gesichert und gibt im Fehlerfall POSIX konforme Fehlermeldungen zurück. Bei der Erstellung des „Board Support Packages“ (BSP), welches die Einstellungen für den Xilkernel enthält, können Static-Threads angegeben werden, welche ohne weitere Angaben nach dem Start des Kernels ausgeführt werden. Hinzu kommen die aus der POSIX stammenden Funktionen zum Erstellen und Bearbeiten von Threads. Diese Threads sind C-Funktionen, die als Argument einen Void-Pointer akzeptieren und keinen Rückgabewert haben [Xilinx (2011l)].

Das Scheduling der Tasks erfolgt je nach Konfiguration im Round-Robin oder nach dem Prioritäts-Prinzip. Ist RR aktiviert, so erfolgt das Scheduling nach dem FiFo Prinzip, in dem die aktiven Threads in einer einzelnen Queue abgearbeitet werden und anschließend an das Ende der Queue gelegt werden. Bei dem prioritätsbasiertem Scheduling wird für jede Prioritätsstufe eine Queue angelegt. Diese werden abgearbeitet bis die Queue mit der höchsten Priorität leer ist, anschließend wird die nächste Prioritäts Queue abgearbeitet (vgl. Abbildung 17). In beiden Scheduling Algorithmen wird ein XPS Timer verwendet, welcher nach einem vorher eingestelltem Zeitintervall einen Interrupt generiert. Dieser Interrupt startet den Scheduler, welcher den nächsten aktiven Task ausführt [Xilinx (2011l)].

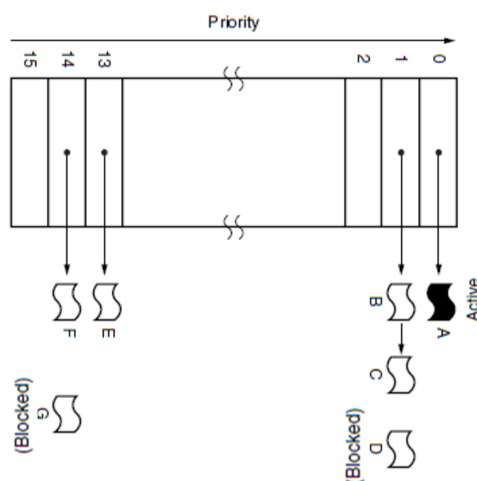


Abb. 17: *Scheduling des Xilkernels bei aktiviertem Prioritätsscheduling. Jede Prioritätsstufe enthält eine Queue, welche die jeweiligen Threads beinhaltet [Xilinx (2011l)].*

Zur Kommunikation zwischen den Threads werden die vom Xilkernel bereitgestellten Mutexe, Semaphoren und Message Queues verwendet. Diese stehen ebenfalls, wie in der POSIX API beschreiben, zur Verfügung. Die Anzahl sowie die Aktivierung der Inter Process Communication ist vor der Verwendung des Xilkernels im BSP festzulegen.

3.9 IP Stack zur softwareseitigen Anbindung des Ethernetcontrollers

Xilinx bietet im EDK eine Portierung des „LightWeight IP“ (LwIP) an, welcher am Swedish Institute of Computer Science entwickelt wurde. Dieser IP Stack ist speziell auf eingebettete Systeme mit einem kleinen Speicher ausgelegt und hat je nach Konfiguration einen Speicherbedarf von ca. 40 kByte ROM und ca. 10 kByte RAM. Konfigurierbar ist dabei eine Unterstützung von beispielsweise UDP, TCP, und DHCP [Dunkels (2004)].

Die LwIP Bibliothek unterstützt neben dem *xps_ethernetlite* Modul den *xps_ll_temac*, welcher in diesem Projekt verwendet wird. Dieser verwendet optional einen FiFo oder einen „Soft Direct Memory Access“ (SDMA) zum Speichern der Daten. Bei der Verwendung des SDMA wird ein MPMC benötigt, welcher mit diesem Modus konfiguriert ist (vgl. Abbilding 18). Die Module

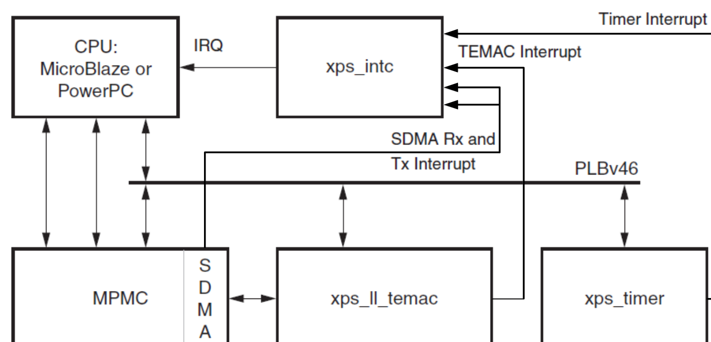


Abb. 18: *Minimal Architektur eines SoCs mit xps_ll_temac Modul und SDMA [Xilinx (2011f)].*

werden zum einen für die Verwendung des LwIP, zum anderen für den Betrieb des Xilkernels genutzt [Xilinx (2011f)].

- *MicroBlaze*:
  Der MicroBlaze führt die Software des Systems aus.
- *xps_intc*:
  In diesem Modul werden die Interrupts des Systems gebündelt und an den MicroBlaze weiter gereicht.
- *MPMC*:
  Der auszuführende Code des Programmes wird über den MPMC aus einem DRAM gelesen. Die ein- und ausgehenden Daten des *xps_ll_temac* werden über eine SDMA Schnittstelle an den MPMC übergeben.
- *xps_ll_temac*:
  Hier werden die zu sendenden Daten an das Phy Interface übergeben und eingehende Daten über die SDMA Schnittstelle im DRAM gespeichert. Der Interrupt zeigt beispielsweise an, dass ein Frame empfangen oder gesendet wurde.
- *xps_timer*:
  Der Timer wird für das Scheduling im Xilkernel verwendet. Es wird nach dem Ende einer eingestellten Periode ein Interrupt generiert, welche das Ablaufen einer Zeitscheibe signalisiert.

LwIP unterstützt den RAW Mode und den Socket Mode, in dem der Scheduler des Xilkernels zur parallel laufenden Verarbeitung der TCP/IP Daten genutzt wird. Bei der Verwendung des Socket Modes ist weiter das Prioritätslevel anzugeben, in welchem die Threads des LwIP laufen. Diese Threads dienen der Kommunikation mit dem *xps_ll_temac* und dem Aufbereiten der Daten für die Weiterbearbeitung in der Anwendersoftware.

In Listing 2 ist der Ablauf zum Initialisieren des LwIP dargestellt. Sobald der Applikations Thread gestartet wurde, stehen die Funktionen zum Lesen und Schreiben über die Sockets bereit [Xilinx (2011k)].

Listing 2: *Ablauf der Initialisierung des LwIP*

```
int main_thread()
{
        /* Initialisieren des LwIP */
                lwip_init();

        /* Erzeugen des Threads der die Schnittstelle einrichtet */
        sys_thread_new("network_thread", network_thread, NULL,
                THREAD_STACKSIZE DEFAULT_THREAD_PRIO);

        return 0;
}

void network_thread(void *p)
{
        struct netif *netif;
        struct ip_addr ipaddr, netmask, gw;

        /* Erzeugen der MAC Adresse*/
        unsigned char mac_ethernet_address[] =
        {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};
        netif = &server_netif;

        /* Konfigurieren der IP Adresse, der Netzmaske und des Standard Gateways */
        IP4_ADDR(&ipaddr,192,168,1,10);
        IP4_ADDR(&netmask,255,255,255,0);
        IP4_ADDR(&gw,192,168,1,1);

        /* Einschalten des Interfaces und als Standard setzen */
        xemac_add(netif, &ipaddr, &netmask,     &gw, mac_ethernet_address,
                EMAC_BASEADDR)
        netif_set_default(netif);
        netif_set_up(netif);

        /* Starten des LwIp Lese Threads  */
        sys_thread_new("xemacif_input_thread", xemacif_input_thread,
                netif,THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);


        /* Starten des Apllikations Threads */
        sys_thread_new("app_thread", web_application_thread, 0,
                THREAD_STACKSIZE DEFAULT_THREAD_PRIO);

}
```

3.10 Motorola S-Record zur Speicherung von ausführbarem Code im Flash

Das Motorola S-Record (SREC) Dateiformat ist ein ASCII basiertes Format zur Kodierung von Programm- und Datendateien, welches von Xilinx zur Speicherung von MicroBlaze Programmen im Flash-Speicher verwendet wird. Die ASCI Kodierung unterstützt dabei eine visuelle Darstellung und das Editieren bereits vorhandener Dateien. Die in den eigentlichen Dateien enthaltenen Binärwerte werden so kodiert, dass jedes Byte durch zwei ASCI Charactere ersetzt werden, bei der das erste Zeichen die hochwertigen vier Bits darstellen, so dass eine Big Endian Kodierung entsteht (vgl.Gleichung 1) [Motorola (1992)].

$$01001000 \Rightarrow 48 \Rightarrow „H“ \quad (1)$$

Jede Zeile der SREC Datei besteht aus den Feldern Typ, Record-Länge, Adresse, Code/Daten und einer Checksumme, welche jeweils mit einem „CR“ abgeschlossen sind (vgl. Tabelle 6) [Motorola (1992)].

Tabelle 6: *Zeilenaufbau des SREC Formats [Motorola (1992)]*

Feld	Charactere	Inhalt
Typ	2	SREC Typ
Record-Länge	2	Anzahl der Charakter Paare ohne den Typen und die Record-Länge
Adresse	4,6,8	die Adresse, an die die Daten zu schreiben sind.
Code/Daten	0 - 2n	Die Daten die in den Speicher geschrieben werden. Es kann beispielsweise Programm-Code enthalten.
Checksumme	2	Das letzte Byte des Einerkomplement der Summe, der Daten präsentiert von den Paaren aus Charakteren aus den Record-Längen, Adress und Code/Datum Feld.

Die in dem Feld Typ angegebenen Typen sind [2] [Motorola (1992)]:

- S0: Beschreibt den Header-Record mit Informationen über die in der Datei enthaltenen Daten.
- S1/S2/S3: Record mit Daten, die an eine 2/3/4 Byte Adresse geschrieben werden.
- S5 : Dieser Record beinhaltet die Anzahl der vorangegangenen S1-S3 Records.
- S7/S8/S9: Mit diesen Records wird eine Terminierung einer Datei angezeigt, die S1/S2/S3 Records enthalten hat.

[2] Die Typen 4 und 6 sind nicht definiert.

4 Architektur des FPGA basierten Distributed Computing Systems

Die Berechnung komplexer Algorithmen, die auf herkömmlichen PCs eine lange Zeitspanne zur Bearbeitung in Anspruch nehmen, wird auf Supercomputern oder DCSs vollzogen. Bei dem Einsatz von DCS werden die Projekt-Daten von CPUs und GPUs berechnet, die zumeist von freiwilligen Teilnehmern dem Projekt zur Verfügung gestellt werden. In dem „Distributed SoC Network“ (DSN) werden FPGA basierte SoCs zur Berechnung der Projekt-Daten verwendet, welche einen Geschwindigkeitsvorteil gegenüber CPUs aufweisen (vgl. Kapitel 1). Zur Realisierung dieses Netzwerkes sind die folgenden Anforderungen erfüllt worden:

- Bereitstellen einer Infrastruktur zur Kommunikation zwischen den Anwender-Projekten und den SoCs
- Aufbau eines SoC-Clients mit der Fähigkeit zur Kommunikation mit dem Server und der partiellen Rekonfiguration von FPGA Teilbereichen
- Zur Algorithmenauswertung sind Rekonfigurationsdateien sowie entsprechenden Datensätze zur Verfügung zur stellen.

Die Architektur des DSN besteht dabei aus den folgenden Komponenten:

- Projekt:
  Das Projekt stellt Konfigurationsdateien und die Daten zur Verfügung, die zu bearbeiten sind. Diese werden an das SoC gesendet und die Ergebnisse je nach Projekt auf dem Anwender Client weiterverarbeitet.
- SoC:
  Das SoC berechnet mit der vom Projekt bereitgestellten Konfiguration die Daten und schickt die Ergebnisse nach der Verarbeitung zurück an das Projekt.

Unter der alleinigen Verwendung dieser Komponenten ergibt sich eine Client-Server Architektur mit dem Projekt als Server und dem SoC als Client (vgl. Abbildung 19). Die Kommunikation findet hierbei direkt zwischen dem Soc und dem Projekt statt. Eine Verwaltung der SoCs und die Kontrolle der jeweiligen Zustände obliegt dem Projekt, so dass hier keine Fokussierung auf das eigentliche Projekt erfolgt (vgl. Tabelle 7).

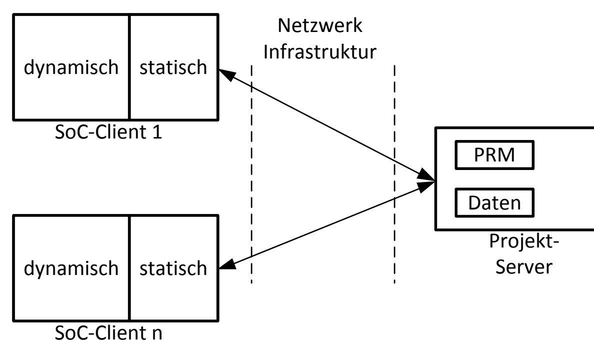


Abb. 19: *Aufbau des DSN unter der Verwendung der „Client-Server“ Architektur*

Ein entscheidender Nachteil der Client-Server Architektur ist, dass jedes SoC die Adresse des Projektes zu kennen hat. Dies führt dazu, dass für jedes Projekt eine Teilmenge der zur Verfügung stehenden SoCs genutzt wird und die SoCs im Leerlauf sind, sobald das Projekt keine Daten mehr zur Verarbeitung zur Verfügung stellt, wodurch die Ressourcen erneut ungenutzt sind. Zur Vermeidung dieses Leerlaufes sowie einer effizienteren Nutzung der SoCs wird ein

Tabelle 7: *Vor- und Nachteile der „Client-Server“ Implementierung*

	Vorteile	Nachteile
Kommunikation	direkte Kommunikation	nur ein Projekt ist dem SoC bekannt
Verwaltung	direkte Verwaltung der SoCs	Jedes Projekt enthält eine eigene Verwaltung
Ausfallsicherheit	-	ist der Projektserver nicht erreichbar werden keine Daten verarbeitet
Auslastung	alle SoCs arbeiten für ein Projekt	sind die Daten für das Projekt berechnet, so sind die SoCs im Leerlauf
SoC Anzahl	die SoCs stehen allein einem Projekt zur Verfügung	es stehen nur Projekt eigene SoCs zur Verfügung

Broker eingesetzt, der als zentraler Punkt des DSN agiert (vgl. Abbildung 20) [Bengel (2004)]. Die Projekte und SoCs melden sich beim Start am Broker an, welcher die SoCs auf die Projekte verteilt (vgl. Anhang A). Mit dieser Architektur wird eine optimale Auslastung der SoCs erreicht, sowie eine gleichmäßige Verteilung der SoCs auf die zu bearbeitenden Projekte. Die weiteren Eigenschaften sind in Tabelle 8 dargestellt.

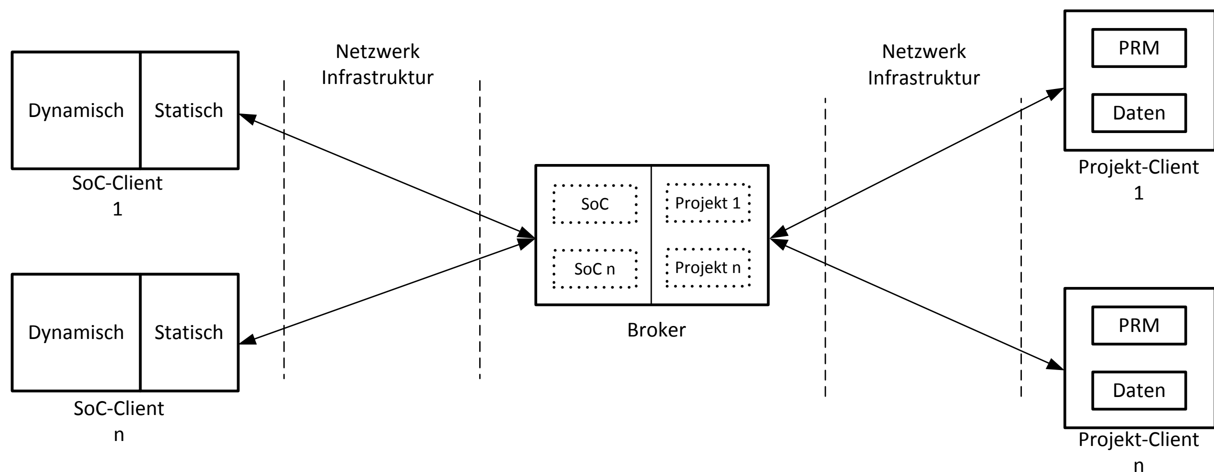


Abb. 20: *Client-Broker-Client Architektur des DSN mit zentraler Stelle für alle Projekte und SoCs zur optimalen Verteilung der Ressourcen auf die Projekte.*

4.1 SoC-Client zur Berechnung der Datensätze im Distributed Computing System

Der SoC-Client wird in dem DSN für die Bearbeitung der Projektdaten verwendet. Zur Erfüllung dieser Aufgabe waren die folgenden Punkte zu realisieren:

- Konfiguration des statischen Systems beim Anschalten: Dies erfolgt um eine Konfiguration vom Anwender zu vermeiden, so dass sich die SoCs selbstständig mit dem Broker verbinden.

Tabelle 8: *Vor- und Nachteile der Client-Broker-Server Implementierung*

	Vorteile	Nachteile
Kommunikation	für alle SoCs existiert ein zentraler Server	die Daten werden über eine weitere Komponente geschickt
Verwaltung	zentrale Verwaltung der SoCs und Projekte	keine direkte Kontrolle innerhalb der Projekte
Ausfallsicherheit	bei der Implementierung mit mehreren Brokern entsteht eine hohe Ausfallsicherheit	wird ein alleiniger Broker eingesetzt, sind alle SoCs im Leerlauf sobald dieser ausfällt
Auslastung	sobald die Daten für ein Projekt berechnet wurden, stehen Daten aus anderen Projekte zur Verarbeitung an	-
SoC Anzahl	die SoCs aller Teilnehmer stehen jedem Projekt zur Verfügung	-

- Abarbeitung von Software mit einem Microprozessor: Die Verbindung zum Broker sowie das Speichern der Daten hat der Microprozessor zu steuern. Dieser stellt die zu bearbeitenden Datensätze nach der Übertragung ebenfalls dem dynamischen Bereich zur Verfügung.
- Kommunikation mit dem Server: Zu Übertragung der Daten ist ein Netzwerk Interface bereitzustellen.
- Speicherplatz zum Halten von PRM- und Projekt-Daten: Auf dem SoC-Clienten sind die zu bearbeitenden Datensätze zu Speichern, bis das Projekt-Modul dieses bearbeitet hat.
- Bereitstellung eines dynamischen Bereiches zur Bearbeitung der Projektdaten: Der dynamische Bereich wird den Projekten zur Verfügung gestellt und mit den entsprechenden Module beschrieben.

Zur Bearbeitung dieser Aufgaben sind die folgenden Komponenten zu realisieren (vgl. Abbildung 21).

- Microprozessor:
  Der Microprozessor Verwaltet die ankommenden Rekonfigurationen und Datensätze und organisiert das Speichern der bearbeiteten Daten.
- Ethernetcontroller:
  Stellt die Anbindung zum Netzwerk her und übergibt die übertragenen Daten an den Speicherkontroller.
- Speicherkcntroller:
  Ansteuerung eines externes Speichers zum Speichern von PRMs und Projekt-Daten.
- Rekonfigurationscontroller:
  Rekonfiguriert die PRR während des Betriebes mit den PRMs, die im Speicher des Systems gehalten werden.

- Konfigurationscontroller:
  Initialisiert den FPGA beim Einschalten des Systems. Die Daten sind auf einem nicht-flüchtigen Speicher zu laden, wie beispielsweise einem Flash basiertem Speichermedium.
- Dyn_Interface:
  Dient der Kommunikation mit den PRMs und synchronisiert die Daten aus dem Speicher mit der Verarbeitungsgeschwindigkeit der PRM.

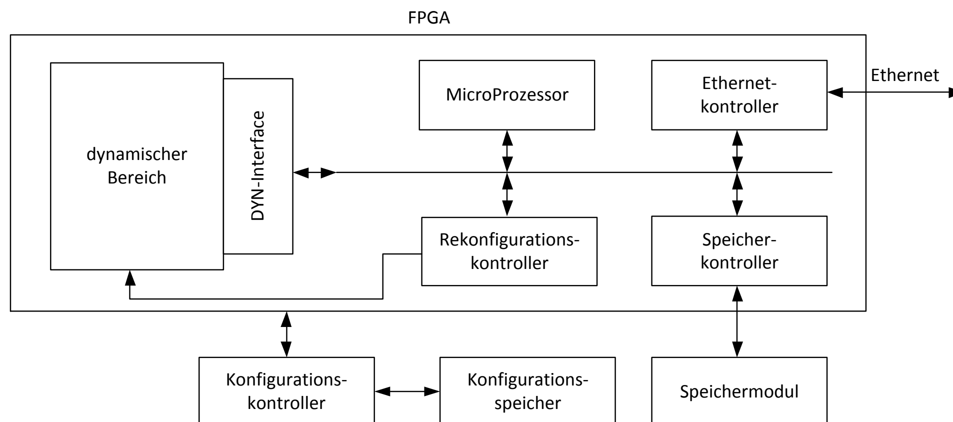


Abb. 21: *Minimaler Systemaufbau des SoC-Clients zur Erfüllung der Aufgaben im DSN*

Der dynamische Bereich definiert die FPGA-Teilfläche und die verfügbaren Ressourcen im FPGA, welche für die Projekt PRMs zu verwenden sind. Die Ansteuerung dieses Bereiches erfolgt über ein Interface, welches so entworfen ist, dass viele Projekte in der Lage sind mit diesem zu Arbeiten. Hier wird ein FiFo basiertes Interfaces verwendet, welches die Daten bereithält, die vom Projekt zur Verfügung gestellt werden. Das Eingangs-FiFo (*IN_FiFO*) wird über ein Bus System mit Daten versorgt, welches an den Speicher angeschlossen ist. Nach der Bearbeitung der Daten durch das Projekt-Modul, werden diese in das Ausgangs-FiFo (*OUT_FiFO*) geschrieben und vom Bus System an die definierte Stelle im Speicher geschrieben (vgl. Abbildung 22).

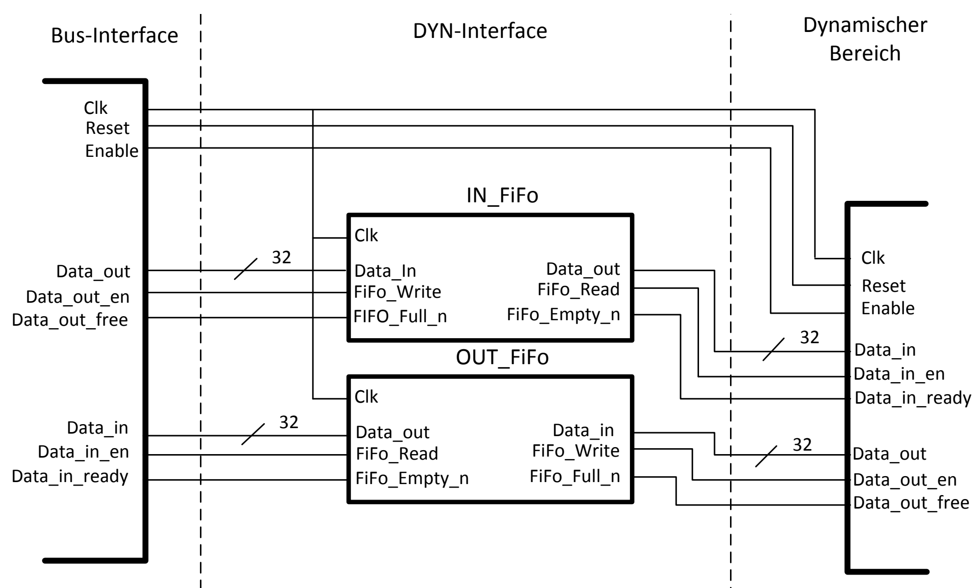


Abb. 22: *FiFo basiertes Interfaces zwischen dem dynamischen Bereich und dem Bus System*

Das Füllen der FiFos wird über das vom SoC-Client verwendete Bus-System gesteuert. Die Daten werden dabei über eine Ethernet Anbindung in den genutzten Speicher geschrieben und anschließend dem *Dyn_Interface* bereitgestellt.

Protokoll zur Kommunikation zwischen Server und SoC

Das SoC verarbeitet die Projekt Daten, wobei die hierfür verwendete PRM beim Einschalten des Systems vom Projekt-Client über den Server, an das SoC übertragen wird. Ist dieses übertragen worden und die PRR erfolgreich programmiert, so werden dem SoC die zu berechnenden Daten übermittelt. Sind die Daten durch die Projekt-PRM bearbeitet worden, erfolgt das Senden des Ergebnisses an den Server, welcher diese weiter an den Projekt-Client übermittelt.

Das Übertragen der PRMs sowie der Eingangsdaten erfolgt jeweils in einem Vier-Phasen-Handshake (vgl. Abbildung 24):

1. Anfrage zum Übertragen „r“
2. Bestätigung des Partners „a“
3. Übertragen der PRM oder Eingangsdaten
4. Bestätigung, dass die Übertragung stattgefunden hat „o“

Aufgebaut sind diese Nachrichten aus sechs Byte, die sich zusammensetzen aus der Art der Anfrage sowie der Phase als Char Datentyp und der Länge der zu übertragenden Daten als Integer (vgl. Abbildung 23). Bei der Bestätigung der Anfrage (Phase 2) sowie der Bestätigung,

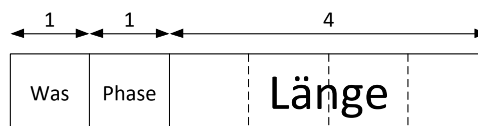


Abb. 23: *Nachricht des 6 Byte Overlay Protokolls, welches zwischen Server und SoC-Client versendet wird. Wobei das erste Byte die Art der Übertragung, also PRM, Datensatz oder Antwort, beinhaltet. Das zweite Feld beinhaltet die Phase im Protokoll und die folgenden 4 Bytes die Länge der zu übertragenen Daten.*

dass alle Daten empfangen wurden (Phase 4) wird jeweils die Länge mit übermittelt, welche bei der Anfrage bekanntgegeben wurde (vgl. Abbildung 24). So besteht beispielsweise die Anfrage zum Versenden eines vier Byte großen Datensatzes aus:

$$„d“ + „r“ + 0x0004$$

Die Übertragung der Eingangsdaten erfolgt in einem definierten Format, welches aus sechs Bytes zuzüglich des Datensatzes besteht. Die ersten vier Bytes dieses Formates enthalten die Länge der zu übetragenen Ausgangsdaten mit dem Zusatz „*OL*“. Im Anschluss werden die zu bearbeitenden Daten gesendet. Die Länge der Ausgangsdaten dient, wie die Länge der Eingangsdaten, der Reservierung von Speicherbereichen und nach der Berechnung zur Angabe der Übertragungslänge (vgl. Abbildung 25).

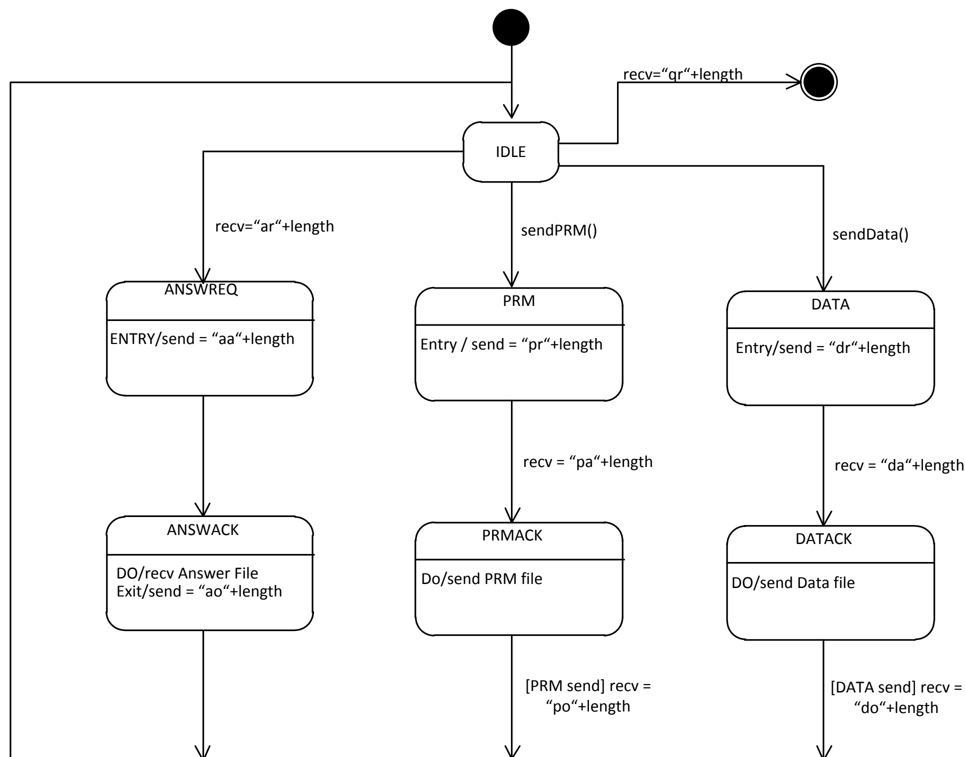


Abb. 24: *Das Kommunikationsprotokoll zur Übertragung der Daten zwischen Soc und Server aus Sicht des Servers.*

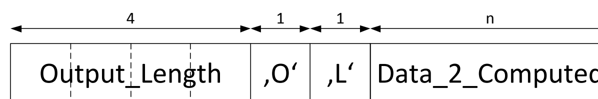


Abb. 25: *Datenformat der Eingangsdaten, welche zur Berechnung bereitgestellt werden*

4.2 Bereitstellen der Daten über den Projekt-Client

Der Projekt-Client stellt die Daten für die Konfiguration sowie die zu bearbeitenden Datensätze bereit. Die hierbei genutzten Anwendungsfälle sind in Abbildung 26 dargestellt.

- Anwender:
  - Anmelden: Beim Starten des Projekt-Clients stellt dieser die Verbindung zum Broker her und meldet sich an.
  - PRM bereitstellen: Der Anwender teilt dem Projekt-Client mit, welche Konfiguration bereit stehen und an den Broker gesendet werden sollen. Ist die Auswahl getroffen, werden die PRMs an den Broker gesendet.
  - Datensatz bereitstellen: Ist die Auswahl der Konfiguration getroffen, werden die zugehörigen Datensätze ausgewählt. Diese werden anschließend an den Broker gesendet und an die SoC-Clients verteilt.
- Broker:
  - Antwort übergeben: Der berechnete Antwortdatensatz wird vom Broker an den Projekt-Client übergeben, dieser teilt daraufhin das Ergebnis dem Anwender oder einer weiteren Anwendung mit.

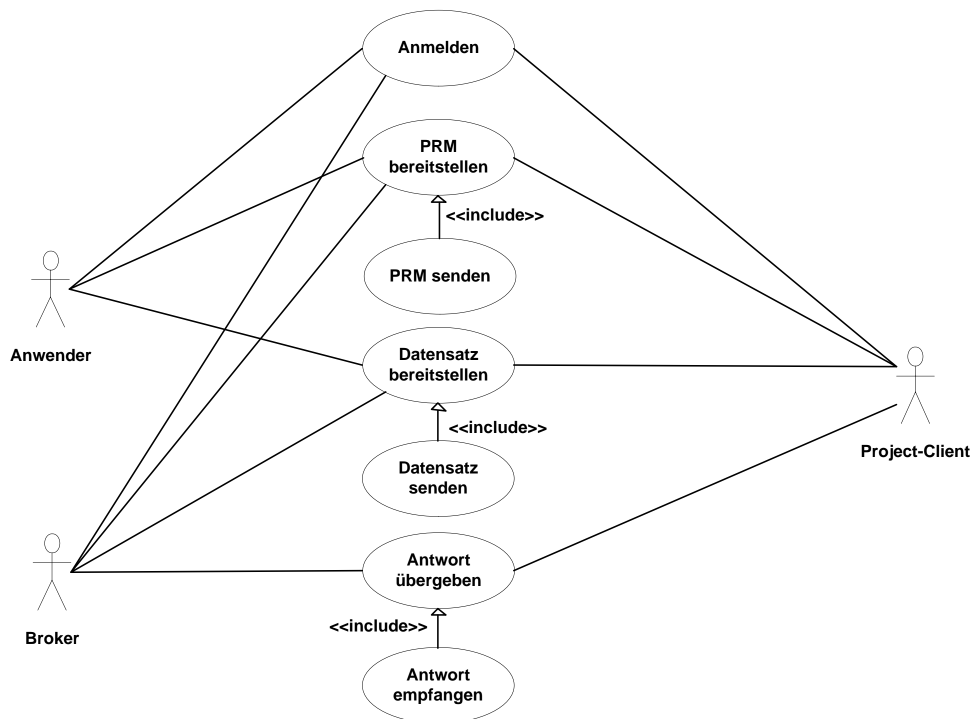


Abb. 26: *Use-Case des Projekt-Clients mit der Interaktion des Anwenders sowie der Kommunikation zwischen dem Project-Client und dem Broker (vgl. Abbildung 20).*

Zur Realisierung dieser Funktionen ist der Projekt-Client in die Komponenten Connection, Persistenz und Manager aufgeteilt (vgl. Abbildung 28). Diese sind über Interfaces realisiert, welche eine abstrakte Definition einer Schnittstelle darstellen, welche bei der realisierung konkretisiert werden.

- Connection: Das Connection Modul baut die Verbindung zum Broker auf und versendet die Daten, die zur Konfiguration und Berechnung verwendet werden.
- Persistenz: Das Speichern und Lesen der Daten erfolgt im Persistenz Modul. Dieses kapselt die eigentliche Speicherung, wie beispielsweise das Speichern in einer Datenbank, von dem restlichem Projekt-Client ab.
- Manager: Die Verteilung der Daten sowie die Kommunikation zwischen dem Connection und dem Persistenz Modul wird über den Manager gesteuert. Dieser gibt ebenfalls die Daten an eine User-Schnittstelle weiter und reagiert auf Eingaben des Benutzers.

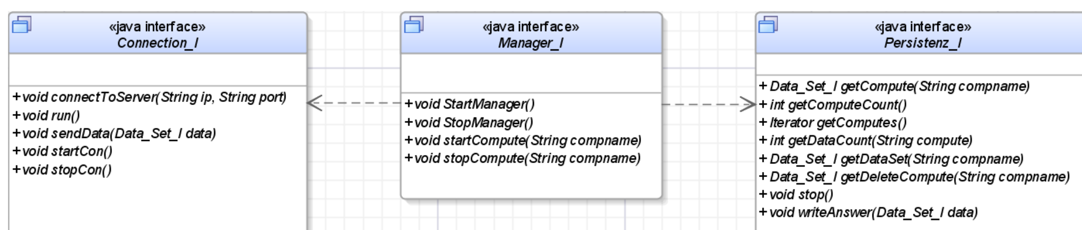


Abb. 27: *Klassendiagramm der Interfaces Connection_I, Manager_I und Persistenz_I zur Realisierung der Projekt-Clienten mit den zur Erfüllung des Use-Cases verwendeten Funktionen.*

Datenaustausch zwischen Broker und Projekt-Client

Die Kommunikation zwischen dem Broker und den einzelnen Projekt-Clienten erfolgt auf der Basis von XML [Abts (2010)]. Dies erfordert im Gegensatz zu einer CORBA oder RMI basierten Lösung keine weiteren Infrastruktur Elemente wie einen Registry Server, so dass eine direkte Kommunikation erfolgt, ohne beispielsweise ein Naming-lookup zu vollziehen, welches die Adresse des Servers liefert. Zum Versenden der Konfigurationsdateien sowie der Datensätze ist eine Klasse zu definieren, welche zum einen den Datensatz beschreibt, zum anderen den Inhalt der Datei bereitstellt (vgl. Abbildung 28).

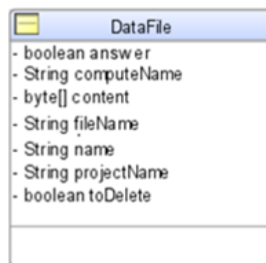


Abb. 28: *Attribute der zur Übertragung verwendeten DataFile Klasse in vereinfachter Form zur besseren Darstellung ohne Getter/Setter Methoden, welche für die Erzeugung der XML Dateien genutzt werden [XStream (2011)].*

Die folgenden Attribute werden dazu verwendet:

- answer: Dieser Wert gibt an, ob die Datei eine Antwort-Datei mit Auswertungsergebnissen ist. Der Wert wird auf der Seite des Projekt-Clienten zur Auswahl des Speicher-Ortes verwendet.
- computeName: Name der Konfigurationseinheit
- content: Inhalt der Datei als Byte Array. Dieser ist vor dem Versenden aus der Datei zu lesen.
- fileName: Enthält den konkreten Namen der Datei. Dies ist bei einer PRM beispielsweise: *prmKonfig.bit* .
- name: Dieser Namen wird gesetzt, wenn es sich bei der Datei um einen Datensatz handelt. Ist dieser Name leer, handelt es sich um eine Konfigurationsdatei.
- projectName: Der Name des Projektes, zu dem diese Daten gehören.
- toDelete: Bei der Abmeldung einer Konfiguration vom Broker wird dieses Flag gesetzt und die entsprechenden Dateien werden vom Broker gelöscht.

Die mit der Klasse *DataFile* erstellten Objekte werden als XML Datei zwischen dem Broker und dem Projekt-Client ausgetauscht, welche wie in Listing 3 aufgebaut ist. Diese Implementierung des Austausches erlaubt ebenfalls den Transfer von Projekt-Daten zwischen verschiedenen Technologien und Programmierspachen, so dass beispielsweise ein Projekt-Client in C# implementiert werden kann oder eine Übergabe von Datensätzen aus einem System, wie beispielsweise einem autonomen Fahrzeug erfolgen kann, welches so Rechenleistung auslagert und anhand der berechneten Ergebnisse Entscheidungen trifft.

Listing 3: *XML Aufbau der zu versenden Objekte zwischen Broker und Projekt-Client*

```
0 <?xml version="1.0" ?>
  <FileIO.DataFile>
    <projectName>String</projectName>
    <computeName>String</computeName>
    <name>String</name>
5   <answer>boolean</answer>
    <toDelete>boolean</toDelete>
    <fileName>String</fileName>
    <content>byte[]</content>
  </FileIO.DataFile>
```

4.3 Broker zur Verteilung der SoC-Clienten und der Datensätze

Die Kommunikation zwischen den SoC-Clienten und den Projekten erfolgt über den Broker, der so den zentralen Punkt des DSN darstellt und die Verteilung der SoC Ressourcen auf die Projekte vornimmt. Zur Realisierung dieser Funktionalität sind die in Abbildung 29 dargestellten Anwendungsfälle implementiert.

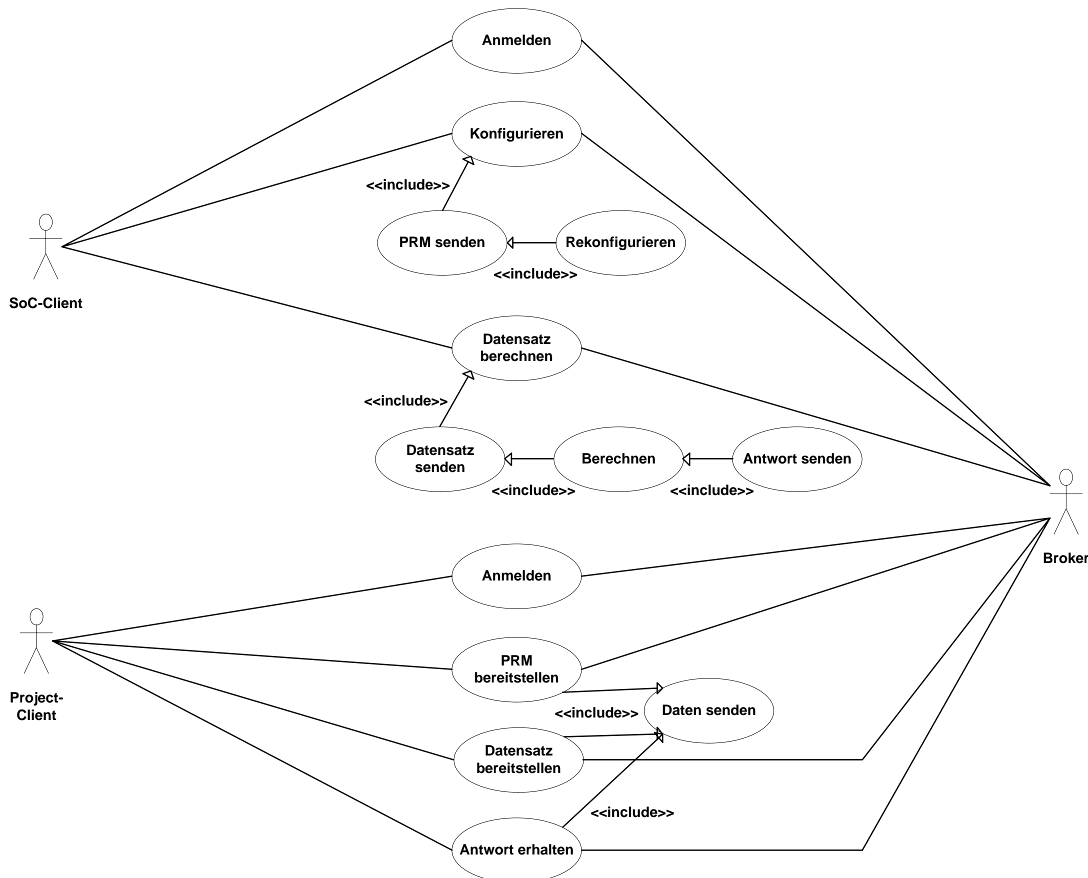


Abb. 29: *Aufbau des DSN unter der Verwendung der Client-Server Architektur*

- SoC-Client:
  - Anmelden: Der SoC-Client meldet sich beim Broker an und wird so in die Liste der zu verwaltenden SoCs aufgenommen.

  - Konfigurieren: Wird dem SoC-Client ein Projekt zugewiesen, so wird die dazugehörigen Konfigurationsdatei versendet und auf dem FPGA eine Rekonfiguration vorgenommen.
  - Datensatz-Berechnen: Ist der Client einem Projekt zugewiesen und konfiguriert, werden die zugehörigen Datensätze übermittelt, verarbeitet und anschließend wird die entsprechende Antwort an den Broker gesendet.

- Project-Client:
  - Anmelden: Das Projekt stellt eine Verbindung zum Broker her und wird in die Liste der angemeldeten Projekte aufgenommen.
  - Konfiguration bereitstellen: Die zum Projekt gehörenden Konfigurationen werden an den Broker übertragen.
  - Datensatz bereitstellen: Die Datensätze zu den Konfigurationen werden übertragen und zur Berechnung zur Verfügung gestellt.
  - Antwort erhalten: Ist ein Datensatz berechnet, so wird dieser zurück an das Projekt übertragen.

Zur Realisierung dieser Anforderungen wird eine Unterteilung der Aufgaben in die Komponenten SoC-Factory, Project-Factory und Scheduler vollzogen.

- SoC-Factory: Die SoC-Factory ist für die Verwaltung der SoC-Clienten zuständig. Ist ein neuer Client angemeldet, wird dies dem Scheduler mitgeteilt, welcher diese auf die verfügbaren Projekte verteilt und anschließend die Konfigurationsdateien und Datensätze bereitstellt.
- Project-Factory: Hier wird die Schnittstelle zu den Projekten bereitgestellt. Bei einem Verbindungsaufbau eines Projektes werden dessen Eigenschaften dem Scheduler mitgeteilt und die PRMs und Datensätze bei einer Anfrage vom Scheduler übergeben.
- Scheduler: Der Scheduler verteilt die angemeldeten SoCs auf die Projekte. Wird einem SoC ein neues Projekt zugewiesen, so übergibt der Scheduler die dazugehörigen Daten.

Die in Abbildung 30 dargestellten Interfaces repräsentieren die Funktionen des Servers zur Verwaltung der SoC- und Project-Clients sowie das Scheduling der Ressourcen. Zur Kommunikation innerhalb der Komponenten im Server sind zusätzlich die folgenden Interfaces für den Server implementiert:

- IClient: Dieses Interface stellt die Repräsentation des eigentlichen Clienten da. Es regelt die Kommunikation mit dem Clienten und enthält eine Instanz der *IClientDescription*.
- IClientDescription: Die Beschreibung enthält die Daten, die für den Scheduler notwendig sind. Hier wird beispielsweise der Name des Clienten gespeichert oder der aktuelle Zustand. Die Werte der Beschreibung werden vom Clienten gesetzt und vom Scheduler ausgelesen.
- IProject: Das *IProject* Interface steuert die Kommunikation mit dem Projekt. Es erstellt und verwaltet die Daten für die einzelnen Konfigurationen sowie den zu verarbeitenden Daten.
- IProjectDescription: Die Beschreibung des Projektes enthält die Anzahl der Konfigurationen sowie der Anzahl der dazugehörigen Datensätze.

- ADataSet: Die Abstrakte Klasse *ADataSet* enthält die Informationen über die Datensätze, wie den Namen der Datei und der dazugehörigen PRM. Die Funktion *getData()* ist die Abstrakte Funktion zum Erhalten der Daten. Diese können beispielsweise in einer Datenbank oder als Datei auf dem Speichermedium des Servers gespeichert werden

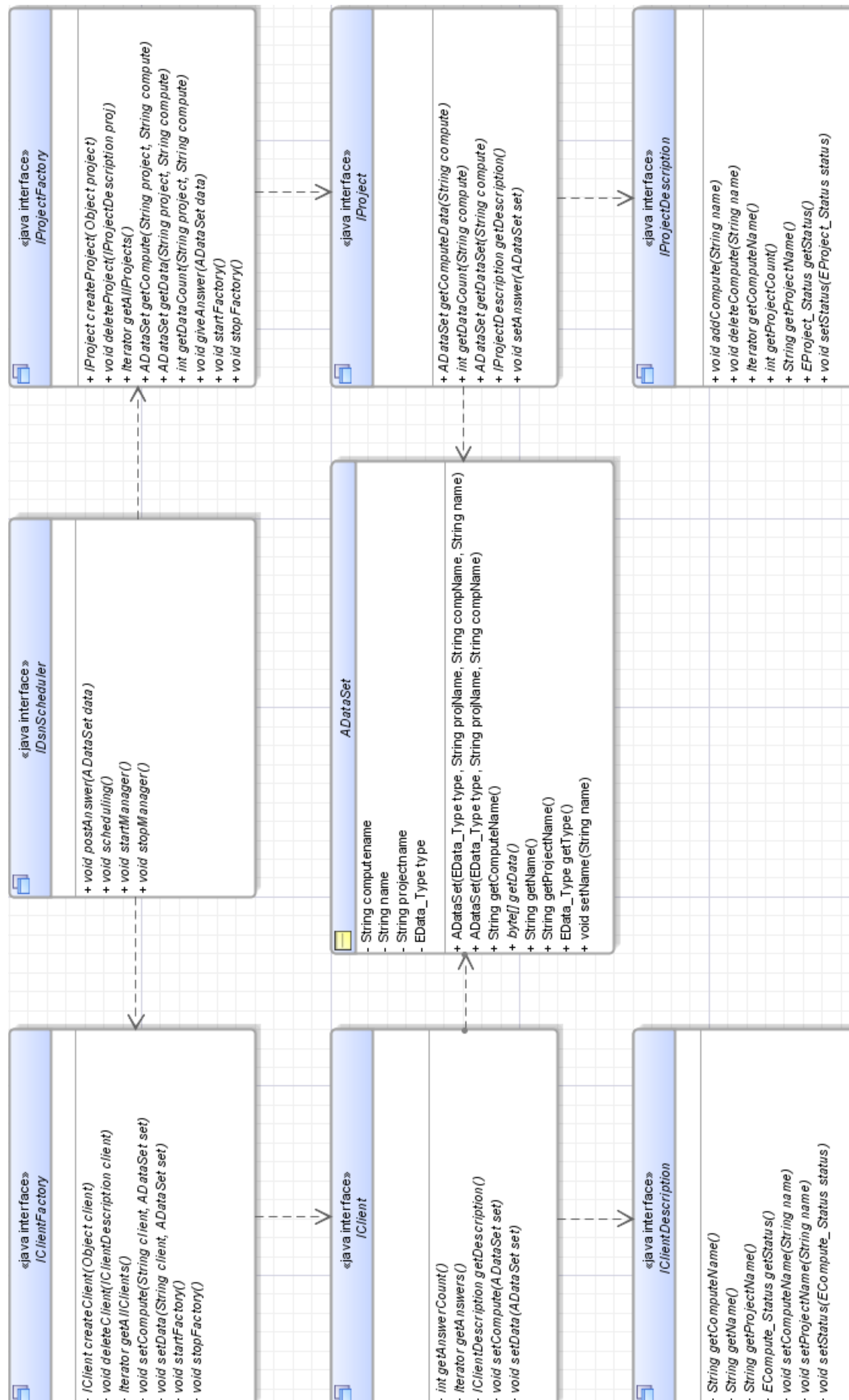


Abb. 30: *Use-Case des Brokers mit der Kommunikation zwischen den SoC-Clients und den Projekten.*

5 Implementierung des SoC-Clients

Die SoCs stellen den Projekten ihre Ressourcen zur Verarbeitung der Datensätze zur Verfügung. Hierzu ist ein System auf Basis des in Kapitel 4.1 entworfenen SoCs zu erstellen:

- Ein statisches Subsystem ist zu implementieren, welches die Kommunikation mit dem Broker vornimmt und die Datensätze zur Rekonfiguration und zur Verarbeitung empfängt und verwertet. Hierzu ist ein Microprozessor System zu verwenden, welches die Anbindung an das Netzwerk sowie einen Speicher bereitstellt.
- Die Kommunikation zwischen dem statischen und dem dynamischen System ist zu implementieren und eine effiziente Anbindung zu evaluieren. Dem dynamischen System sind genügend Ressourcen bereit zu stellen, um komplexe Algorithmen zur verarbeiten.
- Das System ist so zu implementieren, dass eine selbstständige Konfiguration des statischen Systems erfolgt und keine Interaktion mit dem Anwender stattfindet.

Die Implementierung des SoC-Clients erfolgt auf Basis des Xilinx EDK und wird in den folgenden Kapiteln durchgeführt (vgl. Anhang G).

5.1 Statisches Subsystem mit MicroBlaze zur Kommunikation mit dem Broker

Das statische System baut die Kommunikation zum Server auf und konfiguriert die PRR mit der übertragenen PRM. Es besteht aus den folgenden Komponenten (vlg. Abbildung 31,32 , 33 und Anhang D):

- MicroBlaze (microblaze_0):
  Der MicroBlaze ist auf Basis der vorgegebenen „Maximum Performance“ Einstellung konfiguriert, um eine schnelle Übergabe der Daten an die Netzwerkschnittstelle sowie das Speichern der ankommenden Daten zu erreichen (vgl. Kapitel 3.2). Die so vorgegebene Konfigruation wird mit der Verwendung von Interrupts erweitert, um das Scheduling im Xilkernel mit einem Timer zu steuern. Die Anbindung an den Speichercontroller erfolgt über einen Daten- und einen Instruktions-Cache, mit einer Tiefe von jeweils 32 kByte und vier Cache-Lines, sowie der Write-Back Konfiguration im D-Cache, welches bei dieser Single-Core Konfiguration zur eine Steigerung der ausführungsgeschwindigkeit sorgt aber gleichzeitig keine inkosistenz Probleme verursacht. Der Cache deckt den ganzen Adress-Bereich des angebundenen DDR-Speichers ab und verringert zum einen die direkten Zugriffe auf den Speicher, zum anderen wird die Ladezeit der Befehle und Daten verringert [Wilken (2011)].
- MPMC (DDR3_SDRAM):
  Dem MicroBlaze stehen über den „Multi-Port Memory Controller“ (MPMC) 256 MB DDR3 Speicher zur Verfügung. Dieser beinhaltet zum einen das Programm zur Steuerung des SoC-Systems, zum anderen die Daten zur Rekonfiguration sowie die zur Berechnung verwendeten Datensätze und deren Ergebnisse. Die Konfiguration des MPMC erfolgt auf Basis des angeschlossenen Speichers. Dieser bestimmt beispielsweise das Timingverhalten, welches im MPMC anzugeben ist. Die Timings, des auf dem ML605 verfügbaren Speichers MT4JSF6464HY-1G1, sind im Konfigurationsmenü des MPMC hinterlegt und werden bei der Auswahl des entsprechenden BSP beim Erstellen des Projekts automatisch

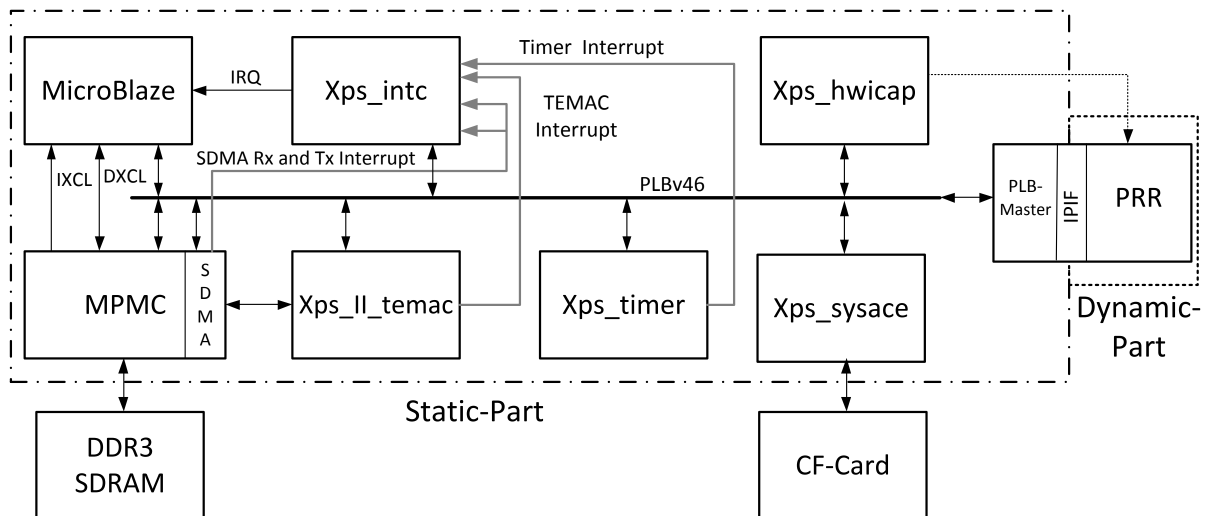
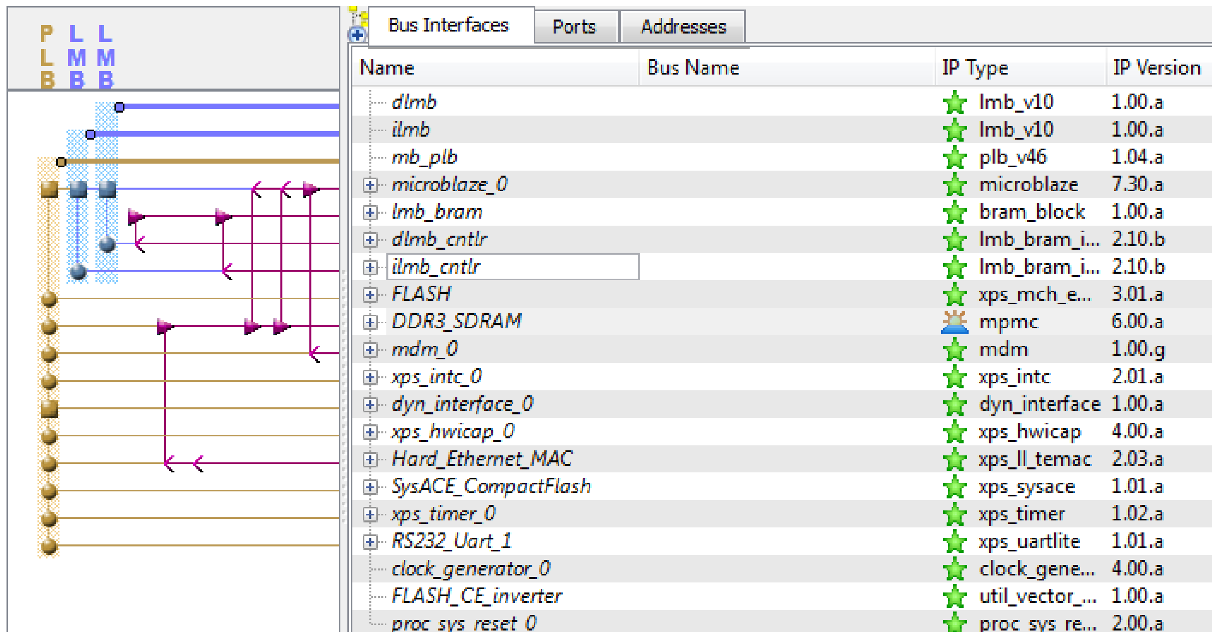


Abb. 31: *SoC-Client mit Komponenten des Xilinx EDK*



Name	Bus Name	IP Type	IP Version
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.04.a
microblaze_0		microblaze	7.30.a
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
FLASH		xps_mch_e...	3.01.a
DDR3_SDRAM		mpmc	6.00.a
mdm_0		mdm	1.00.g
xps_intc_0		xps_intc	2.01.a
dyn_interface_0		dyn_interface	1.00.a
xps_hwicap_0		xps_hwicap	4.00.a
Hard_Ethernet_MAC		xps_ll_temac	2.03.a
SysACE_CompactFlash		xps_sysace	1.01.a
xps_timer_0		xps_timer	1.02.a
RS232_Uart_1		xps_uartlite	1.01.a
clock_generator_0		clock_gene...	4.00.a
FLASH_CE_inverter		util_vector_...	1.00.a
proc_sys_reset_0		proc_sys_re...	2.00.a

Abb. 32: *Aufbau des SoC Systems im System Assembly View des EDK*

Bus Interfaces | Ports | Addresses

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name
microblaze_0's Address Map						
dlmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	dlmb
ilmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	ilmb
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB	mb_plb
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb
xps_timer_0	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB	mb_plb
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb
DDR3_SDRAM	C_SDMA_CTRL_BASEADDR	0x84600000	0x8460FFFF	64K	:SDMA_CTRL2	
xps_hwicap_0	C_BASEADDR	0x86800000	0x8680FFFF	64K	SPLB	mb_plb
Hard_Ethernet_MAC	C_BASEADDR	0x89480000	0x894FFFFF	512K	SPLB	mb_plb
FLASH	C_MEM0_BASEADDR	0x8C000000	0x8DFFFFFF	32M	SPLB	mb_plb
DDR3_SDRAM	C_MPMC_BASEADDR	0x90000000	0x9FFFFFFF	256M	XCL0:XCL1:SDMA_LL2:SPLB3	microblaze_0_IXCL
dyn_interface_0	C_BASEADDR	0xCB000000	0xCB00FFFF	64K	MPLB:SPLB	mb_plb

Abb. 33: *Adressbelegung der Funktionselemente am PLB sowie der angeschlossenen Bus-Systeme im SoC-Client*

gesetzt. Der MPMC besitzt bis zu acht Ports, welche zur Kommunikation und Konfiguration des MPMC dienen. Hier sind folgende Anschlüsse belegt:

– I/DXCL: Die ersten beiden Ports des MPMC dienen dem Anschluss der Caches des MicroBlaze.

– PLB: Der PLB dient der Kommunikation des MicroBlaze mit den Peripherie Elementen im System. Über diesen Bus liest und schreibt das *DYN_Interface* als Bus-Master die Daten aus/in den Speicher.

– SDMA: Der „Soft Direct Memory Access“ (SDMA) bietet einen Full-Duplex Zugriff auf den DDR- Speicher und wird vom Ethernetcontroller verwendet. Zuzüglich zum Anschluss an das Peripherie Modul erfolgt das Anbinden einer Konfigurationseinheit an den PLB (vgl. Abbildung 34). Dieses Konfigurationsmodul wird verwendet, um DMA Zugriffe zu intiieren und Status-Informationen abzufragen.

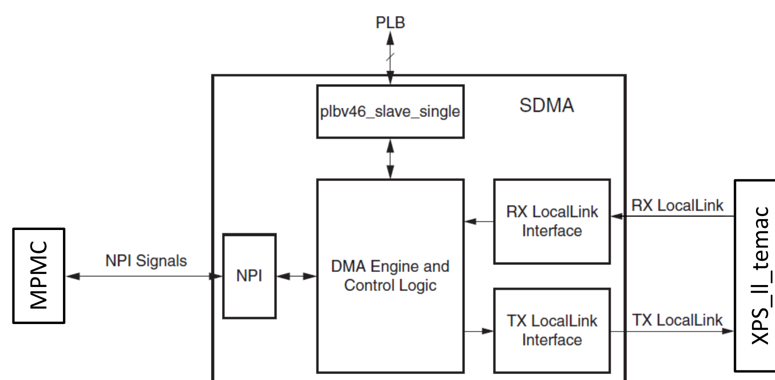


Abb. 34: *Interner Aufbau des SDMA Ports mit LokalLink zum xps_ll_temac, NPI zur Ansteuerung des MPMC und dem PLB zur Konfiguration [Xilinx (2010c)].*

- ICAP (xps_hwicap_0):
  Über das XPS_HWICAP werden die PRM in die PRR geschrieben, wobei diese IP die FPGA-internen HW-ICAP Komponenten steuert und ein Interface zum PLB bereitstellt. Konfiguriert ist der HW-ICAP mit einem Schreib-FiFo von 64 Byte und der Lese-FiFO mit 128 Byte. Die Anbindung des HW-ICAP erfolgt ohne Interrupt. Konfiguriert wird die PRR mit einem Pollingverfahren, da eine Unterbrechung der Rekonfiguration geschieht, wenn die Interrupts des Systems angeschaltet sind (vgl. Kapitel 5.4).

- Netzwerk Anbindung (Hard_Ethernet_Mac):
  Dieses Modul überträgt die PRM und Datensätze zwischen dem SoC-Client und dem Broker über eine Ethernet Schnittstelle. Die Kommunikation zwischen dem MicroBlaze und dem Modul erfolgt über den PLB und wird zur Konfiguration des IPs genutzt. Die übertragenen Daten werden über das LocalLink Interface über einen SDMA Port des MPMC in den DDR-Speicher geschrieben. Als Alternative steht hier ein LocalLink FiFo zur Verfügung, welches für Systeme gedacht ist, in denen kein SDMA-fähiger Speicher Controller verwendet wird. Diese Variante erhöht jedoch den Bedarf an BRAM Modulen, da die Daten direkt in einem BRAM basierten FIFO gespeichert werden. Verwendet wird hier weiter der FPGA-interne TEMAC Hardcore, da dieser bereits als Ressource integriert ist und keine weiteren Ressourcen wie Slices nutzt. Die Adresse des Phy-Interfaces wird in der Software des Clients gesetzt, da so eine schnelle Änderung dieser vollzogen werden kann, was bei der Verwendung mehrerer Clienten in einem Netzwerk von Vorteil ist, da die mehrfach Vergabe von MAC-Adresse zu Fehlfunktionen führt.

- SysACE (SysAce_CompactFlash):
  Das SysACE Modul dient der Ansteuerung einer CompactFlash Karte. Auf dieser Karte wird das statische Subsystem gehalten und beim Einschalten des Systems in den FPGA geschrieben.

- PRR-Anbindung (dyn_interface_0):
  Die Kommunikation zwischen der PRR und dem statischen System findet über das *dyn_interface* statt. Es stellt ein Master Interface zum PLB des MicroBlaze zur Verfügung, so dass Daten ohne die Beteiligung des MicroBlaze aus dem Speicher gelesen bzw. in diesen geschrieben werden können (vgl. Abbildung 31).

Daten zu Empfangen und zu Versenden, sowie die Daten für die Bearbeitung durch das dynamische System vorzubereiten, sind Teilaufgaben des statischen SoC-Systems. Die Anwendungs-Aufgabe des SoCs ist es Projekt-Daten zu verarbeiten, so dass für den statischen Part des Systems ein geringer Ressourcenbedarf anzustreben ist, um für das dynamische System viele Ressourcen zur Verfügung zu stellen. In Anhang B ist der aktuelle Ressourcenbedarf des statischen Systems dargestellt. Besonderes Augenmerk bei Implementierung der Projekt-PRMs, findet die Nutzung von Slices. Diese haben eine maximale Auslastung von 24%, wodurch 76% dem Projekt zur Verfügung stehen. Die Ressourcen, welche eine hohe Auslastung erfahren, wie beispielsweise die ICAPs, sind für die Implementierung der Projekt-PRMs nicht von Bedeutung und stellen so keine Einschränkung dar.

Aus-/Eingangssynchronisation des Ethernet IPs

Bei dem Aufbau des SoC-Client mit dem „Base System Builder“ (BSB) des EDK wird ein „User Constraint File“ (UCF) erstellt. Dieses enthält die physikalischen Elemente, die das Design verwendet, wie die I/O-Elemente oder dem Hardware TEMAC. Die vom BSP erstellte Datei ist auf das ML605 Entwicklungsboard angepasst und enthält bereits die Verwendeten Constraints. Bei der Implementierung des Systems mit EDK wird der SoC-Client erfolgreich erstellt. Im Gegensatz dazu erfolgt das Erstellen mit dem generierten UCF bei der Implementierung von PlanAhead und den entsprechenden PRMs nicht ohne Fehler, welche auftreten, wenn bei der Implementierung einer PRM das statische Modul importiert wurde. Erzeugt wird der Fehler durch den Parameter „*Force*“, welcher bei Elementen des *Hard_Ethernet_MAC* verwendet wird (vgl. Listing 4). Eine Änderungen dieses Parameters auf „*True*“ führt dazu, dass die Implementierung erfolgreich erstellt wird. Die Unterschiede dieser Befehle sind folgend beschrieben [Xilinx (2011a)]:

- TRUE: Ein Eingangs- oder Ausgangs-FlipFlop soll in einem IOB implementiert werden. Kann dies nicht erfolgen, so wird ein Slice-FF verwendet.

- FORCE: Das FlipFlop wird in ein IOB implementiert. Ist dies nicht möglich, wird ein Fehler generiert und die weitere Ausführung des „MAP“ Prozesses wird gestoppt.

Listing 4: *Fehlerhafte Constraints der SoC UCF erstellt im EDK bei der Verwendung im PlanAhead*

```
0
  # Constrain the GMII physical interface flip-flops to IOBs
  # Changed from 'true' to 'force'
  INST "Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/
       RXD_TO_MAC_7" IOB =FORCE;

5 NET "fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin[0]" LOC = AN13;
```

Die Implementierung des Systems mit beiden Versionen der UCF Datei zeigt, dass im Falle der IOB Elemente kein Unterschied besteht und die selben Logik Blöcke verwendet werden. Der verwendete IOB wird direkt mit dem Eingangs-Pin *fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin[0]* des Ethernet IPs verbunden und der Input-Buffer so direkt genutzt wird (vgl. Abbildung 35, 36 und 37 sowie Listing 4).

5.2 Interface zur Kommunikation mit dem dynamischen Subsystem

Das statische Subsystem kommuniziert mit dem dynamischen Subsystem, um dieses zu prametrisieren und die Eingangs- sowie Ausgangsdaten zu übergeben. Für diese Kommunikation stehen der PLB, zum Anschluss an den MicroBlaze und Speicher sowie der FSL, welcher eine direkte Kommunikation mit dem MicroBlaze ermöglicht, zur Verfügung.

Tabelle 9: *Gegenüberstellung des PLB und des FSL als Interface zum dynamischen Subsystem [Xilinx (2009b)][Xilinx (2011e)]*

	FSL	PLB
Anbindung	direkte Anbindung an den MicroBlaze	Bus-Topologie mit Bus Arbiter
Prozessorsauslastung	hoch, da jede Kommunikation über den MicroBlaze läuft	niedrig, wenn IP ein PLB-Master ist; sonst hohe Auslastung, da Datentransfer über den MicroBlaze gesteuert wird
Speicherzugriff	nur über den MicroBlaze	als PLB-Master werden direkte Zugriffe verwendet

Verwendet wird als Interface IP ein PLB-Master, da dieser eine direkte Kommunikation mit dem Speicher ermöglicht, welche den MicroBlaze für eine Priorisierung des LwIP entlastet. Dadurch ist das *DYN_Interface* in der Lage, die Daten direkt aus dem DDR-Speicher zu lesen und nach einer Verarbeitung wieder in den DDR-Speicher zu schreiben ohne das der MicroBlaze jedes Datenwort transferiert. So wird verhindert, dass der Datentransfer die Rechenleistung des MicroBlazes in anspruch nimmt und diesen Blockiert, beziehungsweise der Datentransfer durch den Aufruf der LwIP Funktionen oder Interrupts verlangsamt wird.

Erstellt und parametrisiert ist das Interface mit dem „Create and Import Peripheral Wizard“ des EDK:

- IPIF Services:
  - Software Reset: Der Software Reset setzt das angeschlossene Modul zurück, wobei ein Register in der PLB-Logik des zu erstellen Moduls erzeugt wird, welches das Reset Signal der Anwender-Logik setzt. Dieses Signal ist mit dem System-Rest gekoppelt und bei einer logischen „1“ auf dem System-Reset oder dem Setzen des Software-Resets wird das Modul zurück gesetzt. Genutzt wird der Software-Reset um allein das *DYN_Interface*, ohne andere Module zu beeinflussenm zurückzusetzen.
  - User logic software register: Gibt an, dass Register zu implementieren sind, welche über das PLB-Interface des IPs geschrieben und gelesen werden können.

	Name	Site	Type	#Pins
1	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<0>	ILOGIC_X2Y49	ILOGICE1	4
2	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<1>	ILOGIC_X2Y50	ILOGICE1	4
3	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<2>	ILOGIC_X2Y51	ILOGICE1	4
4	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<3>	ILOGIC_X2Y52	ILOGICE1	4
5	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<4>	ILOGIC_X2Y53	ILOGICE1	4
6	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<5>	ILOGIC_X2Y58	ILOGICE1	4
7	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<6>	ILOGIC_X2Y60	ILOGICE1	4
8	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC<7>	ILOGIC_X2Y61	ILOGICE1	4
9	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RX_DV_TO_MAC	ILOGIC_X2Y48	ILOGICE1	4
10	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RX_ER_TO_MAC	ILOGIC_X2Y47	ILOGICE1	4
11	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld0	IODELAY_X2Y49	IODELAYE1	8
12	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld1	IODELAY_X2Y50	IODELAYE1	8
13	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld2	IODELAY_X2Y51	IODELAYE1	8
14	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld3	IODELAY_X2Y52	IODELAYE1	8
15	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld4	IODELAY_X2Y53	IODELAYE1	8
16	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld5	IODELAY_X2Y58	IODELAYE1	8
17	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld6	IODELAY_X2Y60	IODELAYE1	8
18	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld7	IODELAY_X2Y61	IODELAYE1	8
19	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideldv	IODELAY_X2Y48	IODELAYE1	8
20	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideler	IODELAY_X2Y47	IODELAYE1	8

Abb. 35: *Belegung der Logik Elemente nach dem Map-Prozess mit dem FORCE Parameter, dargestellt im FPGA-Editor*

	Name	Site	Type	#Pins
1	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[0]	ILOGIC_X2Y49	ILOGICE1	4
2	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[1]	ILOGIC_X2Y50	ILOGICE1	4
3	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[2]	ILOGIC_X2Y51	ILOGICE1	4
4	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[3]	ILOGIC_X2Y52	ILOGICE1	4
5	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[4]	ILOGIC_X2Y53	ILOGICE1	4
6	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[5]	ILOGIC_X2Y58	ILOGICE1	4
7	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[6]	ILOGIC_X2Y60	ILOGICE1	4
8	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RXD_TO_MAC[7]	ILOGIC_X2Y61	ILOGICE1	4
9	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RX_DV_TO_MAC	ILOGIC_X2Y48	ILOGICE1	4
10	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/RX_ER_TO_MAC	ILOGIC_X2Y47	ILOGICE1	4
11	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld0	IODELAY_X2Y49	IODELAYE1	8
12	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld1	IODELAY_X2Y50	IODELAYE1	8
13	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld2	IODELAY_X2Y51	IODELAYE1	8
14	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld3	IODELAY_X2Y52	IODELAYE1	8
15	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld4	IODELAY_X2Y53	IODELAYE1	8
16	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld5	IODELAY_X2Y58	IODELAYE1	8
17	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld6	IODELAY_X2Y60	IODELAYE1	8
18	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideld7	IODELAY_X2Y61	IODELAYE1	8
19	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideldv	IODELAY_X2Y48	IODELAYE1	8
20	Hard_Ethernet_MAC/Hard_Ethernet_MAC/V6HARD_SYS.I_TEMAC/SINGLE_GMII.I_EMAC_TOP/gmii/YES_IO_1.ideler	IODELAY_X2Y47	IODELAYE1	8

Abb. 36: *Belegung der Logik Elemente nach dem Map-Prozess mit dem TRUE Parameter, dargestellt im FPGA-Editor*

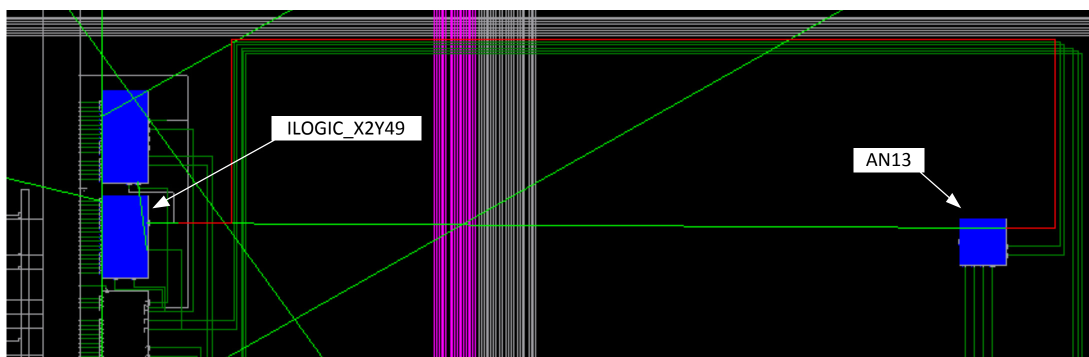


Abb. 37: *Verbindung des Input-Buffers mit dem AN13 Pin des fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin[0] Eingangssignal*

- User logic master: Bei aktiveren der „User logic master“ Option werden die Signale für den Bus Master erstellt und der *User_Logic* Entity übergeben, welche von der Anwender-Logik zur Steuerung des PLBs genutzt werden (vgl. Abbildung 38 und 39).

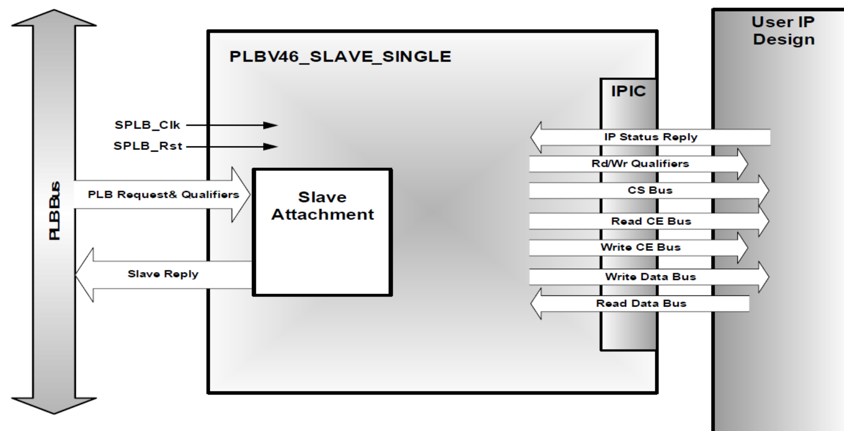


Abb. 38: *Standard Interface eines PLB Slave IPs [Xilinx (2008c)]*

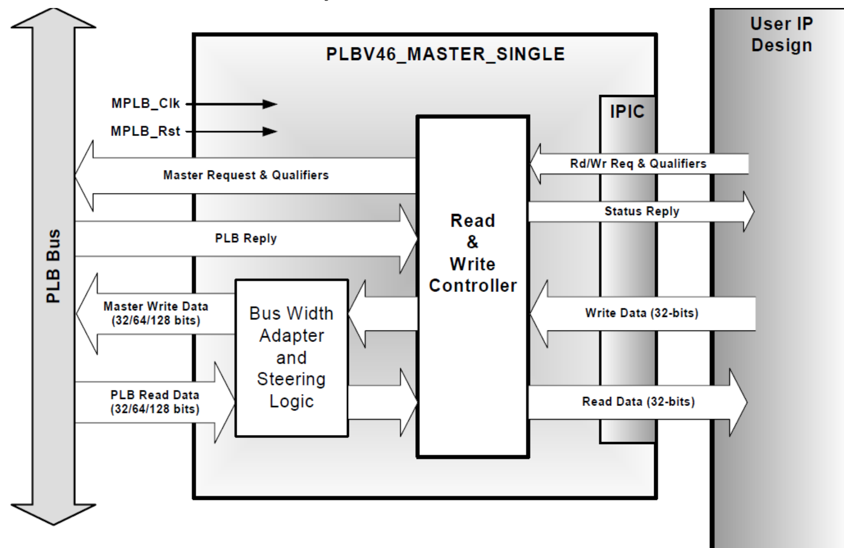


Abb. 39: *Signal Erweiterung des Slave PLB IPs mit dem Master Interface [Xilinx (2008b)]*

- fünf 32 Bit Software Register:
  1. Kontroll-Register: Das Kontroll Register beinhaltet die Status Bits des IPs
     - Bit 0 Bus2IP_Mst_Error: Zeigt einen Fehler auf dem PLB an [IBM (2007)].
     - Bit 1 Bus2IP_Mst_Rearbitrate: Dieses Flag wird vom PLB gesetzt, wenn der angeforderte Befehl zum aktuellen Zeitpunkt vom Slave nicht ausgeführt werden kann [IBM (2007)].
     - Bit 2 Bus2IP_Mst_Cmd_Timeout: Zeigt an, ob bei der Anfrage ein Timeout aufgetreten ist, dies geschieht beispielsweise bei der Verwendung einer ungültigen Adresse [IBM (2007)].
     - Bit 3 read_reached_sig: Wird gesetzt, wenn die Endadresse der Eingangsdaten erreicht ist.

- Bit 4 write_reached_sig: Beim Erreichen der Endadresse der Ausgangsdaten wird dieses Flag gesetzt.
- Bit 30 prm_enable: Dieses Bit wird von der Anwender Software gesetzt und gibt an, ob die PRM Daten verarbeiten kann.
- Bit 31 start_bit: Die Anwender Software setzt dieses Bit zum Starten der Lese- und Schreibbefehle im PLB Master.

2. Anfangsadresse der Eingangsdaten
3. Endadresse der Eingangsdaten
4. Anfangsadresse der Ausgangsdaten
5. Endadresse der Ausgangsdaten

Zudem besteht das Interface aus einem Eingangs- und einem Ausgangs-FiFo, welche die Daten vorhalten, die an die PRR übergeben bzw. von dieser gelesen werden (vgl. Abbildung 40). Die Größe dieser FIFOs wurde auf 16 Datenworte gesetzt, da die Schnelligkeit des Lesens und Schreiben in jedem Projekt unterschiedlich ist und somit eine Berechnung der optimalen Größe nicht stattfinden kann. So kann die Verarbeitung der Daten in einem Takt erfolgen oder mehrere Takte in Anspruch nehmen, wie es beispielsweise bei einigen Bildverarbeitungsalgorithmen der Fall ist (vgl. Kapitel 7). Ein vollständiges Füllen des Eingangs-FiFos erfolgt bei einer Verarbeitungszeit mit mehr als fünf Takten, da diese Zeit vom PLB genutzt wird, um die Leseanfrage durchzuführen. Ebenso wird ein volles Ausgangs-FiFo erreicht, wenn Daten in einem zeitlichen Abstand von weniger als fünf Takte in das Ausgangs-FiFO geschrieben werden [Xilinx (2008b)].

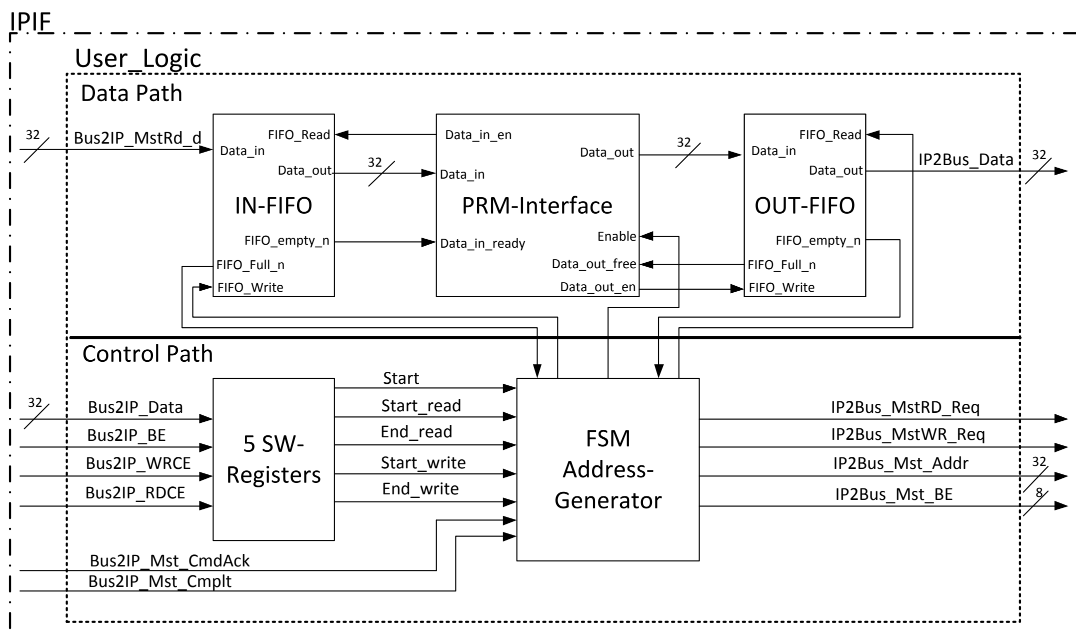


Abb. 40: *Dyn_Interface zur Kommunikation zwischen dem PLB und der PRR*

Aufgebaut ist das *DYN_Interfaces* nach dem Prozessor-Element Entwurfsmuster. Dieses Mustert unterteilt das Modul in einen Steuer- und einen Datenpfad [Gajski (1997)]. Der Steuerpfad gibt hierbei dem Datenpfad die auszuführenden Aktionen vor, wie beispielsweise die Übernahme von Eingangswerten (vgl. Abbildung 40, *FIFO_Write* Enable über die FSM). Der Datenpfad

führt die Schritte zur Verarbeitung aus und stellt dem Steuerpfad Statussignale bereit, diese werden genutzt um mit den von außen anliegenen Signalen den nächsten Zustand zu ermitteln.

Die Ansteuerung des PLB-Masters erfolgt mit einem Automaten in einer Sequentiellen Abarbeitung, da das verwendete IPIF Interface keinen Befehlspuffer enthält (vgl. Abbildung 41[3]):

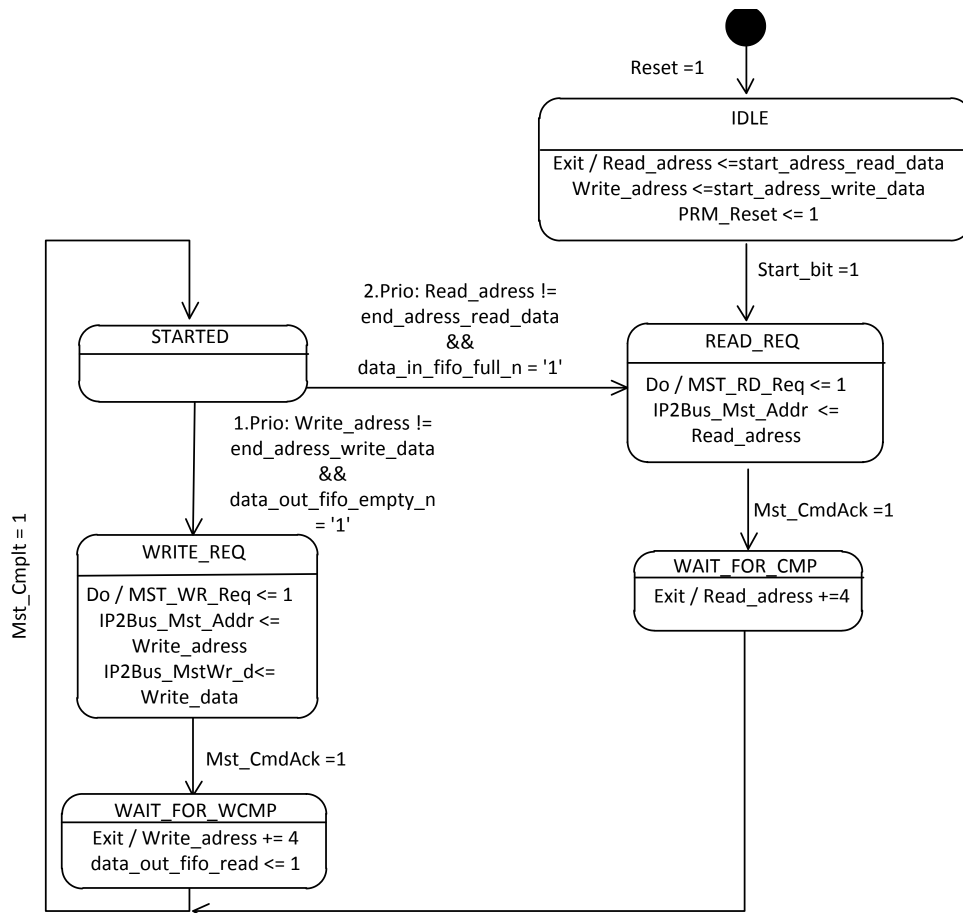


Abb. 41: *Zustandsdiagramm des Dyn_Interface IPs zur Ansteuerung des PLB (vgl. Anhang F)*

- IDLE: Beim Setzen des Start-Bits durch die Anwender Software, werden die bereits in die jeweiligen Register gespeicherten Start-Adressen in die Zähler übernommen und die PRM zurückgesetzt, um den definierten Anfangszustand der PRM zu erhalten.
- READ_REQ: Die zu lesende Adresse wird an den Adress-Bus *IP2Bus_Mst_Addr* gelegt und die Lese-Anfrage über das setzen des *MST_RD_REQ* gestartet. Bei einem akzeptieren der Anfrage vom PLB-Master wird das *MST_RD_REQ* zurück auf 0 gesetzt.
- WAIT_FOR_CMP: Wird das angeforderte Datenword vom PLB bereitgestellt, erhöht der Zähler die Lese-Adresse um vier, dies erfolgt, da der PLB Byteweise adressiert, wodurch bei einem Erhöhen um eins, vier mal das gleiche Datenwort am Eingang anstehen würde. Die gelesenen Daten werden im Eingangs-FiFo gespeichert.

[3]Die Darstellung des Mealy-Automat erfolgt als UML-Zustandsdiagramm, um eine einheitliche Darstellung zu erreichen. Die zu den Eingangssignalen gehörigen Ausgaben werden als „Exit“ Funktion angegeben und bei mehreren möglichen Übergängen mit einem Guard gekennzeichnet, wenn die Aktion nur bei bestimmten Übergängen auftritt.

- STARTED: Ist bereits ein Lese-Vorgang bereits abgeschlossen, werden die Daten gelesen und geschrieben bis die jeweiligen End-Zählerstände erreicht sind. Hierbei erfolgt eine Priorisierung des Schreibbefehls um ein vollständiges Füllen des Ausgangs-FiFos zu vermeiden.
- WRITE_REQ: Ist ein Datenwort im Ausgangs-FiFo vorhanden, wird dieses an den Ausgangs-Daten-Bus der *User_Logic* gelegt und der jeweilige Zählerstand des Schreibzählers an den Adress-Bus. Die Schreibanfrage wird über das Setzen des *MST_WR_Req* gestartet und beim Akzeptieren des PLB Busses zurückgesetzt.
- WAIT_FOR_WCMP: Beim erfolgreichen Schreiben der Daten in den Speicher, wird die Schreibadresse erhöht und die Daten aus dem FiFo gelöscht.

5.3 Definition des dynamischen Bereiches

Die Größe und Position des dynamischen Bereiches wird mit einem AREA_GROUP Constraints in dem PlanAhead-Projekt zugehörigem UCF definiert. Diesem wird im ersten Schritt die Instanz zugewiesen, dessen Position von dem Constraint festgelegt wird. Für jede verwendete Ressource wird diese Eingrenzung durchgeführt und erzeugt ein Rechteck zwischen dem Start- und dem Endpunkt, so wird in Listing 5 für die AREA_GROUP *pblock_dyn_interface_0_USER_LOGIC_I_PRR* die Slices zwischen den Koordinaten X0Y0 und X57Y239 definiert (vgl. Abbildung 42).

Listing 5: *AREA_GROUP des dynamischen Bereiches im SoC-Client*

```
0 INST "dyn_interface_0/dyn_interface_0/USER_LOGIC_I/PRR" AREA_GROUP = "
      pblock_dyn_interface_0_USER_LOGIC_I_PRR";
AREA_GROUP "pblock_dyn_interface_0_USER_LOGIC_I_PRR" RANGE=SLICE_X0Y0:SLICE_X57Y239;
AREA_GROUP "pblock_dyn_interface_0_USER_LOGIC_I_PRR" RANGE=RAMB18_X0Y0:RAMB18_X3Y95;
AREA_GROUP "pblock_dyn_interface_0_USER_LOGIC_I_PRR" RANGE=RAMB36_X0Y0:RAMB36_X3Y47;
```

Das hier erzeugte Constraint ist beschränkt auf die Verwendung von Slices und RAMB Elementen. Dies erfolgt, da DSP Elemente im MPMC verwendet werden, welche eine festgelegte Position in den verwendeten Clock-Regions besitzen und nicht in der PRR liegen dürfen, da sonst bei dem „Place and Route“ Prozess ein Fehler erzeugt wird, welcher ein Erstellen der Bitfiles verhindert. Dem dynamischen Bereich stehen so die in Tabelle 10 dargestellten Ressourcen zur Verfügung. Die zur Verfügung stehenden SLICEL Elemente sind hierbei Slices die ausschließlich für Logik verwendeten werden. SLICEM Elemente können dagegen auch als Speicher verwendet werden, wie beispielsweise als Schiebe-Register ohne dabei die FLipFlops im Slice zu verwenden [Xilinx (2010h)].

Tabelle 10: *Ressourcen des dynamischen SoC Systems*

Ressource	Zur Verfügung
LUT	55680
FD_LD	111360
SLICEL	7440
SLICEM	6480
RAMBFIFO36E1	192

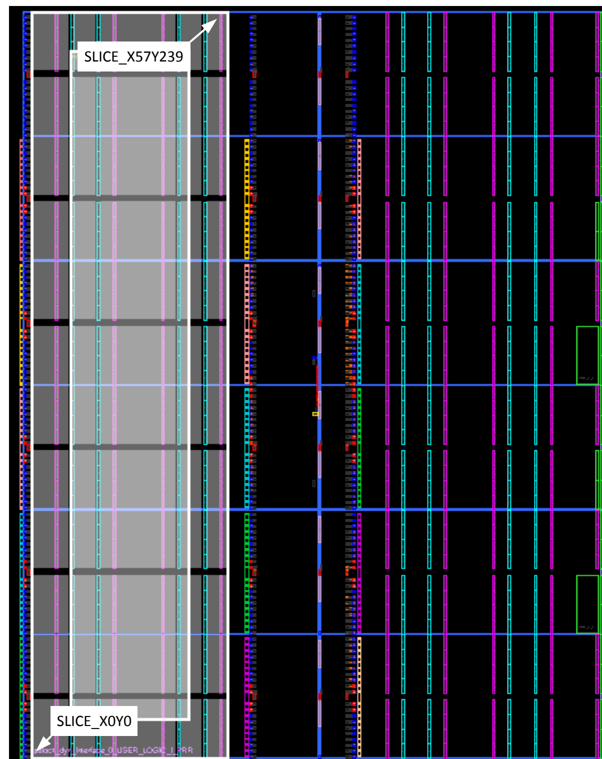


Abb. 42: *Dynamischer Bereich des SoC-Clients als grafische Darstellung im PlanAhead*

Das Erzeugen der verwendbaren Netzlisten der PRMs erfolgt unter der Verwendung der ISE Software. Vor der Synthese sind die folgenden Einstellungen zum Synthese Prozess zu treffen, so dass die Implementierung allein auf Basis der zur Verfügung stehenden Ressourcen erfolgt.

- dsp_utilization_ration: 0
- use_dsp48 : false
- iobuf : false

Beim Erstellen des Bitfiles mit einer AREA_GROUP werden die Ressourcen auf die definierten Bereiche verteilt. Mit dem FPGA-Editor sind die erstellten Design-Files grafisch darstellbar und zeigen, dass die Module in den jeweiligen Bereiche platziert werden. So werden in Abbildung 43 die Ressourcen auf dem kompletten FPGA verteilt (rot), in Abbildung 44 wird das statische System allein auf der rechten Seite platziert. Der gelbe Kreis zeigt die Ressourcen, die in der PRR von der PRM belegt sind.

5.4 Software Client des statischen SoC Systems

Die Software des SoCs empfängt die PRMs, konfiguriert die PRR und verarbeitet die Datensätze, die vom Broker übertragen werden. Dieses findet auf der Basis des Xilkernels statt, welcher die Software Komponenten steuert. Der Xilkernel stellt den Scheduler des Systems zur Verfügung, welcher für das LwIP erforderlich ist und ebenfalls zum Verarbeiten und Versenden der Datensätze genutzt wird (vgl. Kapitel 3.9). Weiter stellt er Software-Mutexe und Semaphoren zur Verfügung, welche zur Synchronisation der einzelnen Tasks in der SoC-Client Software genutzt werden. Konfiguriert ist der Xilkernel im Prioritätsmodus, in dem der Thread mit der

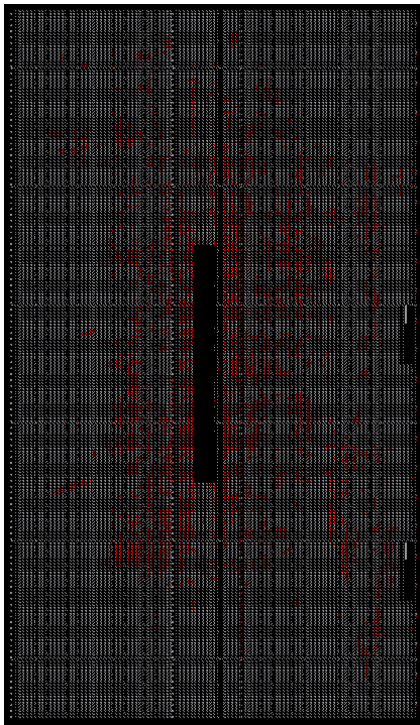


Abb. 43: *Verteilung der genutzten Ressourcen im FPGA ohne definierter PRR, dargestellt im FPGA-Editor*

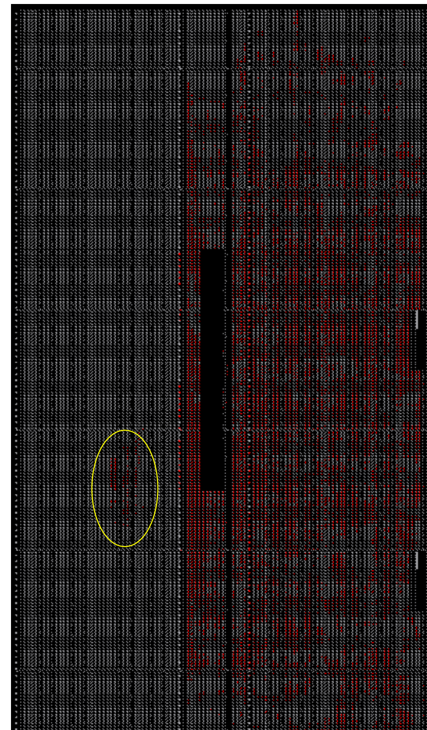


Abb. 44: *Verteilung der genutzten Ressourcen im FPGA mit definierter PRR und einer aktiven PRM (gelber Kreis)*

höchsten Priorität bearbeitet wird, während die anderen im Ready oder Wait Zustand sind. Weiter wird der Xilkernel so konfiguriert, dass Semaphoren und Mutexe verwendet werden. Die weiteren Konfigurationen sind in Listing 6 dargestellt.

Listing 6: *Parameter im MSS File des BSP der SoC Software (vgl. Anhang 16)*

```
0


BEGIN OS
 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 5.00.a
5 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
 PARAMETER SYSTMR_SPEC = true
 PARAMETER SYSTMR_DEV = xps_timer_0
10 PARAMETER SYSINTC_SPEC = xps_intc_0
 PARAMETER SCHED_TYPE = SCHED_PRIO
 PARAMETER PTHREAD_STACK_SIZE = 4096
 PARAMETER ENHANCED_FEATURES = true
 PARAMETER CONFIG_KILL = true
15 PARAMETER CONFIG_YIELD = true
 PARAMETER CONFIG_SEMA = true
 PARAMETER CONFIG_TIME = true
 PARAMETER CONFIG_DEBUG_SUPPORT = true
 PARAMETER VERBOSE = true
20 PARAMETER DEBUG_MON = true
 PARAMETER CONFIG_PTHREAD_MUTEX = true
 PARAMETER MAX_PTHREADS = 15
 PARAMETER MAX_PTHREAD_MUTEX = 15
 PARAMETER MAX_PTHREAD_MUTEX_WAITQ = 15
25 PARAMETER STATIC_PTHREAD_TABLE = ((main_thread ,1))
END


BEGIN LIBRARY
30 PARAMETER LIBRARY_NAME = lwip130
 PARAMETER LIBRARY_VER = 2.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
```

```
 PARAMETER API_MODE = SOCKET_API
 PARAMETER LWIP_DHCP = true
 PARAMETER DHCP_DOES_ARP_CHECK = true
END

BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = xilmfs
 PARAMETER LIBRARY_VER = 1.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
END
```

Das Prioritäts-Scheduling wird verwendet, um eine zeitnahe Bearbeitung der Eingangsdaten zu gewährleisten, wobei hierbei eine Soft-Real-Time Bedingung vorliegt, da keine Beschädigung des Systems erfolgt, wenn die Daten nicht rechtzeitig verarbeiten werden. Im worst-case erfolgt eine erneute Übertragung der Daten, was allein zu Performance Einbrüchen führt [Buttazzo (2005)]. So werden die Threads, die nicht zum LwIP gehören, mit einer Priorität gestartet, die niedriger ist, als die der Netzwerk Threads. Insgesamt sind nach der Initialisierungsphase, die folgenden Threads vom Xilkernel zu verwalten:

- LwIP: Im LwIP werden die ankommenden TCP/IP Daten-Pakete verarbeitet und den weiteren Threads die enthaltenen Daten übergeben. Weiter werden die zu sendenden Daten zum Versenden vorbereitet und dem Ethernet-Modul übergeben.
- SoC-Client: Der SoC-Client Task initialisiert das System beim Starten und wartet anschließend auf Anfragen vom Broker. Beim Erhalten einer Anfrage zum Datentransfer wird entweder die Funktion für die PRM oder eines Datensatzes aufgerufen bis die Daten vollständig übertragen sind.
- Send_Data: Dieser Task überprüft das Array zum Verwalten der vorhandenen Datensätze und sendet die verarbeiteten Datensätze an den Broker.
- Compute_Data: Der Compute_Data Tasks konfiguriert das *DYN_Interface* zum Verarbeiten der Datensätze und Markiert bereits verarbeitete Daten für den Versand zum Broker.

Der Konfigurationseintrag *STATIC_PTHREAD_TABLE* erlaubt das Erstellen eines Threads, der beim Starten des Xilkernels aufgerufen wird, ohne diesen vorher im Code zu erstellen. So wird der *main_thread* gestartet, sobald der Xilkernel initialisiert ist. Dieser enthält die Initialisierung der verwendeten LwIP Bibliothek und der Netzwerk Schnittstelle (vgl. Abbildung 45).

Die Adressierung des Netzwerk Interfaces erfolgt je nach Infrastruktur, so besteht die Wahl zwischen einer festen IP oder einer Konfiguration über das „Dynamic Host Configuration Protocol" (DHCP), wobei die Auswahl der Konfigrationsmethode beim Kompilieren der Software über *#define _DHCP_* gesetzt wird (vgl. Kapitel 3.4 und 3.9). Ist die Vergabe der Adresse abgeschlossen, so werden die Initialisierungen des SoC-Clients gestartet. Diese beinhalten zum einen die Initialisierung des Dateisystems (vgl. Kapitel 5.4.1) zum anderen die der Ringbuffer zum Speichern der Ein- und Ausgangsdaten (vgl. Kapitel 5.4.2) und der Mutexe zum Synchronisieren der Zugriffe auf die Ringbuffer. Ist dies durchlaufen, wird die Verbindung zum Broker aufgebaut und die ankommenden Daten verarbeitet (vgl. Abbildung 45).

Wird das Signal zur Übetragung einer PRM empfangen, so wird die bereits existierende Datei im Dateisystem gelöscht und eine neue Datei erstellt (vgl. Kapitel 4.1). Daraufhin werden die Daten in die Datei geschrieben und bei einer erfolgreichen Übertragung, die Antwort an den Server gesendet. Zur anschließenden Rekonfiguration der PRR werden die Interrupts des MicroBlazes abgeschaltet. Dies erfolgt, da eine Unterbrechung der Konfiguration in einigen Fällen zum Stillstand des Systems führt. Ist die Rekonfiguration ebenfalls erfolgreich durchgeführt

worden, werden die Interrupts wieder eingeschaltet, so dass das Scheduling im Xilkernel, sowie die Verarbeitung der Ethernet Daten wieder aufgenommen werden (vgl. Abbildung 46).

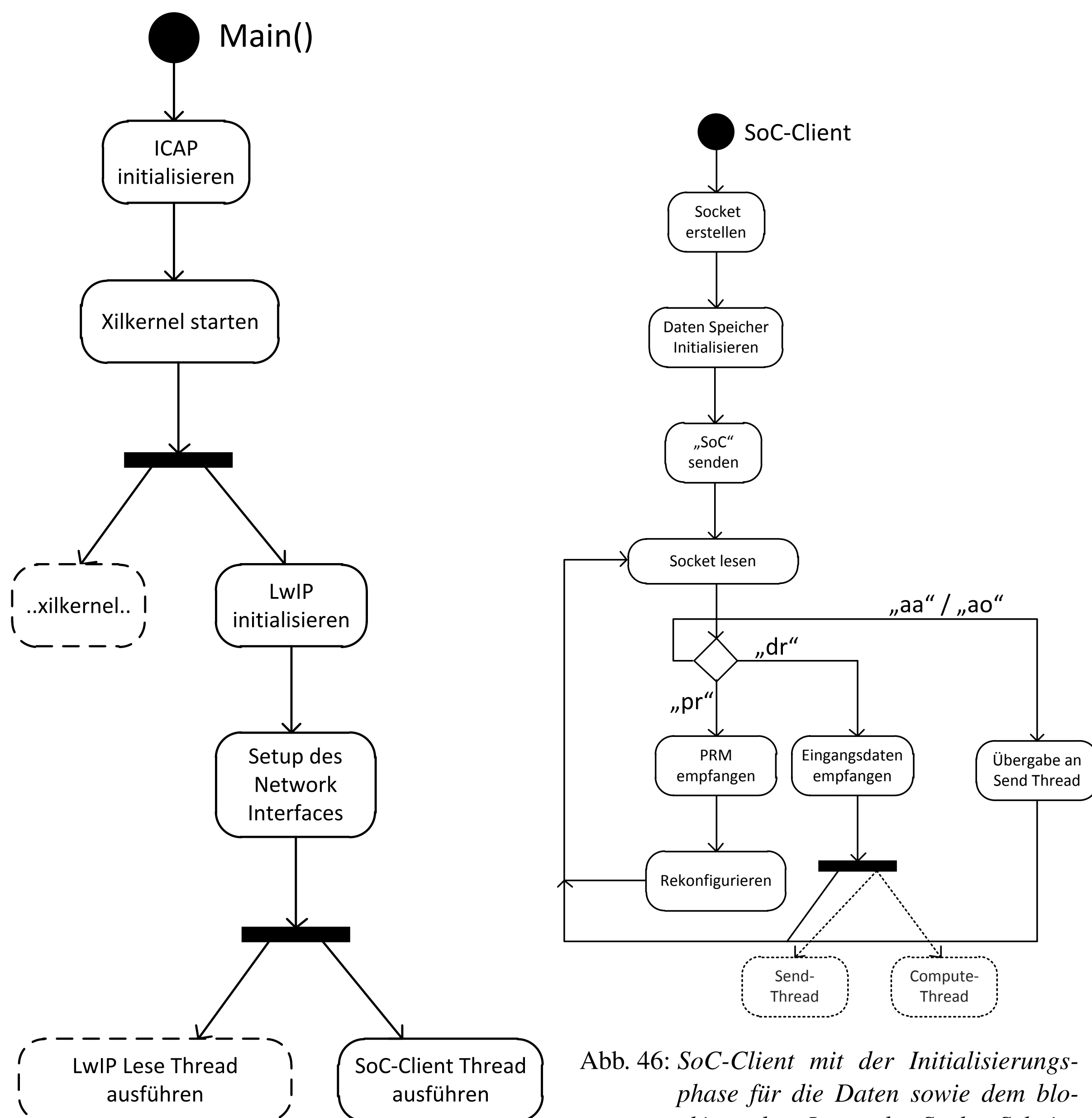


Abb. 45: *Start des SoC-Clients mit der Initialisierung des Xilkernels und der anschließenden Adressierung des Ethernet Interfaces sowie dem Starten des SoC-Client Threads und dem LwIP.*

Abb. 46: *SoC-Client mit der Initialisierungsphase für die Daten sowie dem blockierendem Lesen der Socket Schnittstelle und dem warten auf Anfrage mit dem in Kapitel 4.1 definiertem Protokoll.*

Beim Empfangen des Signals zum Starten einer Eingangsdaten Übertragung, wird das Verwaltungsarray durchsucht und der noch zur Verfügung stehende Speicherplatz berechnet (vgl. Kapitel 4.1). Steht der angeforderte Speicherplatz zur Verfügung, wird die jeweilige Start- und End-Adresse in das Verwaltungsarray geschrieben und die Daten in den Speicher geladen. Beim Empfangen des ersten Datensatzes, werden die Threads zum Bearbeiten der Daten sowie der zum Senden der Daten an den Server gestartet. Zur Vermeidung der Verarbeitung von nicht voll-

ständig übertragenen Datensätzen, ist das Verwaltungsarray über zwei Mutexe gesichert (vgl. Abbildung 47, 48 ,49, sowie Kapitel 5.4.2).

- *entry_mutex*: Dieser Mutex wird zur Synchronisation zwischen dem Receive- sowie dem Compute-Thread verwendet. Beim Empfangen eines neuen Datensatzes wird der Mutex gesperrt, bis der Datensatz übertragen ist. Dies erfolgt ebenfalls, wenn der *compute_data* Thread das Verwaltungsarray nach einem zu berechnendem Datensatz durchsucht.
- *computed_mutex*: Dieser Mutex wird vom *compute_data* Thread gesperrt, sobald dieser gestartet wird und den Status des *DYN_Interface* abfragt. Der *send_data* Thread sperrt ebenfalls den Zugriff, während die Daten versendet werden.

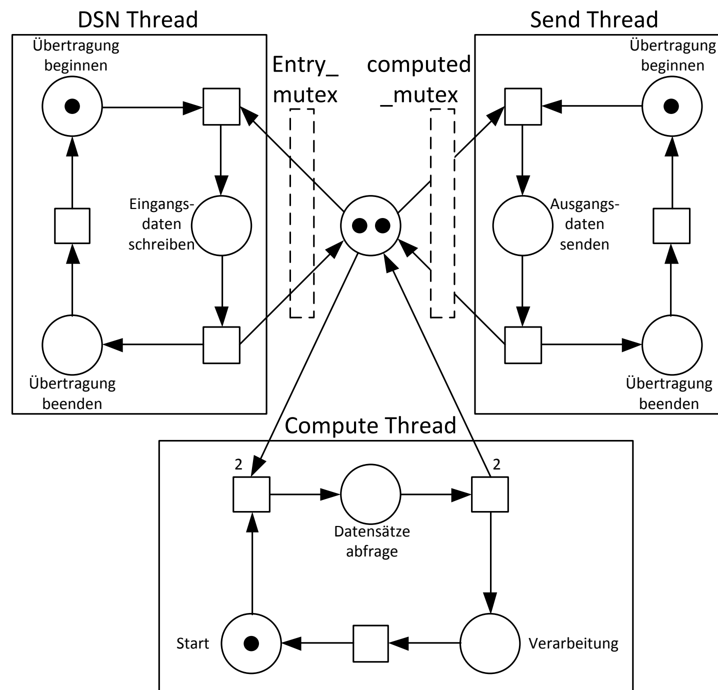


Abb. 47: *Petri-Netz Beschreibung des gegenseitigen Ausschlusses zur Synchronisation der Datensätze sowie der Zugriffe auf das Verwaltungsarray beim Empfangen der Daten im SoC-Client sowie der Verarbeitung und dem anschließenden Versenden*

5.4.1 Dateisystem zur Speicherung der PRM-Dateien im SoC-Client

Zur Verwaltung der empfangenen PRM-Dateien wird ein Dateisystem verwendet. Zur Auswahl stehen hierfür im SDK das „LibXil FATFile System“ (FATFS) und das „LibXil Memory File System“ (MFS). Diese werden von Xilinx bereitgestellt und sind für den Standalone und Xilkernel Betrieb vorbereitet. Sie unterscheiden sich wie in Tabelle 11 dargestellt.

Im SoC-Client findet das MFS Verwendung, da dieses eine unbegrenzte Anzahl an Schreibzugriffen zulässt und einen höheren Datendurchsatz hat (vgl. Anhang C). Zwar stehen durch die Verwendung des DDR3 Speichers die Daten nach einem Power-Off nicht mehr zur Verfügung, dieses wird aber durch das Abrufen einer neuen PRM beim Starten des Systems abgefangen.

Zur Verwendung des MFS wird ein zusammenhängender Speicherbereich im DDR3-Speicher verwendet. Dieser kann zum einen durch das Anlegen eines Arrays als Variable, zum anderen durch das Erstellen einer Section im Linker-Script erzeugt werden. In diesem Fall wird die

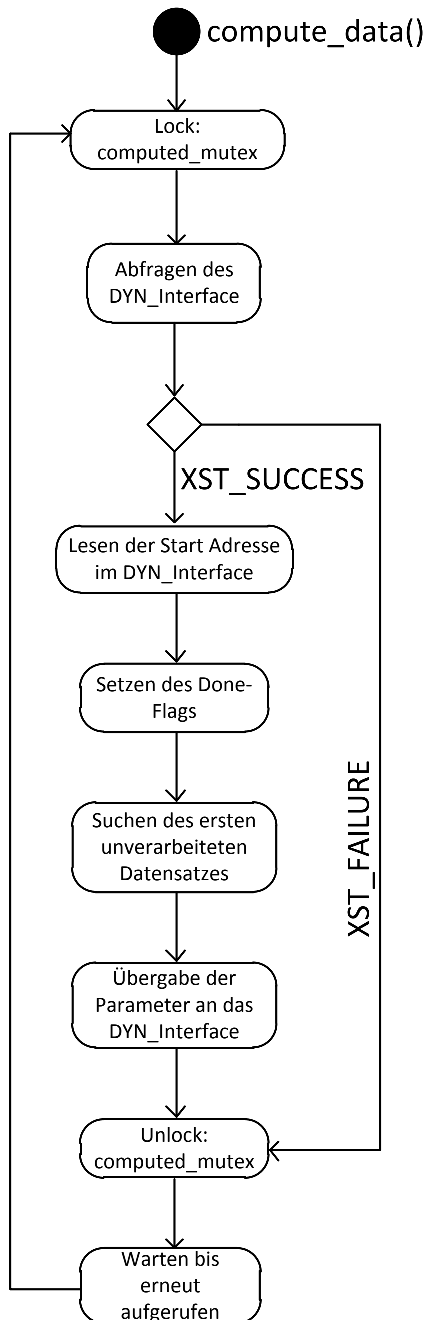


Abb. 48: *Compute-Thread zum Verarbeiten der Daten mit der Synchronisation über die Mutexe und der Interaktion mit dem DYN_Interface.*

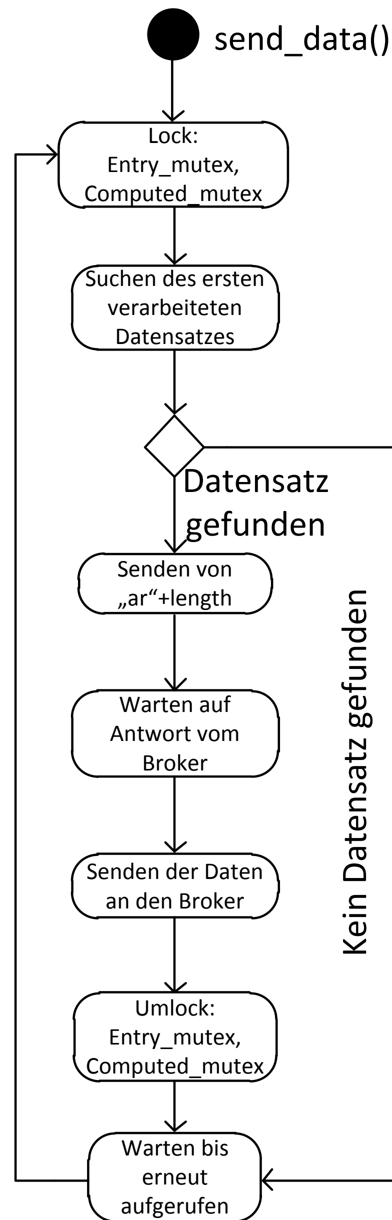


Abb. 49: *Send-Thread zum Versenden der bereits verarbeiteten Daten*

Tabelle 11: *Eigenschaften der zur Verfügung stehenden Dateisysteme [Xilinx (2011d)][Xilinx (2011c)]*

Eigenschaft	FATFS	MFS
Speicherort	die Daten werden auf der Compact-Flash Karte des ML605 gespeichert	Speichert die Daten im B-RAM oder über den MPMC in einem externen Speicher
Persistenz	nach einem Ausschalten des Systems, stehen die Daten weiter zur Verfügung	die Daten stehen bis zum Ausschalten des Systems zur Verfügung.
Kapazität	es steht die Kapazität der Compact-Flash Karte bereit. Diese beträgt 2GB und enthält ebenfalls die Dateien zum Programmieren des FPGAs bei Power-On.	durch die Verwendung des externen DDR3-Speichers als Umgebung zum Ausführen der Programme, steht nicht der gesamte Speicher von 256 MB zur Verfügung
Anzahl der Zugriffe	durch die Flash Technologie sind ca. 100000 Zugriffe möglich, bis der Speicher unbrauchbar wird.	bei dem B-RAM und DDR sind eine unbegrenzte Anzahl an Zugriffen erlaubt.

_filesection erzeugt und ihre Größe auf 10 MB festgelegt (vgl. Listing 7). Diese Größe ist ausreichend, da eine komplette Konfiguration des XC6VLX240T mit dem statischen System 9 MB beträgt und diese bereits eine PRM enthält .

Listing 7: *Linker Script Section zur Speicherung der PRM*

```
0 .filesection : {
     _filesection_start = .; // Anfang der Filesection
     *(.filesection)
      . += 10000000;                     //Festlegen der Größe der Section auf 10 MB
     _filesection_end = .;   // Ende der Filesection
5 } > DDR3_SDRAM_MPMC_BASEADDR
```

Durch das automatische Generieren der Variable *_filesection_start* im Linkerscript, ist dem Programm bei der Kompilation die Startadresse der Section bekannt. Sie wird in einer Header Datei als externe Variable deklariert und deren Adresse als Pointer auf die erste Adresse genutzt. Dieses Verfahren wird ebenfalls auf die Variable am Ende der Section angewandt. Im Anschluss wird die Adresse der Start-Variablen von der Adresse der End-Variablen subtrahiert, wodurch die Länge der *filesection* berechnet wird, so dass diese bei einer Änderung der *filesection* direkt zur Verfügung steht (vgl. Listing 8, Zeile 0 - 3).

Vor der Verwendung des XILMFS ist dieses zu Initialisieren, wobei die Anfangsadresse des Bereiches übergeben wird, der als Dateisystem zu verwenden ist. Weiter wird die Größe übergeben und ein Parameter zur Art der Initialisierung. Diese sind [Xilinx (2011d)]:

- MFSINIT_NEW :Das Dateisystem wird neu erzeugt und alle Daten gehen verloren. Diese Parameter wird hier verwendet, da keine Daten aus einem Image zu laden sind und die zu verwendenden PRMs beim Anmelden an den Broker neu übertragen werden (vgl. Listing 8, Zeile 6).
- MFSINIT_IMAGE : Initialisiert das System mit Daten, die vorher an die entsprechende Adresse geschrieben wurden.

- MFSINIT_ROM_IMAGE : Initialisiert das System als „Read Only“ mit Daten, die vorher an die entsprechende Adresse geschrieben wurden.

Ist die Initialisierung abgeschlossen, so stehen die Funktionen des Dateisystems zur Verfügung. Diese sind beispielsweise:

- int mfs_file_open(char *filename, int mode): Diese Funktion öffnet die Datei zum Lesen. Über den Rückgabewert, wird die Datei bei ihrer weiteren Verwendung identifiziert. Je nach übergebenem *mode* Parameter, wird die Datei zum Lesen (MODE_READ) oder Schreiben (MODE_WRITE) geöffnet oder neu angelegt (MODE_CREATE).
- int mfs_file_write(int fd, char *buf, int buflen): Schreibt die Daten aus dem Puffer an das Ende der Datei.
- int mfs_file_close(int fd): Die Datei wird geschlossen und es sind keine weiteren Operationen auf dieser anwendbar, bis sie erneut geöffnet wird.

Listing 8: *Intialisierung und Verwendung des XILMFS mit Section Variablen (Xilinx (2011d))*

```
extern int _filesection_start;
//Referenz auf die im Linkerscript erstellte Section
extern int _filesection_end;
#define FILE_SECTION_LENGTH (&_filesection_end - &_filesection_start)
//Berechnen der Section länge
..
mfs_init_fs(FILE_SECTION_LENGTH, &_filesection_start, MFSINIT_NEW);
//Initialisieren mit einem leeren Dateisystem
..
file_descriptor = mfs_file_open(file_name, MFS_MODE_CREATE);
//Erzeugen einer Datei
..
mfs_file_write(file_descriptor, recv_buffer, chars_2_write);
//Schreiben von Daten
..
mfs_file_close(file_descriptor);
//Schließen der Datei
```

5.4.2 Speicherung der Ein-und Ausgangsdateien im SoC-Client

Die Ein- und Ausgangsdaten der Projekte werden in je einer eigenen Section gespeichert. Die Verwaltung der Daten wird, anders als bei den PRM-Dateien, über ein Verwaltungsarray geregelt, welches die Adressen der Ein- und Ausgangsdaten enthält. So stehen die Parameter für das *dyn_interface* direkt zur Verfügung und nicht in der „File-Alocation-Table“ (FAT) gespeichert, wie es im XILMFS der Fall ist und welche nicht von außen sichtbar ist ([Microsoft (2000)],[Xilinx (2011d)]). Gehalten werden die Daten in dem *data_position_t* Struct, welches die Größe der jeweiligen Daten und deren Start und Ziel Adresse beinhaltet (vgl. Listing 9).

Listing 9: *Struct zur Verwaltung der Ein- und Ausgangsdaten*

```
typedef struct {
    int input_length; // Länge der Eingangsdaten
    int entry_start_adress; //Start Adresse der Eingangsdaten
    int entry_end_adress; //End Adresse der Eingangsdaten
    int output_length; //Länge der Ausgangsdaten
    int computed_start_adress; //Start Adresse der Ausgangsdaten
    int computed_end_adress; //End Adresse der Ausgangsdaten
    char done; // Flag zum Anzeigen, ob die Daten berechnet wurden
} data_position_t;

static data_position_t data_position[10];
```

Das Verwaltungs-Array *data_position* beinhaltet die Daten der einzelnen Ein- und Ausgangsdaten. Durch dieses ist die Größe, der zu verwaltenden Daten, auf zehn Einträge reduziert. Diese werden in der *.entrydatasection* zum Speichern der Eingangsdaten und der *.computeddatasection*, für die verarbeiteten Daten gehalten. Die Verwaltung der Daten ist in Abbildung 50 dargestellt.

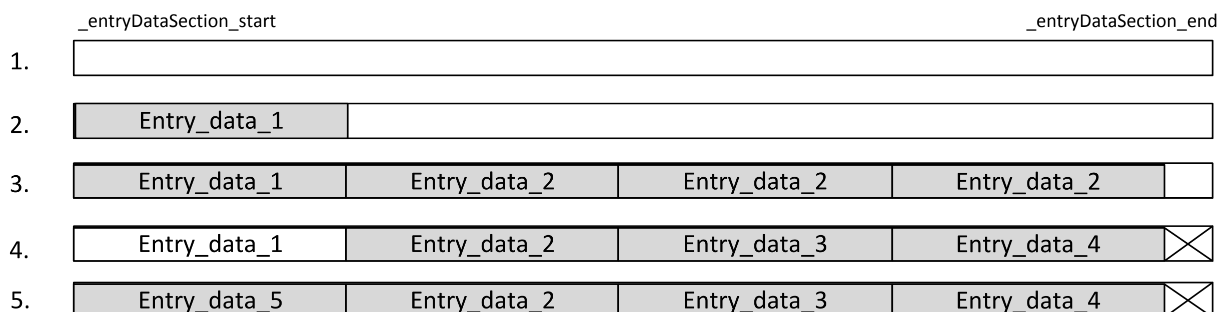


Abb. 50: *Datenverwaltung im SoC-Client für die Ein- und Ausgangsdaten im Beispiel mit fünf Datensätzen und einer jeweiligen Größe von 12 MByte*

1. Nach der Initialisierung des Programms ist der Bereich für die Daten leer und das Verwaltungs Array beeinhaltet keine Werte.
2. Wird der erste Datensatz erhalten, so wird ein Bereich mit dessen Größe reserviert und die Daten an die entsprechende Adresse geschrieben. Die ersten sechs Bytes des Eingangsdatensatzes bestehen aus einem Integer und den Zeichen „ol“, welche die Größe des Ausgangsdatensatzes anzeigen. Diese Größe wird in das *output_length* Feld des Verwaltungs-Arrays geschrieben und zum Reservieren eines Speichersplatzes für die Ausgabe verwendet.
3. Sendet der Server weitere Datensätze, werden sie nacheinander in den reservierten Speicherbereich geschrieben. Dieses erfolgt bis nicht mehr genug Speicher für einen weiteren Datensatz zur Verfügung steht. Ist diese Grenze erreicht, so wird jede weitere Anfrage abgewiesen, bis erneut Speicherplatz bereit steht.
4. Steht ein Datensatz vollständig zur Verfügung, wird ein Bereich für die Ausgangsdaten in der *computedSection* reserviert. Die jeweiligen Start- und End-Adressen für Ein- und Ausgangsdaten, werden in die Register des *dyn_interface* geschrieben und die Daten von der PRM berechnet. Sobald das Status Register des *dyn_interface* anzeigt, dass die Berechnung durchgeführt ist, werden die Daten vom *send_data* Thread an den Server gesendet und anschließend der Eintrag im Verwaltungs-Array gelöscht.
5. Durch das Freigeben des Speichers steht erneut ein passender Bereich bereit, der für einen neuen Datensatz verwendet wird.

5.5 Initialisierung der SoC Konfiguration und der Client Software

Der Konfigurationsspeicher des verwendeten FPGAs ist nicht konsistent, wodurch bei einer Abschaltung der Betriebsspannung die Konfiguration verloren geht und dieser erneut zu Programmieren ist. Unter der Verwendung des SYSACE IPs ist der FPGA über Dateien auf einer CF-Card programmierbar. Diese Dateien bestehen beispielsweise aus den Slice Konfigurationen sowie der Daten im BRAM. Programme, die in externen Speichern gehalten werden, sind über

einen Bootloader aus einem persistenten Speicher zu lesen, da diese nicht über das SYSACE geschrieben werden können.

Xilinx stellt mit dem *srec_bootloader* Programm einen Bootloader zur Verfügung, der SREC Dateien aus dem Flash Speicher liest und sie in den, in der Datei angegebenen, Speicher schreibt und ihn anschließend ausführt. Dieser ist so anzupassen, dass die Start-Adresse der Datei übergeben wird. Die SREC Datei wird mit dem Flash Memory Tool des Xilinx SDK erzeugt, welches als Eingangs-Parameter die „Executable and Linkable Format“ (ELF) Datei des SoC-Clients hat. Diese Datei enthält den kompilierten Programm-Code und wird beim Erstellen des Programms im SDK erzeugt. Zu markieren ist die Einstellung „Convert ELF to SREC“ und die Speicheradressen des Programms im Flash Speicher und im ausführenden Speicher (vgl. Abbildung 51). Werden diese Einstellungen bestätigt, schreibt das Tool die Datei in den Flash Speicher, welche nun dauerhaft zur Verfügung steht.

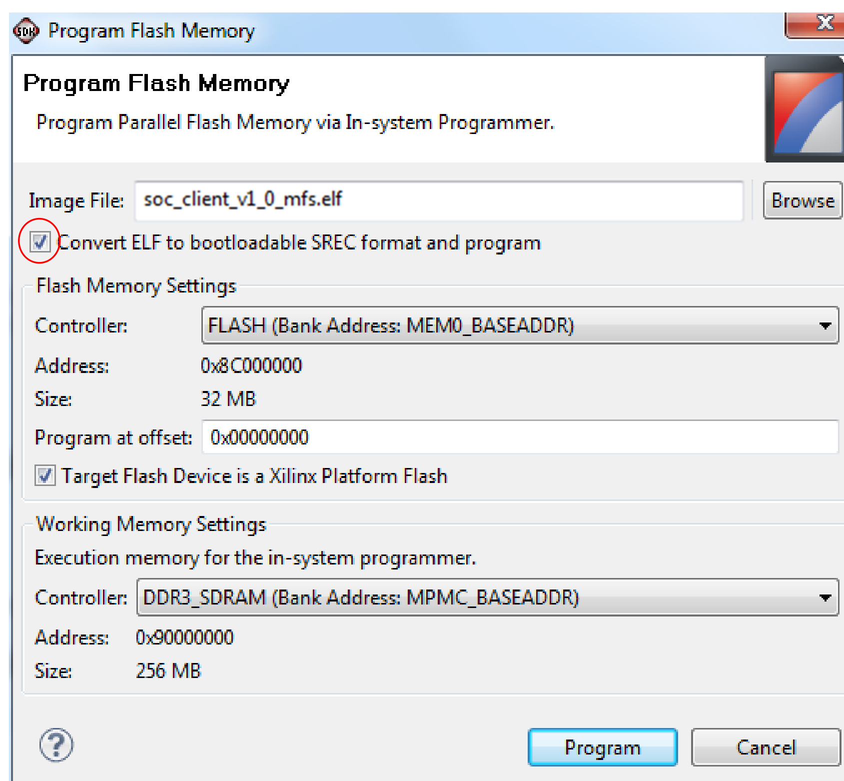


Abb. 51: *Erzeugen einer SREC Datei mit dem Flash Memory Tool des Xilinx SDK*

Das Erstellen einer Datei zur Konfiguration erfolgt auf der Basis der BIT-Datei aus PlanAhead und der ELF-Datei, welches den kompilierten Bootloader enthält. Um eine Konfigurationsdatei zu erstellen, die beim Starten bereits ein Programm ausführt, ist diese mit dem Programm DATA2MEM zu erzeugen. Dieses wird über das „ISE Design Suite Command Prompt“ aufgerufen und mit folgenden Parametern gestartet [Xilinx (2011b)]:

$$data2mem\ -bm\ system.bmm\ -bd\ bootloader.elf\ -bt\ system.bit\ -o\ b\ SoCClient.bit$$

- -bm : Gibt an, dass der folgende Wert auf eine .bmm Datei zeigt. Wird die Endung vom Anwender weggelassen, so wird sie automatisch angehängt.
- -bd : Das nächste Kommando beinhaltet den Namen einer .elf oder .mem Datei. Durch die Verwendung des TAGNAME Befehls kann die übergebene Datei gefiltert werden und nur bestimmt Adress-Bereiche werden übernommen.

- -o b new.bit: Der Befehl -o gibt an, welche Art von Ausgangsdatei erzeugt werden soll. Zur Verfügung stehen hier die Parameter:
  - u = UCF
  - v = Verilog
  - h = VHDL
  - b = BIT
  - p = BMM
  - d = DMP

  Anschließend folgt der Namen der Ausgangsdatei, hier *SoCClient.bit*, welche die FPGA Konfiguration mit dem Bootloader enthält.

Mit der so erstellten Datei wird anschließend die SYSACE Datei erzeugt. Dieses erfolgt über das IMPACT Programm, welches zur Konfiguration von XILINX FPGAs verwendet wird. Zum Erstellen einer SYSACE Datei für das ML605 wird das IMPACT Programm gestartet und ein bereits vorhandenes ML605 Projekt geöffnet oder ein neues erstellt, welches mit der Initialisierung des JTAG konfiguriert wird. Die Datei wird daraufhin mit den folgenden Parametern erzeugt:

- Operating Mode: Novice
- Size: Generic
- Reserve Space: 0
- Configuration Adress 0: Aktiviert

Anschließend wird die mit dem data2mem erstellte Datei zugewiesen und die Datei mit einem Rechtsklick auf den „xc6vlx240t“ und „Generate File“ erzeugt. Als Ausgangsdateien werden zum einen die Datei *xilinx.sys* zum anderen ein Ordner mit dem Projektnamen erstellt. Diese Dateien sind auf die CF-Karte zu kopieren, welche anschließend wieder in den Kartenleser des ML605 einzustecken ist.

Die Konfiguration des FPGAs auf dem Entwicklungsboard wird über die DIP Schalter S1 und S2 kontrolliert und so eingestellt, dass die Schalter S1.4, S2.2, S2.4 und S2.5 auf ON sind, wodurch die Konfiguration per SYSACE und CF-Karte aktiviert wird [Xilinx (2011h)]. Nach dem Anschalten des ML605 ist der SYSACE_RESET Knopf zu betätigen, worauf anschließend der FPGA mit den erstellten Dateien konfiguriert wird.

Bei der Konfiguration des FPGAs über das SYSACE sowie über IMPACT und dem erstmaligen Ausführen der Client-Software, wird das HWICAP nicht vollständig initialisiert, welches zu einem Stoppen bei der Rekonfiguration führt. Dieses Verhalten wird durch den Reset des HWICAP, bei einer nicht vollständigen Initialisierung, verhindert. So ist das Kontroll-Register des HWICAP mit der Reset Maske zu beschreiben, da hier die Verwendung der Reset Funktion ebenfalls einen Stopp der Software auslöst (vgl. Listing 10). Diese Änderung der Initialisierungsfunktion lässt den SoC-Client erfolgreich Ausführen.

Listing 10: *Reset des HWICAP bei der Initialisierung zur Konfiguration mit dem SystemAce*

```
main()
{...
   do{
       status = init_icap();
   }
   while(status != XST_SUCCESS);
...}

init_icap()
{...
  Status = XHwIcap_CfgInitialize(&HwIcap, ConfigPtr,     ConfigPtr->BaseAddress);
  if (Status != XST_SUCCESS) {
    XHwIcap_WriteReg(HwIcap.HwIcapConfig.BaseAddress, XHI_CR_OFFSET, XHI_CR_SW_RESET_MASK);
    XHwIcap_WriteReg(HwIcap.HwIcapConfig.BaseAddress, XHI_CR_OFFSET, ~ XHI_CR_SW_RESET_MASK);
    return XST_FAILURE;
  }

...}
```

6 Broker zur Verteilung der Ressourcen implementiert mit Java

Der Broker dient den SoCs und Projekt-Clients als Single-Point of Contact (vgl. Anhang G). Beim Einschalten der SoCs sowie der Aktivierung der Projekt-Clients verbinden sich diese mit dem Server und er verteilt die Rechenleistung der SoCs an die Projekte. Die Konfigurationen sowie die Datensätze der Projekte und die daraus entstandenen Ergebnisse werden vom Server empfangen und an den jeweilige Projekt-Clienten geschickt.

Die Implementierung des Servers erfolgte in Java aufgrund der Eigenschaften zur Portabilität, des Garbage-Collectors und dem Multithreading. Aufgrund der weiten Verbreitung stehen ebenfalls viele Bibliotheken bereit, wie beispielsweise zur Verarbeitung von XML-Dateien, die die vorhandenen Funktionen erweitern oder verbessern [XStream (2011)]. Weiter wird die in Kapitel 4.3 vorgestellte Architektur verwendet und die Interfaces des Schedulers und der Factories als Basis zur Implementierung genutzt.

Die Kommunikation innerhalb des Servers ist mit dem Observer Pattern realisiert. Dieses Pattern besagt, dass eine Eins-zu-viele-Abhängigkeit zwischen Objekten besteht, in der Art dass alle abhängigen Objekte bei einer Änderung des beobachteten Objektes informiert werden ([Freeman u. a. (2005)],[Gamma u. a. (2005)]). Hierzu wird das von Java zur Verfügung gestellte Interface *Observer* sowie die abstrakte Klasse *Observable* verwendet. Bei jeder Änderung am Zustand des Objektes, wie beispielsweise der Anmeldung eines neuen SoC-Clients, wird so das höher gestellte Objekt benachrichtigt, ohne das dessen Funktionen bekannt sind. Anfragen oder Befehle an unter geordnete Elemente erfolgen über die jeweilige Funktionen der Interfaces (vgl. Abbildung 52).

6.1 Verwaltung der Clients in der Client-Factory

Die Client-Factory dient dem Erstellen und Verwalten konkreter SoC-Client Objekte. Implementiert ist diese Factory nach dem Factory-Method Idiom, welches das Erstellen von Objekten über eine Methode anstatt des Konstruktors beschreibt. Dies hat den Vorteil, dass das aufrufende Objekt von der Implementierung gekoppelt wird und bei einer Weiterentwicklung kein neuer Konstruktor oder neue Klasse bekannt zu geben ist ([Freeman u. a. (2005)],[Gamma u. a. (2005)]). Weiter ist die Client-Factory die Schnittstelle zu den konkreten SoC Verbindungen und leitet die Funktionsaufrufe des Schedulers weiter.

Die Verwaltung der einzelnen SoC-Clients erfolgt aus Basis einer HashMap (vgl. Abbildung 52). Diese ermöglicht das Finden eines Objektes anhand eines Schlüssels, hierbei wird der Name des Clients genutzt, der sich aus der IP Adresse und dem Port zusammensetzt. Dieser Name steht dem Scheduler über die SoC-Client Beschreibung, die dieser verwaltet, zur Verfügung. Wird einem Client eine Konfiguration oder Datensatz zugewiesen, so wird beispielsweise der Funktion *setCompute* der Name des Clients sowie das entsprechenden *ADataSet* übergeben und von der Client-Factory das entsprechenden Client Objekt zurückgegeben.

Abbildung 53 stellt die Interaktion mit den Clienten und dem Scheduler da:

- Wird ein Objekt der Client-Factory erstellt, so erzeugt es einen Thread mit einer Referenz auf sich selbst, damit die Client-Factory die Kommunikation mit den SoC-Clients aufbaut und einen Server Socket bereitstellt, wobei die blockierende Lese-Funktion des Sockets genutzt wird. Erfolgt der Verbindungsaufbau eines Client, so erzeugt der Server Socket eine entsprechenden Client-Socket, der die TCP Verbindung zum SoC-Client aufrecht erhält. Ist dieser Verbindung aufgebaut und der Client hat sich mit der Zeichenkette

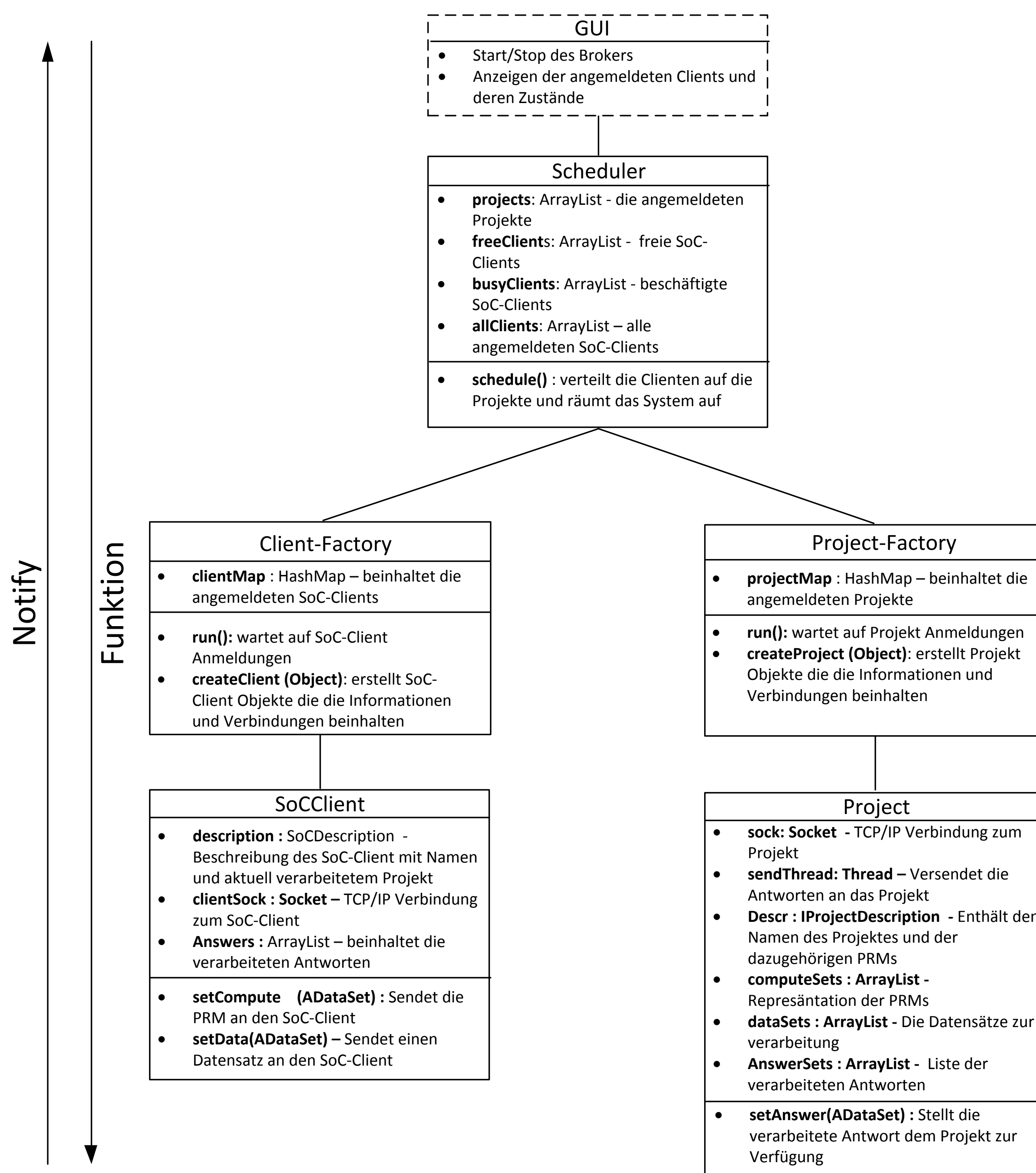


Abb. 52: *Kommunikation innerhalb des Servers über Funktionen und das Observer Pattern, sowie die Eigenschaften der einzelnen Klassen wie dem Speichern der SoC-Clienten und den wichtigsten Funktionen.*

„SoC—“ angemeldet, so wird ein neues *SoCCLient* Objekt erzeugt, welches in der HashMap der Factory gespeichert wird. Die Beschreibung des Clients wird anschließend an den Beobachter der Factory den Scheduler, gesendet, welcher die Beschreibung zu seinen verfügbaren Clients hinzufügt.

- Ist mindestens ein Projekt am Broker angemeldet, wird der *ClientFactory*, über die Funktion *setCompute* mitgeteilt, welchem SoC-Clienten eine Konfiguration zu übermittelt ist. Diese wird der *SoCClient* Repräsentation übergeben und an das SoC übermittelt.

- Nach der Konfiguration des SoC-Clienten werden die jeweiligen Datensätze übertragen, welche vom Scheduler über die *ClientFactory* an das *SoCClient* Objekt übergeben werden. Nach der Verarbeitung der Datensätze sendet der SoC-Client die Antwort an den Broker und die Datei wird vom *SoCClient* Objekt gespeichert.
- Am Ende des Schedulings fragt der Broker die Anzahl der gespeicherten Antworten ab. Diese werden dann von den *SoCCLient* Objekten an den Scheduler übergeben.

Kommunikation mit dem SoC-Client

Das *IClient* Objekt des SoC-Clienten enthält die Informationen über den Clienten sowie die Implementierung der Kommunikation. Die Kommunikation findet hier über die Klasse *SoCCon* statt, welche das Socket zur Datenübertragung verwaltet. Hier werden die Funktionen *sendData* und *sendPRM* zum Senden von Byte-Arrays bereitgestellt und die Daten, die vom SoC gesendet werden, empfangen.

Die Kommunikation mit dem Client erfolgt mit dem in Kapitel 4.1 vorgestelltem Protokoll und wird über das State-Pattern implementiert. Dieses Pattern wird eingesetzt, um zustandsabhängige Verhaltensweisen von Objekten zu erzeugen ohne hierbei auf Switch Anweisungen zurückzugreifen ([Freeman u. a. (2005)],[Gamma u. a. (2005)]). Hierzu wird eine abstrakte Klasse erstellt, die die Aktionen des Automaten enthält:

- SendPRM: Diese Aktion wird aufgerufen, wenn eine PRM an den SoC-Client zu senden ist. Der dazugehörigen Datensatz wird im IDLE-Zustand übergeben und im Konstruktor an den Folgezustand weitergereicht.
- SendData: Beim Senden von Datensätzen wird diese Aktion aufgerufen. Die Übergabe der Daten erfolgt ebenfalls im IDLE-Zustand. Der Datensatz wird anschließend beim Erzeugen des folgenden Objektes gelesen und serialisiert.
- Receive: Werden Daten vom SoC-Clienten empfangen, so werden diese an die receive Funktion übergeben.

Daraufhin sind die Zustände des Automaten in jeweils einer Klasse zu erstellen (vgl. Abbildung 54).

Die Anzahl der implementierten Zustände unterscheidet sich hierbei von denen, die in Kapitel 4.1 vorgestellt wurden (vgl. Abbildung 55). Aufgrund der Passivität der Zustände, werden bereits beim Aufrufen der Funktionen im IDLE-Zustand die Anfragen an den SoC-Clienten verschickt. Die Funktionen *SendPRM* und *SendData* werden vom Scheduler aufgerufen, wenn entsprechende Daten zu versenden sind. Die Funktion *receive* wird von der *SoCCon* Klasse aufgerufen, wenn Daten vom SoC-Client empfangen wurden.

6.2 Project-Factory zur Verwaltung von Projekten

Die Project-Factory dient dem Aufbau der Kommunikation mit einem Projekt sowie dem Verwalten der angemeldeten Projekte und Datensätze. Der Aufbau der Project-Factory basiert wie die Client-Factory auf dem Factory-Method Idiom. Sie trennt weiter die Implementierung der Projekte vom Scheduler und stellt die benötigten Beschreibungen zur Verfügung.

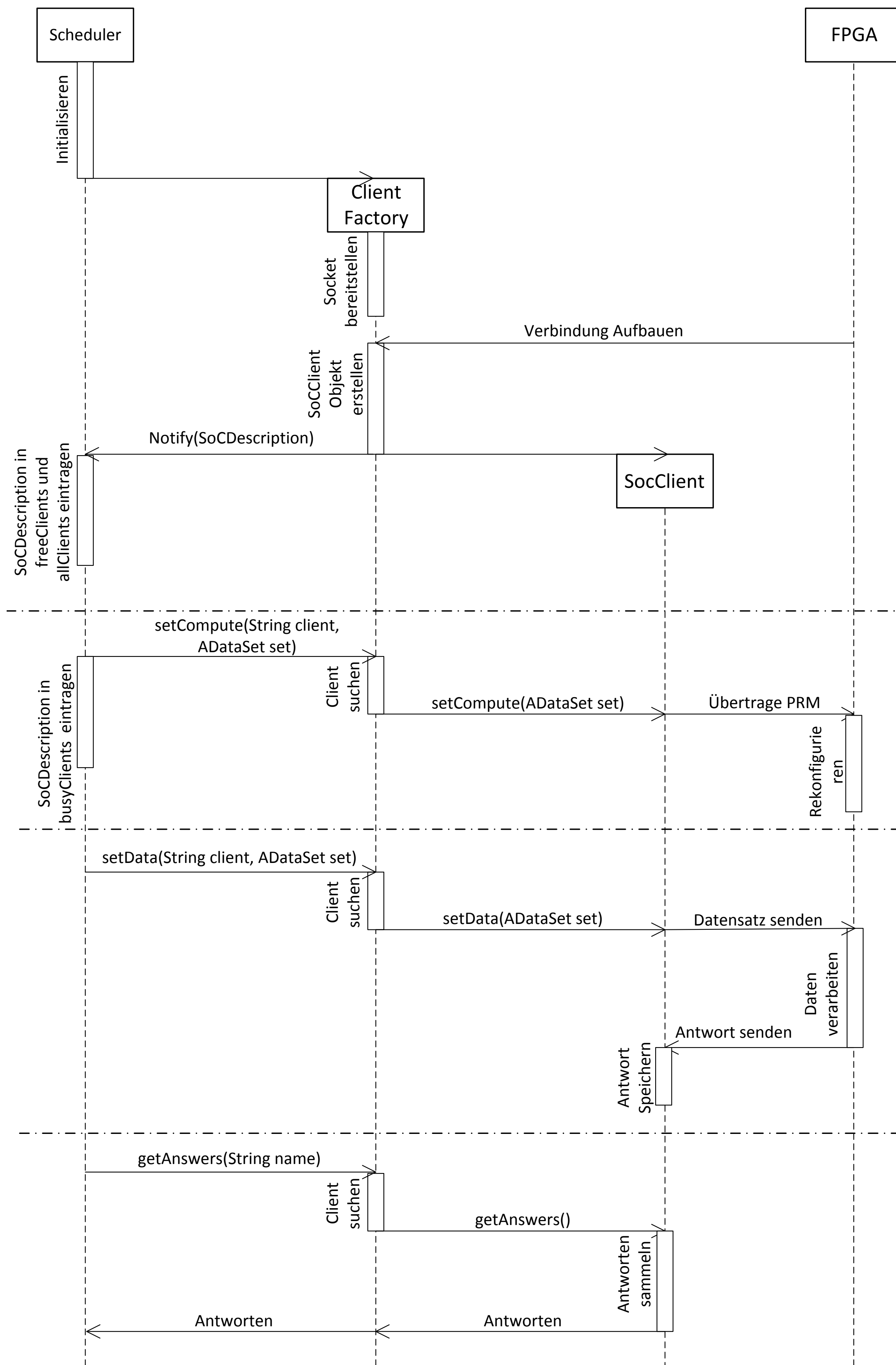


Abb. 53: *Ablaufdiagramm der Client-Factory und der Interaktion mit dem Broker und den angemeldeten SoC-Clients*

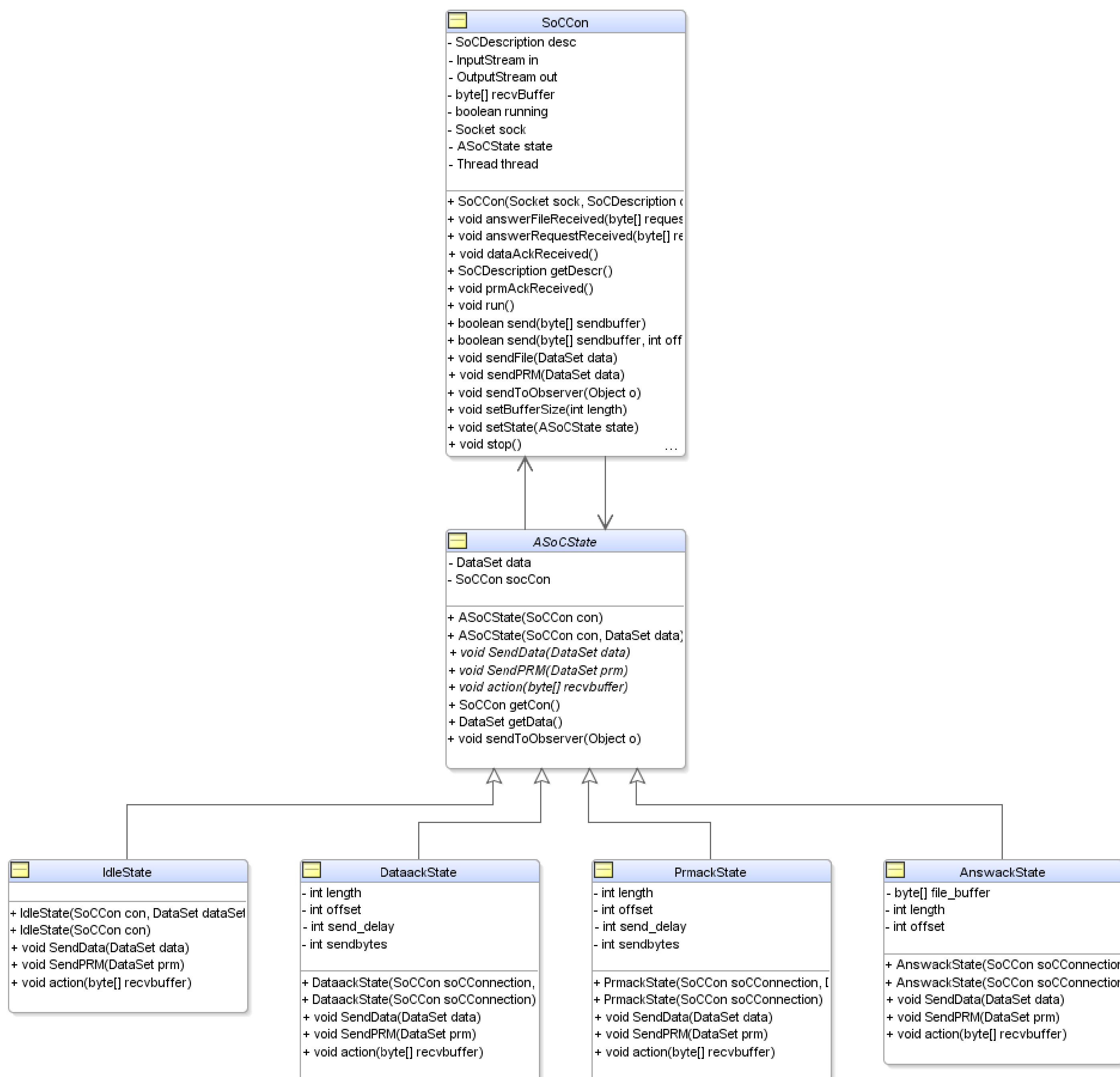


Abb. 54: *State Pattern Implementierung zur Kommunikation des Brokers mit den SoC-Clients*

Das Speichern der Projekt Objekte erfolgt ebenfalls in einer HashMap, wobei der Name des Projektes als Schlüssel verwendet wird. Die Kommunikation zwischen den Projekten und dem Broker erfolgt allein auf Basis der in Kapitel 4.2 definierten XML Struktur, wodurch beim Aufbau der Verbindung kein Name des Projektes zur Verfügung steht. Der Projekt Name wird gesetzt, sobald die erste Konfiguration übertragen wird. Danach wird das Projekt der HashMap hinzugefügt und dem Scheduler die Beschreibung zur Verfügung gestellt.

Die Kommunikation erfolgt ebenfalls auf Basis der Socket Bibliothek, wobei ein Server-Socket auf Kommunikationsanfragen von Projekten wartet (vgl. Abbildung 56). Hierzu wird in der Project-Factory das *Runnable* Interface implementiert und ein Thread erzeugt, der die blockie-

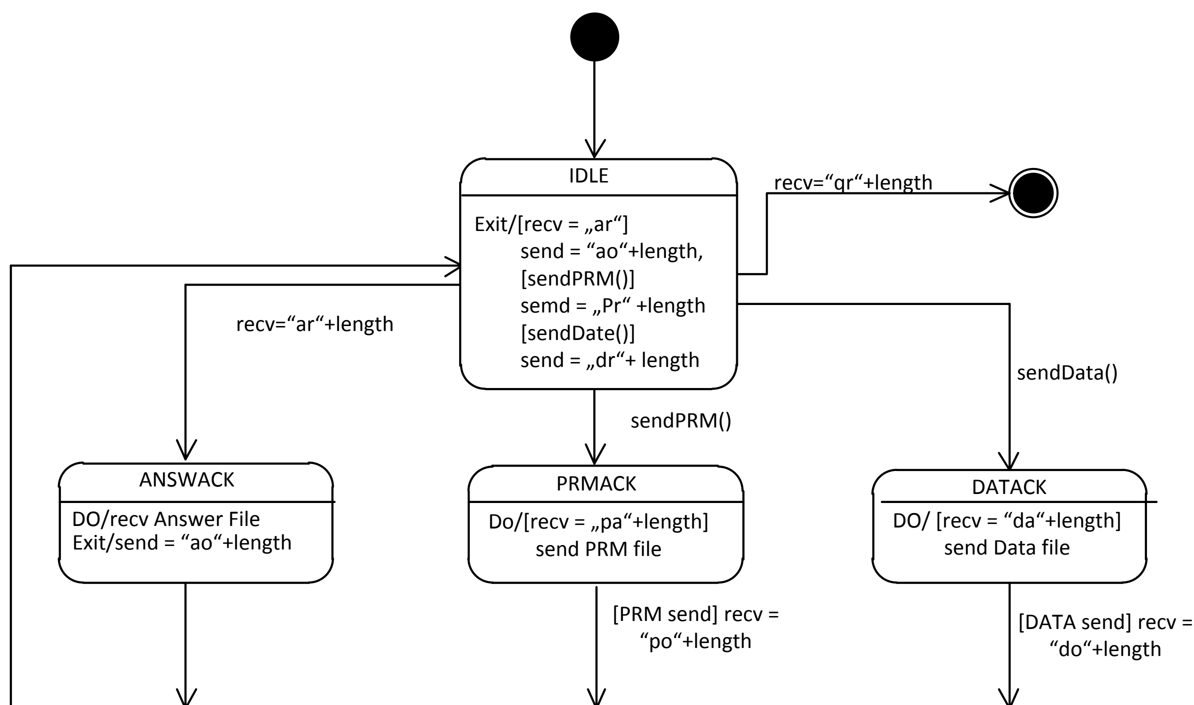


Abb. 55: *Modifizierter Automat zur Kommunikation mit den SoC-Clienten (vgl. Abbildung 54)*

rende Funktion *accept* ausführt und auf die Anfrage eines Projektes einen neuen Socket erstellt, welcher dem Projekt im Konstruktor übergeben wird.

Repräsentation der Projekte im Broker

Die Daten des Projektes werden in der Implementierung als Dateien im Verzeichnis des Brokers gespeichert. Die zugehörigen *ADataSet* Objekte werden in den ArrayLists des Projektes gehalten. Es wird nach Konfiguration, Berechnungsdatensatz und den Antworten, welche in je einer Liste gespeichert sind, unterschieden.

Das Empfangen der Daten erfolgt wie in den Factorys als eigenständiger Thread, um den blockierenden Leseaufruf zu ermöglichen. Hierzu wird die *readObject()* Funktion des *InputStreams* verwendet, welche den kompletten String mit einem Funktionsaufruf liest. Das Konvertieren des Strings in ein Objekt erfolgt mit der XStream Library, da diese die Objekte mit geringem Speicherverbrauch und Prozessorauslastung erzeugt [XStream (2011)]. Das Objekte wird wie in Listing 11 dargestellt erstellt. So wird der String mit dem XML empfangen und daraus ein Objekt erzeugt. Ist dieses Objekt vom Typ, der zur übertragenden Klasse, wird eine entsprechende Type-Cast vorgenommen, die Daten aus dem Objekt extrahiert und als Datei gespeichert.

Sind Antworten vom Scheduler an das Projekt übergeben worden, so werden diese vom *SendThread* an den Project-Client versendet. Dieser *SendThread* ist als innere Klasse aufgebaut und überprüft periodisch den Inhalt der Answer ArrayList. Ist hier eine Antwort enthalten wird aus dem *ADataSet* ein *DataFile* Objekt erzeugt und die Datei gelesen, so dass das Byte-Array im *DataFile* gefüllt wird. Das Erstellen des XML-Strings und die Übertragung der Daten erfolgt daraufhin. Anschließend wird die Datei vom Server gelöscht.

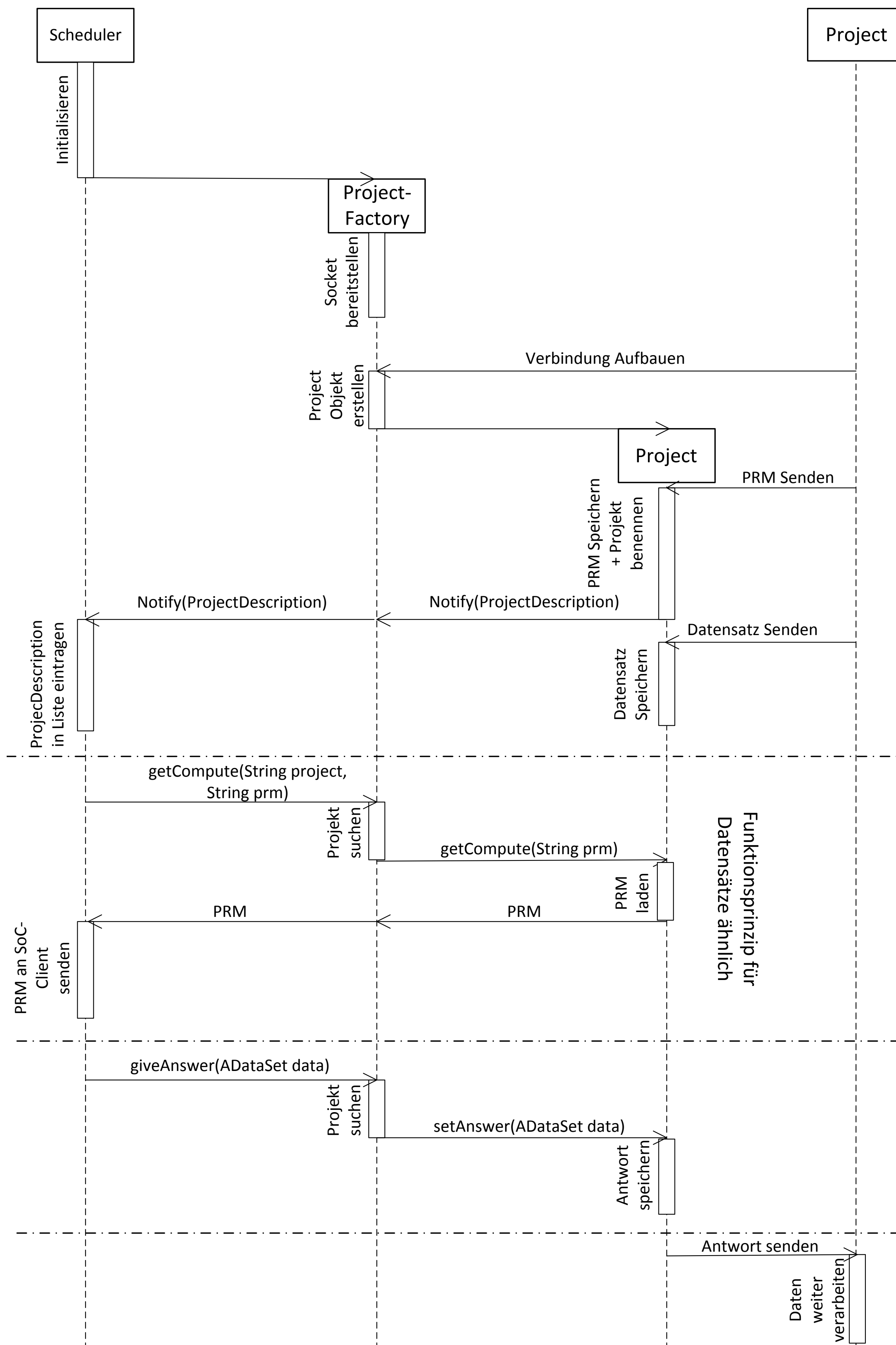


Abb. 56: *Ablaufdiagramm der Project-Factory und der Interaktion mit dem Broker und angemeldeten Projekten*

Listing 11: *Empfangen des XML Strings im Projekt Objekt*

```
String str;
XStream stream = new XStream(new StaxDriver());
DataFile df;

while (running) {
    try {
        str = (String)in.readObject();
        Object o = stream.fromXML(str);

        if (o instanceof DataFile) {
            df = (DataFile)o;
            ...
```

6.3 Scheduling der SoC-Clients und der Projekte

Die Scheduler Komponente ist für die Verteilung der SoC Ressourcen auf die einzelnen Konfigurationen der Projekte zuständig. Dieser arbeitet ähnlich eines Schedulers in einem Betriebssystem, wobei hier ein FiFo basiertes Scheduling implementiert ist. Mit dieser Art der Implementierung werden die zuerst angemeldeten Projekte bevorzugt abgearbeitet und erst nach einem Abschluss der Verarbeitung der Datensätze, wird das nächste Projekt auf die SoCs verteilt. Ein zeitbasiertes Scheduling kommt für den Broker nicht in Frage, da hier die Verarbeitung auf den SoC-Clienten zu unterbrechen wäre, wobei Datensätze verloren gingen und die langwierige Übertragung der PRMs wiederholt stattfinden würde (vgl. Kapitel 7.3). Ebenso wird auf ein Prioritäts-basiertes Scheduling verzichtet, da die Projekte ohne Bevorzugung zu verarbeiten sind.

Das Scheduling der Daten erfolgt alle zehn Sekunden, wobei jeweils vier Schritte durchlaufen werden.

1. cleanUp: Während des Aufräumens wird der Status der zur Verfügung stehenden Konfigurationen überprüft. Stehen für aktive Konfigurationen keine Datensätze mehr zu Verfügung, werden die zugehörigen SoC-Clients als frei kategorisiert und stehen für ein neues Projekt bereit. Weiter wird der Status der Projekte und SoC-Clienten abgefragt. Sind diese auf *TO_DELETE* gesetzt, so werden die jeweiligen Stop Funktionen aufgerufen und die Beschreibungen aus den Listen gelöscht.

2. schedul: Beim Scheduling werden die verfügbaren SoC-Clienten auf die Projekte verteilt. Hierzu wird die maximale Anzahl der Clienten m pro Projekt berechnet, wobei die Anzahl der freien SoCs f mit den bereits belegten SoCs b durch die Anzahl der Projekte p dividiert wird, bei Ergebnissen mit Nachkommastellen werden diese Abgeschnitten und die dadurch nicht verteilten SoC-Clienten im nächsten Durchgang verteilt. Darauf folgend werden die angemeldeten Projekte durchgegangen und die Anzahl der, zum Projekt zugeteilten SoCs, mit der maximalen Anzahl verglichen. Ist die maximale Anzahl kleiner als der bereits zugewiesenen, so werden weitere SoCs dem Projekt zugeteilt bis die maximale Anzahl erreicht ist. Durch diese Methode werden jedem Projekt gleich viele SoCs zugeteilt. Sind einem Projekt am Anfang viele SoCs zugeteilt worden, so werden diese nach der Bearbeitung auf die später angemeldeten Projekte verteilt.

3. schedulDataSet: In dieser Phase werden die einzelnen Clienten durchlaufen und ihnen wird ein Datensatz der Konfiguration zugewiesen. Ist hier keine Konfiguration mehr vorhanden, wird im nächsten Durchlauf der Client als verfügbar markiert.

4. schedulAnswers: Nach der Berechnung werden die Antworten in den *IClient* Objekten verwaltet. Die Anzahl der bereitstehenden Antworten wird vom Scheduler abgefragt und bei einer bereitstehenden Antwort an das zugehörige Projekt weitergeleitet.

Die jeweiligen Beschreibungen der SoC-Clienten und der Projekte werden in insgesamt drei ArrayLists gehalten, welche folgenden dargestellt sind:

- busy- oder freeClients: Die freien bzw. die beschäftigen Clienten werden in je einer Liste gehalten. So erfolgt eine schnelle Einteilung der freien Clienten ohne eine komplette Liste zu durchsuchen, wobei die Komplexität der Suche von $O(n)$ auf $O(1)$ gesenkt wird[4].
- projects: Diese Liste beinhaltet die einzelnen Projekte, welche wiederrum mehrere PRMs und Datensätze enthalten können.

Beim Starten des Servers werden die zugehörigen Factorys gestartet und der Scheduler als Observer in den jeweiligen Beobachter-Listen eingetragen. Erfolgt die Anmeldung eines SoC-Clients oder eines Projektes, wird die jeweilige Beschreibung dem Scheduler mitgeteilt und in die entsprechende Liste eingetragen. Dieser teilt die Ressourcen beim nächsten durchlauf des Scheduling entsprechend zu.

[4] n ist die Anzahl der angemeldeten Clienten

7 Beispiel Projekt zur Verifizierung von Bildverarbeitungsalgorithmen eines autonomen Fahrzeuges

Aktuell finden an der HAW Projekte statt, die ein SoC basiertes autonomes Fahrzeug entwickeln. Zu diesem Zweck wurde in [Sahak (2011)] eine Bildverarbeitungspipeline entworfen, die zur Fahrspurerkennung genutzt wird.

Während der Fahrt des Fahrzeuges werden die von einer Kamera aufgenommen Bilder bearbeitet und einer Kantendetektion unterzogen. Dies erfolgt in Echtzeit, was in diesem Fall eine maximale Latenz von einem Bild bei 60 Bildern pro Sekunde bedeutet. Die Pipeline beinhaltet dabei zwei PRR, welche je nach Anforderung vom System konfiguriert werden (vgl. Abbildung 57). Das System is in der Lage aus den zur Verfügung stehen Filtern jede Kombination zu erzeugen und gegebenenfalls während des Betriebes die Konfiguration zu ändern. Hierbei stehen die folgenden Filter zur Verfügung:

- Median: Das Median-Filter ist ein Rangordnungsoperator, welcher zur Berechnung des Zielpixels die Werte des Ausgangspixels und der ihn umgebenden Pixel vergleicht. Der Wert, der in der Mitte der geordneten Werte liegt, wird als Zielpixel verwendet.
- Binarisierung: Die Binarisierung ist ein Filter, welcher anhand einer vorgegebenen Grauwertschwelle des Ausgangspixels das Zielpixel auf weiß oder schwarz setzt.
- Erosion: Das Erosionsfilter ist ein morphologischer Nachbarschaftsoperator, welcher zur Verkleinerung von weißen Flächen in Bildern eingesetzt wird. Hier werden in einem Binärbild, die Umgebungspixel des Ausgangspixels UND-verknüpft und das Ergebnis als Zielpixel verwendet.
- Sobel: Das Sobel-Filter ist ein Nachbarschaftsoperator, welche Objekt-Kanten in einem Bild hervorhebt. Dies erfolgt unter der Anwendung zweier Matrizen, die mit den Ausgangspixeln multipliziert werden. Anschließend werden die jeweiligen Ergebnisse addiert. Bei einem hohen Ergebnis dieser Addition liegt eine Kante vor und das Zielpixel wird dementsprechend hervorgehoben.

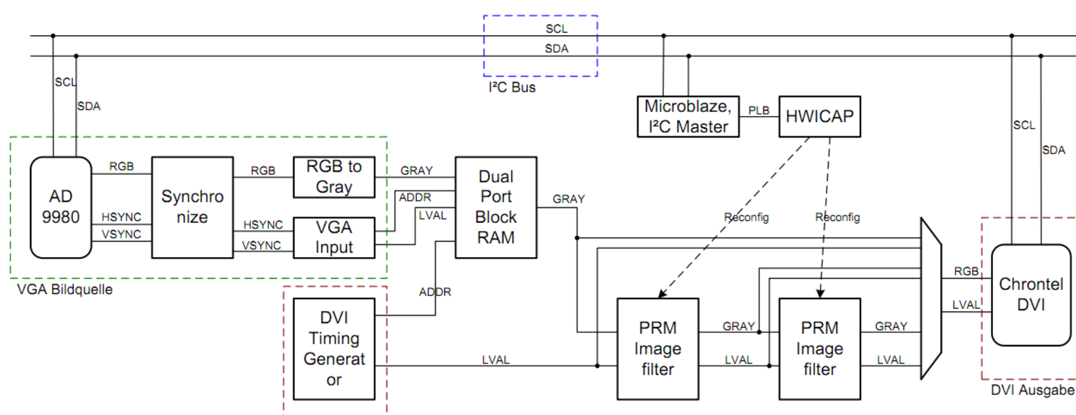


Abb. 57: *Bildstromverarbeitungs-Komponenten des Zielsystems im autonomen Fahrzeug [Sahak (2011)]*

Das Testen der verschiedenen Konfigurationen der einzelnen Filter sowie des Zusammenspiels erfolgt als Projekt im DSN, so dass kein autonomes Fahrzeug zum Testen benötigt wird. Die folgenden Kapitel erläutern die Anpassung für das DSN sowie das Erstellen des Projekt-Clients (vgl. Anhang G).

7.1 Einbindung der Projekt IPs

Die Module der Bildverarbeitungspipeline sind für den Einsatz in einer Echtzeit Umgebung entwickelt, welche den Datenstrom über die VGA-Kontrollsignale steuert. Diese Signale werden anhand von Zählern generiert, welche die horizontalen und vertikalen Koordinaten erzeugen. Auf Basis der Koordinaten werden anschließend die entsprechenden Steuersignale, wie Frame (*FVAL*)- und Line (*LVAL*)-Valid, generiert (vgl. Abbildung 58).

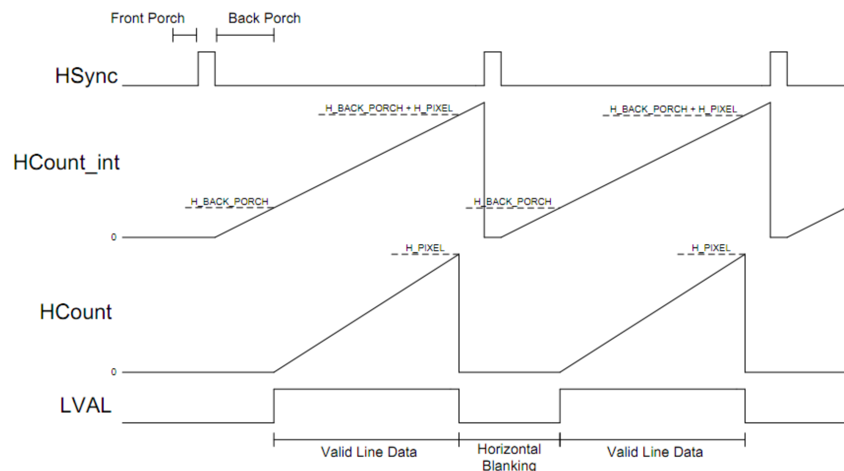


Abb. 58: *Kontrollsignale des Bilddatenstromes im autonomen Fahrzeug auf Basis von Zählern [Sahak (2011)]*

Die eigentlichen Bilddaten werden ausgegeben, wenn keiner der Porch-Bereiche aktiv ist, also die Signale *LVAL* und *FVAL* aktiv sind. Das *LVAL* Signal wird in den Filter Modulen zur Steuerung des IPs verwendet und zeigt an, dass die anliegenden Pixel-Daten zu bearbeiten sind (vgl. Abbildung 59). Die für die Filter verwendeten Signalen unterscheiden sich von den für das Interface vorgegebenen und sind über einen Wrapper zu erzeugen (vgl. Kapitel 4.1).

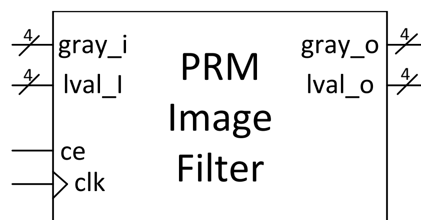


Abb. 59: *Entity der Filter IPs*

Die zu verarbeitenden Bildinformationen liegen vier Bit breit hintereinander im Speicher und werden als 32 Bit Wort der PRR bereitgestellt. Das Erzeugen der Eingangsbreite der Filter wird durch das Multiplexen der zur Verfügung stehenden Eingangssignale erzeugt und von der *INPUT_FSM* gesteuert. Diese ist ebenso für das Erzeugen der *LVAL* zuständig, welche beim Bereitstehen von Bilddaten gesetzt wird. Die Bildgröße ist in den Filter Modulen auf 800×600 Pixel definiert, so dass keine Signale zum Anzeigen einer neuen Zeile zu erzeugen sind (vgl. Abbildung 60).

Die Ausgangsdaten des Filters werden vier Bit-weise in Synchronisations-FlipFlops gespeichert. Dies erfolgt bis insgesamt 32 Bit an Daten zur Verfügung stehen, so dass 8 FlipFlops verwendet werden. Die in den FlipFlops enthaltenen Daten werden konkateniert, so dass ein 32 Bit Wort entsteht, welches am Ausgang des Interfaces zur Verfügung steht. Die Signale zum

Schreiben der Daten, welche vom *DYN_Interface* definiert sind, werden vom *OUTPUT_FSM* erzeugt und die Daten in das FiFo geschrieben (vgl. Abbildung 60 und 61).

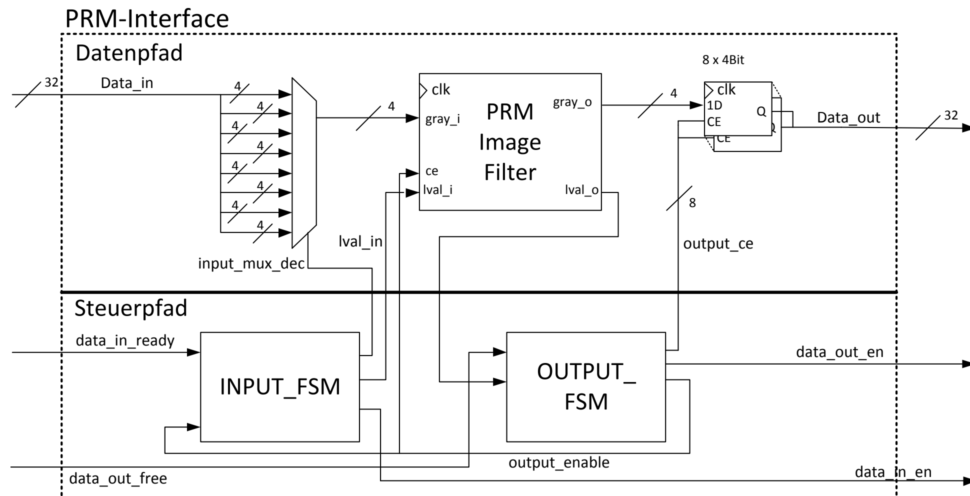


Abb. 60: *Wrapper der Filter-Entity für das DYN_Interface mit Demultiplexing, dem Steuerautomaten sowie dem Konkatenieren der Ergebnisse zur Anpassung an den 32 Bit Ausgangsvektor*

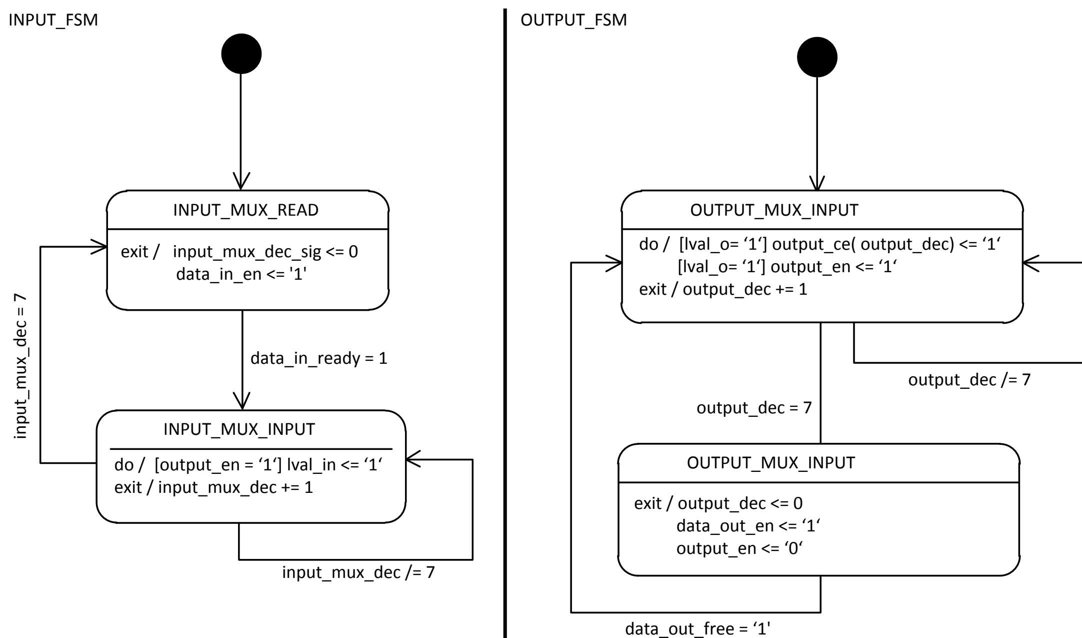


Abb. 61: *Zustandsdiagramm zum Erzeugen der Steuersignale für die Filter-Module sowie dem Lesen und Schreiben der Bilddaten*

7.2 Projekt Software für die Server-Kommunikation und Daten-Vorverarbeitung

Das Testen der Filter erfolgt auf Basis von Bildern, die eine Szene aus Sicht des autonomen Fahrzeuges darstellen. Diese werden mit einer Kamera aufgenommen und im 800×600 Format speichert, welches die definierte Auflösung der Filter ist. Die aufgenommen Bilder werden als JPEG oder PNG mit einer Farbauflösung von 32 Bit gespeichert und mit einem Matlab Script

verändert, so dass ein vier Bit Grauwertbild entsteht, welches das Format der Daten Vorverarbeitung ist. Durch das Einlesen des Bildes wird eine Matrix der Größe $800 \times 600 \times 3$ erzeugt, die das Originalbild im RGB Format enthält. Das Erzeugen des Grauwertbildes entsteht anschließend durch das Addieren der R,G und B Werte und das Multiplizieren des Ergebnisses mit $\frac{1}{3}$. Die höherwertigen vier Bit des entstandenen Grauwerte sind anschließend, unter Berücksichtigung des in Kapitel 4.1 vorgestellten Formates, in eine Datei zu schreiben.

Auf Basis der in Kapitel 4.2 vorgestellten Architektur erfolg der Aufbau des *Project-Client*s (vgl. Abbildung 62). Die Kontrolle des *Project-Client*s wird vom Anwender über eine GUI vollzogen. Diese bietet die Möglichkeit eine Auswahl zu treffen, welche Filter zu testen sind. Ist ein Filter ausgewählt, wird die entsprechende Konfigurationsdatei vom *Manager* aus dem *Persistenz* Objekt angefordert und dieser erhält das *DataFile* Objekt, welches beim Versenden an den Broker in ein XML String gewandelt wird (vgl. Listing 12). Wird eine Antwort vom Broker empfangen, so wird das erhaltene XML-Objekt in ein *DataFile* Objekt gecastet und die Daten vom *Persistenz* Objekt gespeichert.

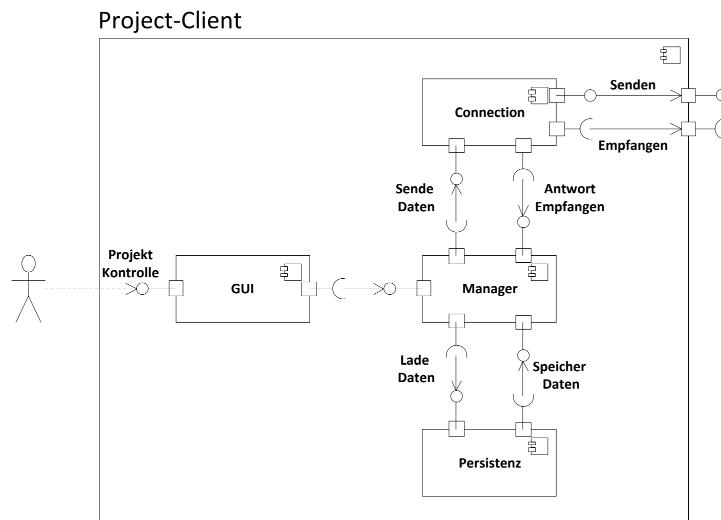


Abb. 62: *Komponenten des Beispiel-Projekts und der Interaktion mit dem Anwender*

Listing 12: *XML String mit dem Inhalt einer PRM*


```
<?xml version="1.0" ?>
<FileIO.DataFile>
        <changed>false</changed>
        <obs></obs>
        <projectName>line</projectName>   <!— Name des Projektes —>
        <computeName>bin</computeName>    <!— Name des Compute Elementes —>
        <answer>false</answer>            <!— Ist diese Datei eine Antwort —>
        <toDelete>false</toDelete>        <!— Soll die Datei mit diesem Namen
                                              gelöscht werden —>
        <fileName>binprm.bit</fileName>   <!— Dateiname —>
        <computeCount>-1</computeCount>   <!— Wie oft soll die Datei berechnet
                                              werden (Compute Elemente -1) —>
        <content>                         <!— Inhalt der Datei —>
                AAkP8A/wD/AP8AAAAWEAIWJpbl9yb3V0ZWQubmNkO1VzZXJJRD0weEZGRkZGRkZGAGIADzZ2bHgy
                NDB0ZmYxMTU2AGMACzIwMTEvMDcvMjYAZAAJMTI6NTM6MjMAZQA03zD/////////////////////
                /////////////////////wAAALsRIgBE//////////+qmVVmIAAAADAAgAEAAAAHIAAAACAAAAAw
                AYABBCUAkzAAgAEAAAAAMAAgAQAAAAAwAIABAAAAASAAAAAwACABAAAAgCAAAAAwAEAAUAGRhQAA
                ......
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
        </buf>
        <count>0</count>
        <pos>0</pos>
        <markpos>-1</markpos>
```

```
25          <marklimit>0</marklimit>
        </bis>
</FileIO.DataFile>
```

7.3 Messtechnische Analyse der Funktion und des Zeitverhaltens

Die Implementierung des Beispiel Projektes wird mit einem Bild getestet, welches ein reales Szenario der Fahrspurerkennung darstellt (vgl. Abbildung 64). Dieses wird unter der Verwendung des in Kapitel 7.2 vorgestellten Skriptes eingelesen und in das in Kapitel 4.2 definierte Format gewandelt. Dieses enthält die Angabe über die Größe des Datensatzes sowie die vier Bit Grauwerte des Bildes. Zur Berechnung werden die Binär-, Median-, Eorsion- und Sobel-Filter verwendet, welche mit dem gleichen Datensatz arbeiten. Das Netzwerk besteht in dem Versuchsaufbau aus zwei ML605 Boards, welche das statische System enthalten. Diese sind mit dem Broker verbunden und erhalten die Konfiguration, welche vom Projekt verteilt wird (vgl. Abbildung 63). Die Filter Konfigurationen werden nacheinander auf die SoCs verteilt, bis alle Filter getestet wurden. Die Abbildungen 65 - 68 zeigen die jeweiligen Ausgaben der einzelnen Filter.

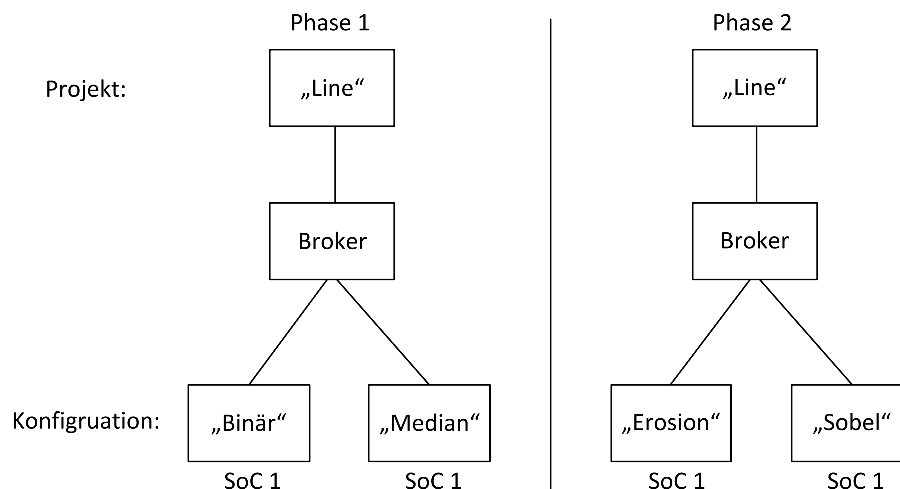


Abb. 63: *Beispiel der Konfiguration im DSN mit zwei SoCs und der Verarbeitung eines Projektes mit vier Konfigurationen*

Das Erstellen der PR-Konfigurationen erfolgt auf Basis des statischen Systems. Dieses wird in PlanAhead den anderen Konfigruationen zur Verfügung gestellt und die Module so an das statische System angepasst. Der in Tabelle 12 angegebene Ressourcenbedarf der einzelnen Module ist den generierten Netzlisten aus dem ISE Design Werkzeug entnommen. Zur Analyse des Zeitverhaltens war der in Abbildung 69 dargestellte Messaufbau zu realisieren. Die LED ist je nach Ereignis ein- und auszuschalten, wodurch die benötige Zeit durch die Zeitdifferenz der aufsteigenden bzw. abfallenden Flanke gemessen wird. Die Ergebnisse zeigen, dass die einzelnen Rekonfigruationen durch das Abschalten der Interrupts dieselbe Zeit benötigen, da die „.bit“ Dateien dieselbe Größe besitzen und das Schreib der Konfiguration nicht durch den Scheduler oder eine Übertragung vom *xps_ll_temach* unterbrochen wird. Die unterschiedliche Übertragungszeit der Module ist durch das Scheduling im Server zu erklären, welches dem Sende-Thread unterschiedlich viel Rechenzeit zuteilt. Die Berechnungsdauer der Datensätze wird ermittelt in dem gemessen wird, wieviel Zeit von der Übergabe der Daten an das *DYN_Interface* bis zum Schreiben der Daten in die letzte Speicher Adresse vergeht. Dies erfolgt im Betrieb, wobei das Scheduling im MicroBlaze in die Messung mit einfließt, so dass

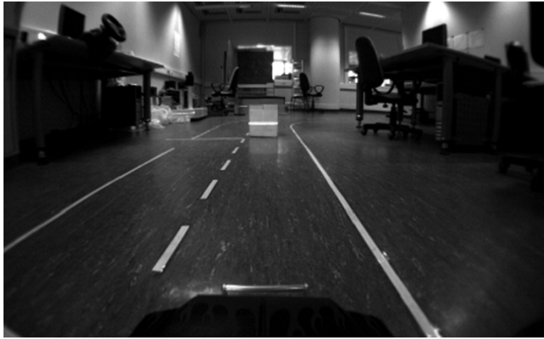


Abb. 64: *Eingangstestbild*

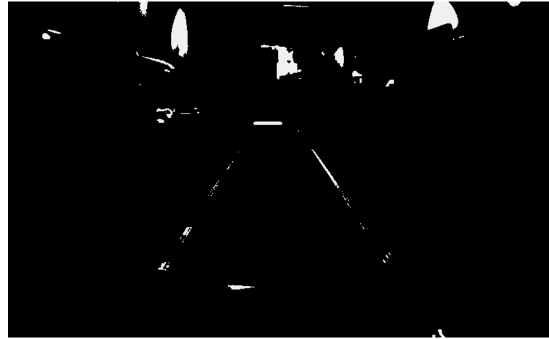


Abb. 65: *Ausgabe des Binären-Filters*

Abb. 66: *Ausgabe des Erosions-Filters*

Abb. 67: *Ausgabe des Median-Filters*

Abb. 68: *Ausgabe des Sobel-Filters*

die Zeit im realen Betrieb des SoC-Client gemessen wird. Bei der Übertragung der Bit-Files zwischen dem Broker und dem SoC-Client kam es immer wieder zu einem Abbruch der Übertragung. Dieser Fehler wurde durch das Einfügen einer Wartezeit zwischen jeweils 100 Byte, die übertragen wurden, behoben. So steht dem SoC-Client eine Millisekunde zur Verarbeitung der angekommenen Daten zur Verfügung.

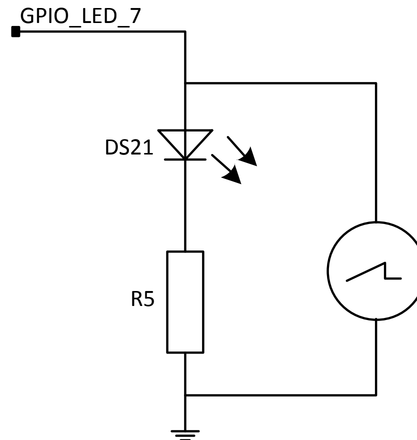


Abb. 69: *Messchaltung zur Analyse des Zeitverhaltens im FPGA [Xilinx (2009a)]*

Tabelle 12: *Ressourcenbedarf der einzelnen Module sowie die Ergebnisse der Zeitmessung zur Übertragung, Rekonfiguration und dem Verarbeiten der Daten der jeweiligen Filter PRMs*

Modul	Ressourcenbedarf		Zeitmessung		
	Slices	RAM	Übertragung der PRM	Rekonfiguration der PRR	Berechnung
Binär	3	0	77 s	31,25 s	25,52 ms
Erosion	237	0	73 s	31,25 s	85,93 ms
Median	531	0	73 s	31,25 s	77,09 ms
Sobel	479	0	75 s	31,25 s	86,45 ms

Die Ergebnisse der Analyse zeigen eine erfolgreiche Implementierung des Systems sowie die Verwendung von FPGAs als verarbeitende Komponenten in einem Distributed Computing Netzwerk.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die Berechnungen in bisherigen Distributed Computer Plattformen werden auf handelsüblichen PCs durchgeführt. Eine hohe Effizienz dieser Systeme ist durch die serielle Abarbeitung sowie die Unterbrechung durch anderes Threads oder Interrupts nicht gegeben. Hier empfiehlt es sich, zuzüglich zu den bisherigen Systemen, FPGAs als Beschleuniger-Plattform zur verwenden. Mit der partiellen Rekonfigurations-Technologie wird ein statisches SoC-System erstellt, welches Teile des FPGAs während des Betriebes rekonfiguriert. Durch die Anbindung solcher SoC-Systeme an einen Server, welcher verschiedene Projekte verwaltet, wird ein FPGA basiertes Distributed Computing System geschaffen.

Das Gesamtsystem besteht aus einer Client-Broker-Server Architektur, wobei die SoCs die Clients und die Projekte die Server repräsentieren. Die SoCs sowie die Projekt-Clients stellen zu Beginn eine Verbindung zu dem Broker her, welcher die einzelnen Teilnehmer verwaltet. Diese Architektur wird vorwiegend durch die Eigenschaften des verwendeten TCP/IP-Stacks auf dem SoC bestimmt, welcher nur eine begrenzte Anzahl an Verbindungen verwalten kann. Die Übertragung der Daten erfolgt mit einem Overlay Protokoll. Dieses wird zur Unterscheidung der stattfindenden Datenübertragung verwendet und ist ein vier Wege Protokoll. Die jeweiligen Anfragen enthalten dabei die Art der Anfragen, sowie den Zustand und die Länge der zu übertragenen Dateien, welche zur Sicherung der Übertragung dient, da beide Seiten wissen, wieviele Daten zu übertragen sind. Diese Anfragen werden vom Initiator der Übertragung generiert, worauf die Datenübertragung nach einer Antwort des Empfängers gestartet wird. Die Kommunikation zwischen dem Broker und den Projekten erfolgt unter der Verwendung von XML. Es wurde ein Objekt definiert, welches die Eigenschaften sowie die Daten enthält. Das aus den Daten erstellte XML Objekt wird versendet und auf der Empfänger Seite wieder entpackt und gespeichert bis die Daten verwendet werden.

Die Implementierung des SoC-Clients erfolgt aus Basis des MicroBlaze Mikroprozessors, welcher zur Steuerung des Clients verwendet wird. Hierzu wird dieser so parametrisiert, dass ein hoher Datendurchsatz erreicht wird, was durch das Verwenden einer fünf-stufigen Pipeline sowie dem Verwenden von Caches erfolgt. Die Anbindung eines externen Speichers erfolgt über den MPMC, welcher ebenfalls an das Ethernet-Interface angeschlossen ist, um die ankommenden Daten zu speichern. Das Interface zum dynamischen Bereich ist als PLB-Master entwickelt, so wird der MicroBlaze entlastet, da dieses Interface die Daten direkt aus dem Speicher liest. Die gelesenen Daten werden der PRR über ein FiFo bereitgestellt, um die unterschiedlichen Laufzeiten in den verschiedenen PRMs auszugleichen. Die Kommunikation zum Broker wird mit der LwIP Bibliothek realisiert, welche das TEMAC Interface des FPGAs unterstützt. Die vom LwIP empfangen Konfigurationen werden über ein Dateisystem im externen Speicher gehalten und bei einer vollständigen Übertragung in die PRR geschrieben. Die zu berechnenden Datensätze werden über einen Ringbuffer gespeichert und die Adressen der Daten dem Interface zur PRR zur Verfügung gestellt, welches die Daten direkt aus dem Speicher liest.

Der Broker basiert auf dem Factory-Idiom zum Erstellen der SoC und Projekt Repräsentation. Diese werden beim Aufbauen der Verbindung erstellt und enthalten Beschreibungen, wie den Namen und die Konfiguration. Stehen SoCs und Projekte bereit, werden die freien SoCs auf die zur Verfügung stehen Projekte verteilt. Dies erfolgt nach dem FiFo Prinzip, wobei die Clienten gleichmäßig auf die Projekte verteilt werden.

Ein Test des Systems erfolgte mit Filter-Algorithmen eines Fahrspurerkennungssystems aus einem autonomen Fahrzeug. Diese werden über einen Wrapper in das Interface eingebunden, um die Steuersignale zu erzeugen. Getestet wurde das System anschließend erfolgreich mit Bildern, die eine reale Situtation des autonomen Fahrzeuges darstellen.

8.2 Ausblick

Die aktuelle Implementierung des DSN stellt ein Prototyp des Systems dar. So erfolgt beispielsweise die Kommunikation zwischen den einzelnen Komponenten unverschlüsselt und die Verwaltung speichert die Daten als Datei im Ordner des Servers. Die folgenden Erweiterungen sind am Prototypen vorzunehmen, um einen öffentlichen Einsatz zu ermöglichen:

- Sichern des Datenverkehrs: Die Kommunikation erfolgt zu diesem Zeitpunkt unverschlüsselt, wodurch keine vertrauenswürdige Verwendung des Systems, beispielsweise über das Internet, gegeben ist. Dieses kann durch das Verschlüsseln des Datenverkehrs mit einem asynchronen Verfahren verbessert werden. Hierzu sollte ein Schlüsselpart mit einem öffentlichen und einem privaten Schlüssel erzeugt werden, wobei der öffentliche Schlüssel den SoC-Clienten bekannt ist, welche den Datenverkehr mit diesem verschlüsseln. Der private Schlüssel ist allein dem Broker bekannt. So findet zum einen eine Authentifizierung der SoC-Clienten statt, zum anderen wird das System unempfindlich gegen Man-in-the-Middle Angriffe. Die Kommunikation zwischen dem Broker und den Projekten sollte ebenfalls mit einer asynchronen Verschlüsselung erfolgen, hier wird bei der Anmeldung der öffentliche Schlüssel des Projektes dem Broker übergeben, welcher den Datenverkehr mit diesem ent- und verschlüsselt.
- Anbindung einer Datenbank: Das Speichern der Konfigurationen und Datensätzen erfolgt aktuell als Datei in dem ausführendem Ort des Brokers. Dies ist durch die Anbindung einer effizienten Datenbank zu ersetzen.
- Kommunikation zwischen SoC-Client und Projekt: Die Kommunikation zwischen dem SoC-Client und dem Projekt erfolgt über den Broker. Dies hat bei einer hohen Anzahl an Projekten und Clienten eine starke Auslastung des Brokers zur Folge. Hier ist das Protokoll so anzupassen, dass der Broker dem SoC-Client die IP-Adresse und Port des Projekt-Clients übermittelt und die Übertragung der Datensätze direkt erfolgt. So wäre weiter jeder SoC-Client für alle Projekte verfügbar, aber die Auslastung des Brokers wird gering gehalten.

Weiter ist eine Anbindung des ML605 als Beschleuniger im PC geplant. So würde die PCI-Schnittstellte des Boards verwendet, um das System direkt in den PC zu integrieren. Ein selbst entwickelter Treiber würde anwendungsspezifische Hardwarebeschleuniger auf dem Board rekonfigurieren. So könnten beispielsweise Verschlüsselungsalgorithmen auf den FPGA ausgelagert werden, was zu einer Entlastung des Host-Systems führt und größere Datenmenge ohne große Prozessoren Auslastung ver- und entschlüsselt.

Literatur

[Abts 2010] ABTS, D.: *Masterkurs Client/Server-Programmierung mit Java: Anwendungen entwickeln mit Standard-Technologien: JDBC, UDP, TCP, HTTP, XML-RPC, RMI, JMS und JAX-WS*. Vieweg+Teubner Verlag, 2010. – ISBN 9783834813244

[Bengel 2004] BENGEL, Günther: *Grundkurs Verteilte Systeme*. Bd. 3. Auflage. Vieweg +Teubner, 2004. – ISBN 3-528-25738-5

[Buttazzo 2005] BUTTAZZO, G.C.: *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2005 (Real-Time Systems Series). – ISBN 9780387231372

[California 2010] CALIFORNIA, University o.: *Open-Source Software für Volunteer Computing und Grid Computing*. 2010. – Verfügbar Online unter `http://boinc.berkeley.edu/`; 15.12.10

[Diffie und Hellman 1977] DIFFIE, W. ; HELLMAN, M.E.: Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. In: *Computer* 10 (1977), june, Nr. 6, S. 74 –84. – ISSN 0018-9162

[Dunkels 2004] DUNKELS, Adam: *The lwIP TCP/IP Stack*. 2004. – Verfügbar Online unter `http://www.sics.se/~adam/lwip/`; 08.03.11

[Fons und Fons 2011] FONS, F. ; FONS, M.: Exploiting run-time reconfigurable hardware in the development of automatic fingerprint-based personal recognition applications. In: *Recent Application in Biometrics*. InTech, 2011

[Foundation 1998] FOUNDATION, Electronic F. ; LOUKIDES, Mike (Hrsg.) ; GILMORE, John (Hrsg.): *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 1998. – ISBN 1565925203

[Freeman u. a. 2005] FREEMAN, Eric ; FREEMAN, Elisabeth ; SIERRA, Kathy: *Entwurfsmuster von Kopf bis Fuß*. 1. O'Reilly, 2005. – ISBN 3897214210

[Gajski 1997] GAJSKI, D.D.: *Principles of digital design*. Prentice Hall, 1997. – ISBN 9780133011449

[Gamma u. a. 2005] GAMMA, E. ; JOHNSON, R. ; HELM, R. ; VLISSIDES, J. ; LARMAN, C.: *DESIGN PATTERNS : ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE*. Addison-Wesley, 2005 (Addison - Wesley Professional Computing Series). – ISBN 1405837306

[Greensted 2010] GREENSTED, Dr. A.: *Otsu Thresholding*. 2010. – Verfügbar Online unter `http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html`; 22.11.11

[Guneysu u. a. 2008] GUNEYSU, T. ; KASPER, T. ; NOVOTNY, M. ; PAAR, C. ; RUPP, A.: Cryptanalysis with COPACOBANA. In: *Computers, IEEE Transactions on* 57 (2008), nov., Nr. 11, S. 1498 –1513. – ISSN 0018-9340

[IBM 2007] IBM: *128-Bit Processor Local Bus Architecture Specifications Version 4.7*, 2007. – Verfügbar Online unter `https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/3BBB27E5BCC165BA87256A2B0064FFB4`; 11.11.11

[IEEE 2009] IEEE: Information technology - Portable Operating System Interface (POSIX) Operating System Interface (POSIX). In: *ISO/IEC/IEEE 9945 (First edition 2009-09-15)* (2009), 15, S. c1 –3830

[Kearney und Jasiunas 2006] KEARNEY, David ; JASIUNAS, Mark: Using Simulated Partial Dynamic Run-Time Reconfiguration to Share Embedded FPGA Compute and Power Resources across a Swarm of Unpiloted Airborne Vehicles. In: *'EURASIP Journal on Embedded Systems* (2006)

[Kumar u. a. 2006] KUMAR, Sandeep ; PELZL, Jan ; PFEIFFER, Gerd ; SCHIMMLER, Manfred ; PAAR, Christof: *Breaking Ciphers with COPACOBANA A Cost-Optimized Parallel Code Breaker or How to Break DES for 8,980 Eur.* 2006. – Verfügbar Online unter `http://www.copacobana.org/paper/CHES2006_copacobana_slides.pdf`; 15.05.11

[Max Planck 2010] MAX PLANCK, Geselschaft: *Einstein@Home*. 2010. – Verfügbar Online unter `http://einsteinathome.org/`; 22.12.10

[Microsoft 2000] MICROSOFT: *Microsoft Extensible Firmware Initiative FAT32 File System Specification*, 2000. – Verfügbar Online unter `http://msdn.microsoft.com/de-de/windows/hardware/gg463084`; 11.11.11

[Motorola 1992] MOTOROLA: *Programmers Reference Manual*, 1992

[NVIDIA 2010] NVIDIA: *NVIDIA Tesla GPUs Power World's Fastest Supercomputer*. 2010. – Verfügbar Online unter `http://pressroom.nvidia.com/easyir/customrel.do?easyirid=A0D622CE9F579F09&version=live&prid=678988&releasejsp=release_157`; 15.12.10

[Olson-Laboratory 2010] OLSON-LABORATORY: *FightAIDS@Home*. 2010. – Verfügbar Online unter `http://fightaidsathome.scripps.edu/`; 2.01.11

[Oxford 2010] OXFORD, University: *The world's largest climate forecasting experiment for the 21st century.* 2010. – Verfügbar Online unter `http://climateprediction.net/`; 15.12.10

[Raikovich und Feher 2010] RAIKOVICH, T. ; FEHER, B.: Application of partial reconfiguration of FPGAs in image processing. In: *Ph.D. Research in Microelectronics and Electronics (PRIME), 2010 Conference on*, july 2010, S. 1 –4

[Raman 2010] RAMAN, Manikandan: China Makes World's Fastest Supercomputer. In: *International Business Times* (2010). – Verfügbar Online unter `http://www.ibtimes.com/articles/76731/20101028/tianhe-1a-tianhe-supercomputer-fastest-supercomputer-\china-ustri-nvidia-amd-gpum-cpu-chip-semiconductor.htm`; 17.12.10

[Rouvroy u. a. 2003] ROUVROY, Gaël ; ST, François xavier ; QUISQUATER, Jean jacques ; LEGAT, Jean didier ; GROUP, Ucl C.: Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES. In: *In FieldProgrammable Logic and Applications - FPL*, Springer-Verlag, 2003, S. 181–193

[Sahak 2011] SAHAK, Edris: *SoC-basierte partielle Rekonfiguration einer modularisierten Bildverarbeitungspipeline*, HAW-Hamburg, Diplomarbeit, 2011

[SpaceDaily 2004] SPACEDAILY: Both Civil and Military Needs Driving European UAV Market. In: *SpaceDaily* (2004). – Verfügbar Online unter http://www.spacedaily.com/news/uav-04a.html; 11.07.10

[Wikipedia 2010] WIKIPEDIA: *Supercomputer — Wikipedia, Die freie Enzyklopädie*. 2010. – Verfügbar Online unter http://de.wikipedia.org/w/index.php?title=Supercomputer&oldid=82613882; 15.12.10

[Wikipedia 2011] WIKIPEDIA: *Fingerabdruck — Wikipedia, Die freie Enzyklopädie*. 2011. – URL http://de.wikipedia.org/w/index.php?title=Fingerabdruck&oldid=95601502. – [Online; Stand 14. November 2011]

[Wilken 2011] WILKEN, Heiko: *Projektbericht: Multiprocessor System-on-Chip, Dual MicroBlaze Implementierung auf einem Spartan-3A*, HAW-Hamburg, Diplomarbeit, 2011

[Wooldridge und Jennings 1995] WOOLDRIDGE, M. ; JENNINGS, N.: Intelligent agents: theory and practice. In: *Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115– 152

[Xilinx 2006a] XILINX: *ML401/ML402/ML403 Evaluation Platform*, 2006. – Verfügbar Online unter http://www.xilinx.com/support/documentation/boards_and_kits/ug080.pdf; 15.11.2011

[Xilinx 2006b] XILINX: *Programming Modern FPGAs*. 2006. – Verfügbar Online unter http://www.xilinx.com/univ/mpsoc2006keynote.pdf; 15.12.10

[Xilinx 2008a] XILINX: *Partial Reconfiguration Software Training*. 2008

[Xilinx 2008b] XILINX: *PLBV46 Master Single*, 2008

[Xilinx 2008c] XILINX: *PLBV46 SLAVE SINGLE*, 2008

[Xilinx 2008d] XILINX: *System ACE CompactFlash Solution*, 2008

[Xilinx 2009a] XILINX: *ML605 Schematics*. 2009. – Verfügbar Online unter http://www.xilinx.com/support/documentation/boards_and_kits/xtp052_ml605_schematics.pdf; 22.10.2011

[Xilinx 2009b] XILINX: *Processor Local Bus (PLB) v3.4*, 2009

[Xilinx 2010a] XILINX: *Getting Started with the Xilinx Virtex-6 FPGA ML605 Evaluation Kit*, 2010. – Verfügbar Online unter http://www.xilinx.com/products/boards/ml605/reference_designs.htm; 21.02.2011

[Xilinx 2010b] XILINX: *LogiCORE IP XPS LL TEMAC*, 2010

[Xilinx 2010c] XILINX: *Multi-Port Memory Controller(MPMC)*, 2010

[Xilinx 2010d] XILINX: *PlanAhead Software Tutorial Overview of the Partial Reconfiguration Flow*, 2010. – Verfügbar Online unter http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/PlanAhead_Tutorial_Overview_of_Partial_Reconfiguration_Flow.pdf; 07.11.11

[Xilinx 2010e] XILINX: *PlanAhead Software Tutorial Partial Reconfiguration of a Processor Peripheral*, 2010

[Xilinx 2010f] XILINX: *Virtex-4 Family Overview*, 2010. – Verfügbar Online unter http://www.xilinx.com/support/documentation/virtex-4_data_sheets.htm; 21.11.2011

[Xilinx 2010g] XILINX: *Virtex-6 Family Overview*, 2010. – Verfügbar Online unter http://www.xilinx.com/products/virtex6/lxt.htm; 21.02.2011

[Xilinx 2010h] XILINX: *Xilinx FPGA Embedded Memory Advantages*, 2010. – Verfügbar Online unter http://www.xilinx.com/support/documentation/white_papers/wp360.pdf; 11.11.11

[Xilinx 2010i] XILINX: *XLogiCORE IP XPS HWICAP*. v.5.0.a, 2010. – Verfügbar Online unter http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf; 22.05.11

[Xilinx 2010j] XILINX: *XPS SYSACE (System ACE) Interface Controller*, 2010

[Xilinx 2011a] XILINX: *Constraints Guide*, 2011

[Xilinx 2011b] XILINX: *Data2MEM User Guide*, 2011

[Xilinx 2011c] XILINX: *LibXil FATFile System (FATFS)*, 2011

[Xilinx 2011d] XILINX: *LibXil Memory File System (MFS)*, 2011

[Xilinx 2011e] XILINX: *LogiCORE IP Fast Simplex Link (FSL) V20 Bus*, 2011

[Xilinx 2011f] XILINX: *lwIP 1.3.0 Library*, 2011

[Xilinx 2011g] XILINX: *MicroBlaze Processor Reference Guide*, 2011

[Xilinx 2011h] XILINX: *ML605 Hardware User Guide*, 2011

[Xilinx 2011i] XILINX: *Virtex-6 FPGA Embedded TEMAC Solution*, 2011

[Xilinx 2011j] XILINX: *Virtex-6 FPGA ML605 Evaluation Kit*, 2011. – Verfügbar Online unter http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm; 07.11.2011

[Xilinx 2011k] XILINX: *XAPP1026, LightWeight IP (lwIP) Application Examples*, 2011

[Xilinx 2011l] XILINX: *Xilkernel*, 2011

[XStream 2011] XSTREAM: *XStream*. Website. 2011. – Verfügbar Online unter http://xstream.codehaus.org/; 24.09.11

Abbildungsverzeichnis

1 Architektur des Distributed Computing Systems mit den SoC-Clienten sowie den zu bearbeitenden Projekt und der Kommunikation über einen Server, welcher die Verteilung der Rechenleistung vornimmt. 3
2 Systemaufbau des SoC-Clients mit den Modulen zur Ansteuerung des Speichers, des dynamischen Bereiches sowie der Kommunikation zum Server . . . 4
3 Hardware Architektur des COPACOBANA Systems [Guneysu u. a. (2008)] . . 7
4 Anordnung der DES-Module in einem FPGA mit geteiltem Schlüsselraum und Datensätzen [Guneysu u. a. (2008)] 8
5 Rekonfigurierbare Pipeline zur Bilddatenverarbeitung [Raikovich und Feher (2010)] 10
6 Ablauf der Zugriffsgewährung mit der Enrolment Phase zum Speichern der Merkmale sowie der Authentication zum Gewähren des Zugriffs bei einer Anfrage [Fons und Fons (2011)] 11
7 Ergebnisse der einzelnen Bildverarbeitungsschritte zur Merkmalserkennung der gespeicherten Bildes (blau) und der Anfrage (rot) zum Merkmalsvergleich [Fons und Fons (2011)] 12
8 Fingerabdruck Authentifizierungssystem basierend auf einer SoC Plattform mit der Fähigkeit zur dynamischen partiellen Rekonfigruation [Fons und Fons (2011)] 15
9 SoC Netzwerkarchitektur bestehend aus dem Netzwerk mit den HW-Module sowie dem Memory-Arbiter. Der Host-Arbiter wird durch einen GPP implementiert. [Kearney und Jasiunas (2006)] 17
10 Blockschaltbild des ML605 Entwicklungsboard mit den verwendeten Anschlüssen der Peripherie im FPGA [Xilinx (2011h)] 21
11 Standardmäßig vorhandene (weiß) und optionale (grau) Komponenten des MicroBlaze Mikroprozessors [Xilinx (2011g)] 21
12 Entwurfsablauf zum Erstellen eines dynamisch partiell rekonfigurierbaren Systems 23
13 *XPS_LL_TEMAC* IP mit Registern für die Ethernet Hard- bzw. Soft-IPs [Xilinx (2010b)] 24
14 Datenpfad des TEMAC IPs [Xilinx (2011i)] 24
15 Interne Register und Interfaces des XPS HWICAP Modules [Xilinx (2010i)] . . 25
16 FPGA Konfigurationsschaltung mit System ACE und Option zur Konfiguration des FPGAs über einen externen JTAG Anschluss [Xilinx (2008d)] 27
17 Scheduling des Xilkernels bei aktiviertem Prioritätsscheduling. Jede Prioritätsstufe enthält eine Queue, welche die jeweiligen Threads beinhaltet [Xilinx (2011l)]. 28
18 Minimal Architektur eines SoCs mit xps_ll_temac Modul und SDMA [Xilinx (2011f)]. 29
19 Aufbau des DSN unter der Verwendung der „Client-Server“ Architektur 32
20 Client-Broker-Client Architektur des DSN mit zentraler Stelle für alle Projekte und SoCs zur optimalen Verteilung der Ressourcen auf die Projekte. 33
21 Minimaler Systemaufbau des SoC-Clients zur Erfüllung der Aufgaben im DSN 35
22 FiFo basiertes Interfaces zwischen dem dynamischen Bereich und dem Bus System 35
23 Nachricht des 6 Byte Overlay Protokolls, welches zwischen Server und SoC-Client versendet wird. Wobei das erste Byte die Art der Übertragung, also PRM, Datensatz oder Antwort, beinhaltet. Das zweite Feld beinhaltet die Phase im Protokoll und die folgenden 4 Bytes die Länge der zu übertragenen Daten. . . 36

24 Das Kommunikationsprotokoll zur Übertragung der Daten zwischen Soc und Server aus Sicht des Servers. 37
25 Datenformat der Eingangsdaten, welche zur Berechnung bereitgestellt werden . 37
26 Use-Case des Projekt-Clients mit der Interaktion des Anwenders sowie der Kommunikation zwischen dem Project-Client und dem Broker (vgl. Abbildung 20). 38
27 Klassendiagramm der Interfaces *Connection_I*, *Manager_I* und *Persistenz_I* zur Realisierung der Projekt-Clienten mit den zur Erfüllung des Use-Cases verwendeten Funktionen. 38
28 Attribute der zur Übertragung verwendeten DataFile Klasse in vereinfachter Form zur besseren Darstellung ohne Getter/Setter Methoden, welche für die Erzeugung der XML Dateien genutzt werden [XStream (2011)]. 39
29 Aufbau des DSN unter der Verwendung der Client-Server Architektur 40
30 Use-Case des Brokers mit der Kommunikation zwischen den SoC-Clients und den Projekten. 42
31 SoC-Client mit Komponenten des Xilinx EDK 44
32 Aufbau des SoC Systems im System Assembly View des EDK 44
33 Adressbelegung der Funktionselemente am PLB sowie der angeschlossenen Bus-Systeme im SoC-Client 44
34 Interner Aufbau des SDMA Ports mit LokalLink zum xps_ll_temac, NPI zur Ansteuerung des MPMC und dem PLB zur Konfiguration [Xilinx (2010c)]. . . 45
35 Belegung der Logik Elemente nach dem Map-Prozess mit dem FORCE Parameter, dargestellt im FPGA-Editor 48
36 Belegung der Logik Elemente nach dem Map-Prozess mit dem TRUE Parameter, dargestellt im FPGA-Editor 48
37 Verbindung des Input-Buffers mit dem AN13 Pin des *fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin[0]* Eingangssignal 48
38 Standard Interface eines PLB Slave IPs [Xilinx (2008c)] 49
39 Signal Erweiterung des Slave PLB IPs mit dem Master Interface [Xilinx (2008b)] 49
40 *Dyn_Interface* zur Kommunikation zwischen dem PLB und der PRR 50
41 Zustandsdiagramm des *Dyn_Interface* IPs zur Ansteuerung des PLB (vgl. Anhang F) 51
42 Dynamischer Bereich des SoC-Clients als grafische Darstellung im PlanAhead 53
43 Verteilung der genutzten Ressourcen im FPGA ohne definierter PRR, dargestellt im FPGA-Editor 54
44 Verteilung der genutzten Ressourcen im FPGA mit definierter PRR und einer aktiven PRM (gelber Kreis) 54
45 Start des SoC-Clients mit der Initialisierung des Xilkernels und der anschließenden Adressierung des Ethernet Interfaces sowie dem Starten des SoC-Client Threads und dem LwIP. 56
46 SoC-Client mit der Initialisierungsphase für die Daten sowie dem blockierendem Lesen der Socket Schnittstelle und dem warten auf Anfrage mit dem in Kapitel 4.1 definiertem Protokoll. 56
47 Petri-Netz Beschreibung des gegenseitigen Ausschlusses zur Synchronisation der Datensätze sowie der Zugriffe auf das Verwaltungsarray beim Empfangen der Daten im SoC-Client sowie der Verarbeitung und dem anschließenden Versenden 57

48 Compute-Thread zum Verarbeiten der Daten mit der Synchronisation über die Mutexe und der Interaktion mit dem *DYN_Interface*. 58
49 Send-Thread zum Versenden der bereits verarbeiteten Daten 58
50 Datenverwaltung im SoC-Client für die Ein- und Ausgangsdaten im Beispiel mit fünf Datensätzen und einer jeweiligen Größe von 12 MByte 61
51 Erzeugen einer SREC Datei mit dem Flash Memory Tool des Xilinx SDK . . . 62
52 Kommunikation innerhalb des Servers über Funktionen und das Observer Pattern, sowie die Eigenschaften der einzelnen Klassen wie dem Speichern der SoC-Clienten und den wichtigsten Funktionen. 66
53 Ablaufdiagramm der Client-Factory und der Interaktion mit dem Broker und den angemeldeten SoC-Clients 68
54 State Pattern Implementierung zur Kommunikation des Brokers mit den SoC-Clients 69
55 Modifizierter Automat zur Kommunikation mit den SoC-Clienten (vgl. Abbildung 54) 70
56 Ablaufdiagramm der Project-Factory und der Interaktion mit dem Broker und angemeldeten Projekten 71
57 Bildstromverarbeitungs-Komponenten des Zielsystems im autonomen Fahrzeug [Sahak (2011)] 74
58 Kontrollsignale des Bilddatenstromes im autonomen Fahrzeug auf Basis von Zählern [Sahak (2011)] 75
59 Entity der Filter IPs 75
60 Wrapper der Filter-Entity für das *DYN_Interface* mit Demultiplexing, dem Steuerautomaten sowie dem Konkatenieren der Ergebnisse zur Anpassung an den 32 Bit Ausgangsvektor 76
61 Zustandsdiagramm zum Erzeugen der Steuersignale für die Filter-Module sowie dem Lesen und Schreiben der Bilddaten 76
62 Komponenten des Beispiel-Projekts und der Interaktion mit dem Anwender . . 77
63 Beispiel der Konfiguration im DSN mit zwei SoCs und der Verarbeitung eines Projektes mit vier Konfigurationen 78
64 Eingangstestbild 79
65 Ausgabe des Binären-Filters 79
66 Ausgabe des Erosions-Filters 79
67 Ausgabe des Median-Filters 79
68 Ausgabe des Sobel-Filters 79
69 Messchaltung zur Analyse des Zeitverhaltens im FPGA [Xilinx (2009a)] . . . 80
70 Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform am Beispiel eines Projektes und eines SoCs i
71 Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform zwischen einem Projekt und dem Broker mit der Interaktion des Anwenders ii
72 Utilisation Diagramm des SoC-Clients mit Implementierung durch PlanAhead . iii

A Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform

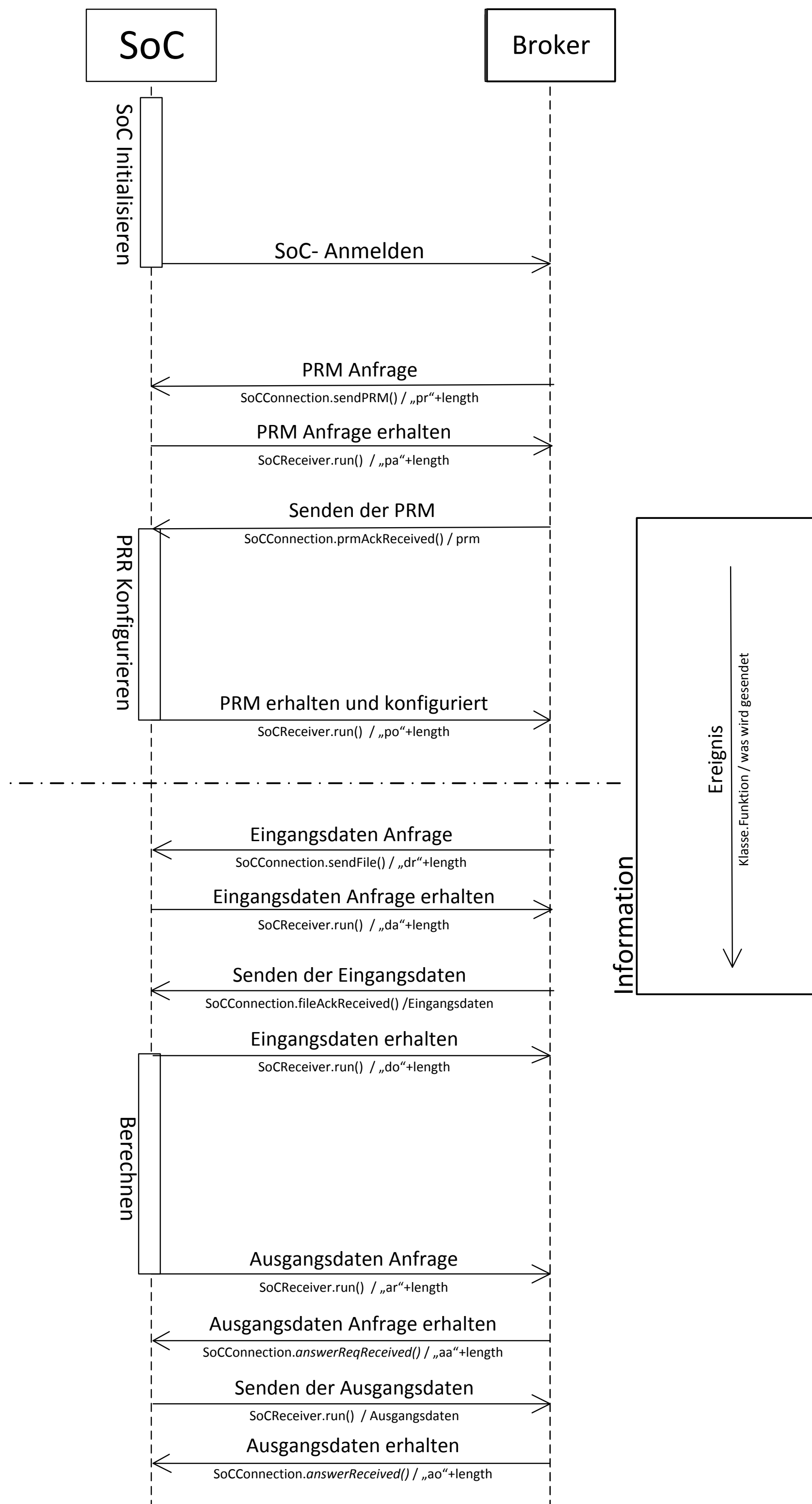


Abb. 70: *Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform am Beispiel eines Projektes und eines SoCs*

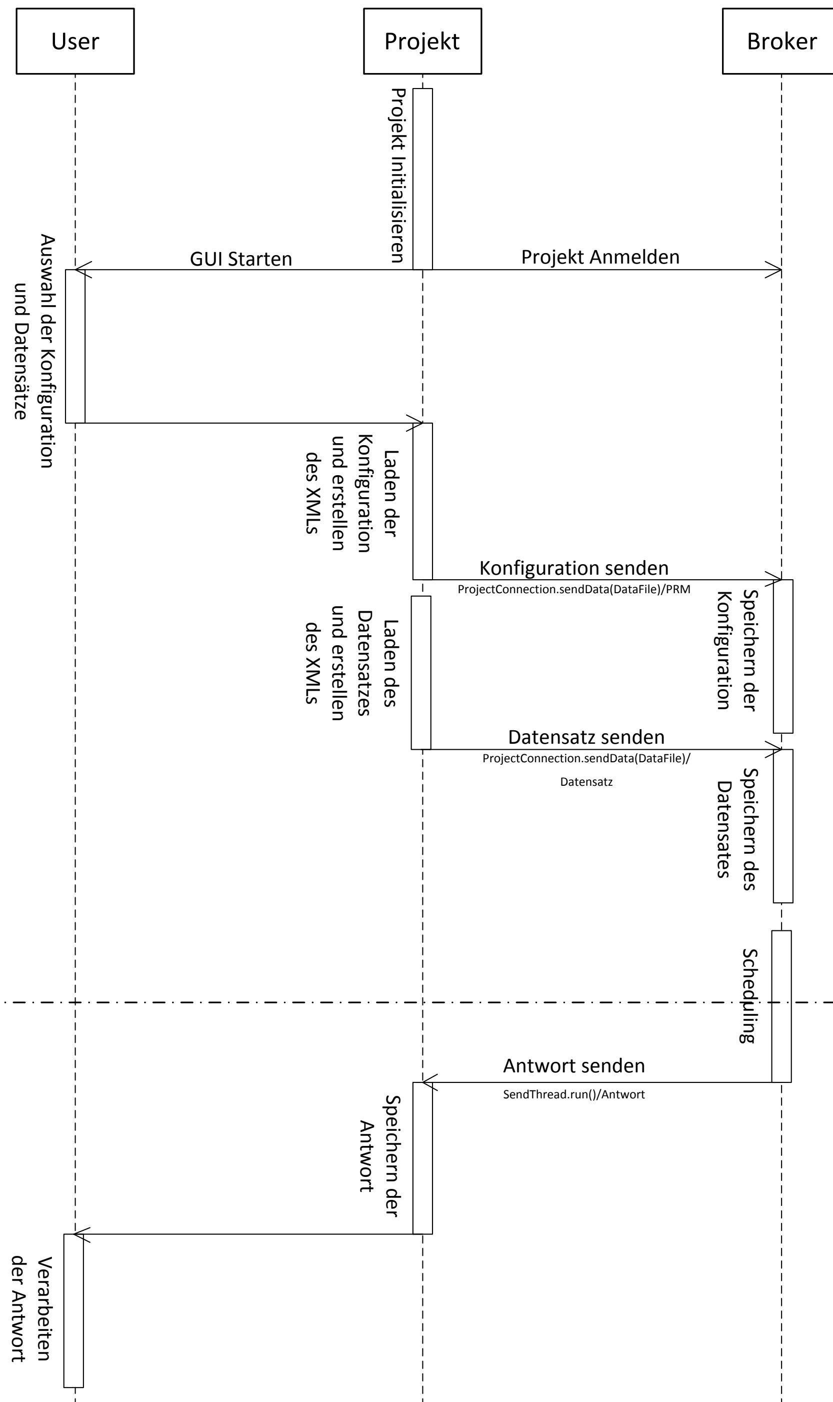


Abb. 71: *Ablaufdiagramm der FPGA-basierten Distributed Computing Plattform zwischen einem Projekt und dem Broker mit der Interaktion des Anwenders*

B Ressourcenbedarf des statischen SoC-Clients

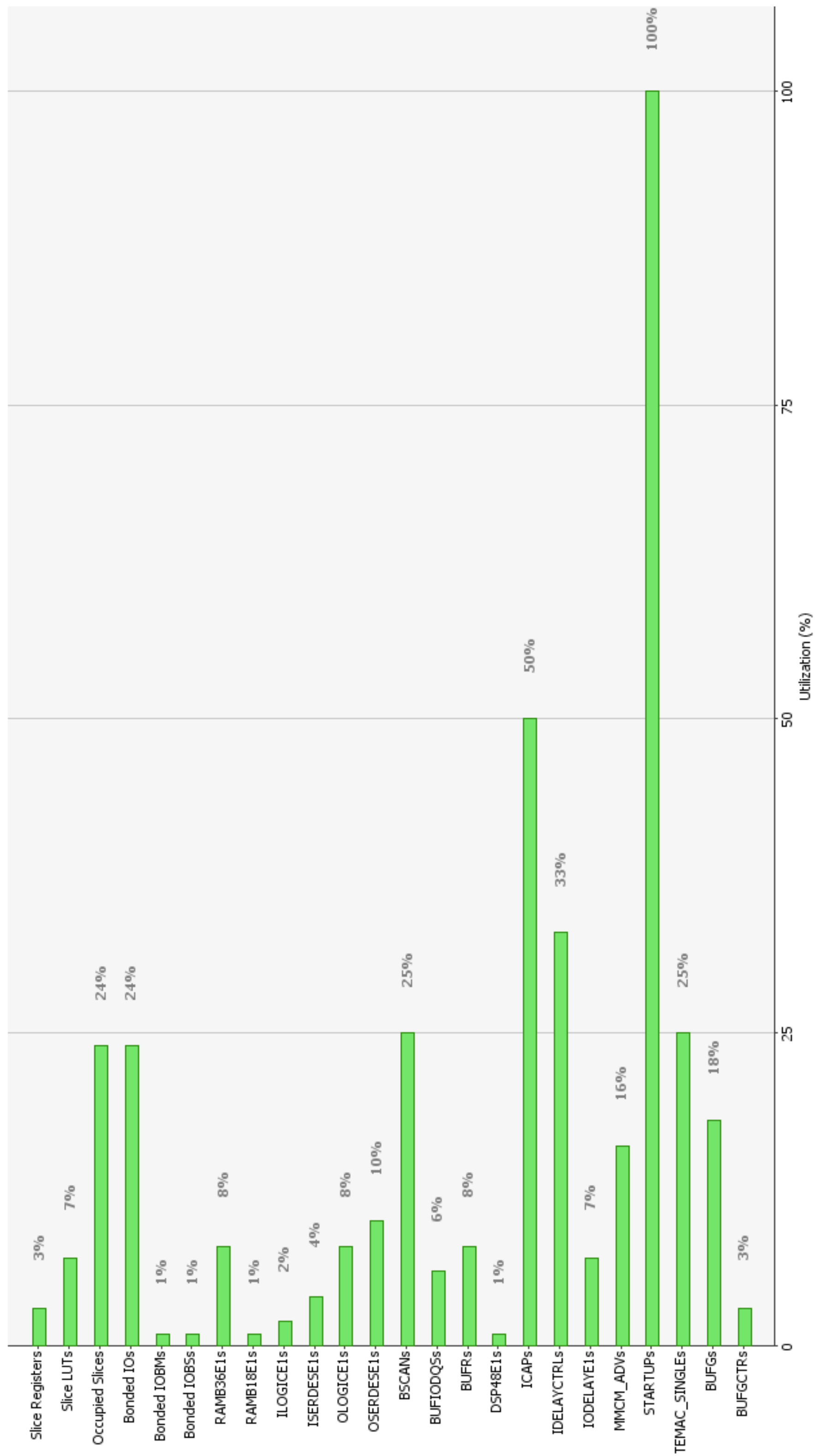


Abb. 72: *Utilisation Diagramm des SoC-Clients mit Implementierung durch PlanAhead*

C Messung zum Datendurchsatz der XILMFS und XILFATFS Dateisysteme

Die Wahl des zu verwendenden Dateisystems ist auch von dem zu erreichendem Datendurchsatz, welcher in $\frac{byte}{sekunde}$ angegeben wird, abhängig. Die zur Ausführung der Schreib-Funktion genutzten Takte werde über das Starten eines Timers vor dem Aufruf und das Stoppen bei der Rückkehr gemessen (vgl. Listing 13). Die, bei der Ausführung des Messprogramms entstandenen Ausgaben, geben die jeweiligen Takte an, die für die vorgegebenen Anzahl an zu schreibenden Bytes benötigt wurde (vgl. Listing 14).

Berechnet wird der Durchsatz mit Formel 2, wobei b die Anzahl der Bytes, a die Anzahl der Takt und f die Frequenz des Timers darstellt.

$$\frac{MByte}{Sekunde} = \frac{\frac{b}{a\frac{1}{f}}}{1000000} = \frac{bf}{1000000a} \tag{2}$$

Das Ergebnis der Messung zeigt, dass das XILMFS eine Datendurchsatz hat, der bis zu 20 mal größer ist, als das XILFATFS [5].

Tabelle 13: *Datendurchsatz der zur Verfügung stehenden Dateisysteme*

Anzahl Bytes	Takte		Datendurchsatz ($\frac{Mbyte}{sekunde}$)	
	XILMFS	XILFATFS	XILMFS	XILFATFS
10	1202	576414	0,831	0,831
100	4024	104365	2,485	0,095
1000	42861	497505	2,333	0,201
10000	455514	4655364	2,195	0,214
100000	4619489	87329276	2,164	0,114
1000000	46224443	883079075	2,163	0,113
10000000	462441969	0	2,162	–

Listing 13: *Messung des Datendurchsatzes mit dem XILFMS und XILFATFS Dateisystem*

```
0
  #include <stdio.h>
  #include "platform.h"
  #include "xtmrctr.h"
  #include "xparameters.h"
5 #include "xilmfs.h"
  #include "xsysace_l.h"
  #include "xsysace.h"
  #include "sysace_stdio.h"

10
  #define TMRCTR_DEVICE_ID          XPAR_TMRCTR_0_DEVICE_ID
  #define TIMER_COUNTER_0   0
  #define BUFFER_LENTH  10
  #define TEST_LENGTH  50000000
15 #define FILENAME  "memtest.txt"


  extern int __testsection_start;

20 int main()
  {
          int Status;
```

[5] Die letzte Messung des XILFATFS hat einen Überlauf des Timer Registers erzeugt, wodurch dieses auf 0 gesetzt wurde.

```
        int i=0,timer_value, fd,write_count;
        char buffer[BUFFER_LENTH];
        XSysAce SysAce;
        SYSACE_FILE *stream;
        XTmrCtr timer;

    init_platform();

    for(i=0;i<BUFFER_LENTH;i++)buffer[i]= (char)i;


    mfs_init_fs(TEST_LENGTH, &__testsection_start, MFSINIT_NEW);


        Status = XSysAce_Initialize(&SysAce, XPAR_SYSACE_0_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                print("Failure␣while␣initializing␣sysace");
                return XST_FAILURE;
        }

        Status = XTmrCtr_Initialize(&timer, TMRCTR_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                print("Failure␣while␣initializing␣timer");
                return XST_FAILURE;
        }

        print("Starting␣XILMFS␣\n\r␣");

        for ( i=10; i < TEST_LENGTH; i*=10){
                fd = mfs_file_open(FILENAME,MFS_MODE_CREATE);
                write_count = 0;
                XTmrCtr_Start(&timer , TIMER_COUNTER_0);

                while(write_count < i){
                        if( 1 != mfs_file_write(fd,buffer,BUFFER_LENTH)){
                                print("Failure␣while␣writing␣into␣XILMFS␣\n\r␣");
                                return XST_FAILURE;
                        }
                        write_count += BUFFER_LENTH;
                }

                XTmrCtr_Stop(&timer , TIMER_COUNTER_0);

                mfs_file_close(fd);
                mfs_delete_file(FILENAME);

                timer_value = XTmrCtr_GetValue(&timer, TIMER_COUNTER_0);
                xil_printf("XILMFS␣:␣\t␣Measure␣Clock␣Cycles␣Xilmfs␣:␣\t␣%d␣by␣%d␣written␣
                    bytes␣\r␣\n␣" , timer_value,i );
        }

        print("XILMFS␣DONE␣\r␣\n");
        print("Starting␣XILFATFS␣\r␣\n");

        for ( i=10; i < TEST_LENGTH; i*=10){
            if ((stream = sysace_fopen(FILENAME, "w")) == NULL) {
                xil_printf("Can't␣open␣file␣(%s)\r\n", FILENAME);
                return XST_FAILURE;
            }
            write_count = 0;
                XTmrCtr_Start(&timer , TIMER_COUNTER_0);

                while(write_count < i){
                        if( -1 == sysace_fwrite(buffer, 1, BUFFER_LENTH, stream )){
                                print("Failure␣while␣writing␣into␣XILFATFS");
                                return XST_FAILURE;
                        }
                        write_count += BUFFER_LENTH;
                }

                XTmrCtr_Stop(&timer , TIMER_COUNTER_0);

                sysace_fclose(stream);
                sysace_remove_file(FILENAME);

                timer_value = XTmrCtr_GetValue(&timer, TIMER_COUNTER_0);
                xil_printf("XILFATFS␣:␣\t␣Measure␣Clock␣Cycles␣XILFATFS␣:␣\t␣%d␣by␣%d␣written␣
                    bytes␣\r␣\n␣" , timer_value,i );
        }
```

```
100         print("XILMFS␣Done");

        cleanup_platform();

        return 0;
105 }
```

Listing 14: *Ausgabe des Messprogramms mit den benötigten Takte*

```
0  Starting XILMFS
   XILMFS: Measure Clock Cycles Xilmfs : 1202 by 10 written bytes
   XILMFS: Measure Clock Cycles Xilmfs : 4024 by 100 written bytes
   XILMFS: Measure Clock Cycles Xilmfs : 42861 by 1000 written bytes
   XILMFS: Measure Clock Cycles Xilmfs : 455514 by 10000 written bytes
5  XILMFS: Measure Clock Cycles Xilmfs : 4619489 by 100000 written bytes
   XILMFS: Measure Clock Cycles Xilmfs : 46224443 by 1000000 written bytes
   XILMFS: Measure Clock Cycles Xilmfs : 462441969 by 10000000 written bytes
   XILMFS DONE
   Starting XILFATFS
10 XILFATFS: Measure Clock Cycles XILFATFS : 576414 by 10 written bytes
   XILFATFS: Measure Clock Cycles XILFATFS : 104365 by 100 written bytes
   XILFATFS: Measure Clock Cycles XILFATFS : 497505 by 1000 written bytes
   XILFATFS: Measure Clock Cycles XILFATFS : 4655364 by 10000 written bytes
   XILFATFS: Measure Clock Cycles XILFATFS : 87329276 by 100000 written bytes
15 XILFATFS: Measure Clock Cycles XILFATFS : 883079075 by 1000000 written bytes
   XILFATFS: Measure Clock Cycles XILFATFS : 0 by 10000000 written bytes //Überlauf des Zählers
   XILFATFS Done
```

D Hardware Konfiguration des MicroBlaze Systems

Listing 15: *MHS-Datei des statischen MicroBlaze Systems*

```
# ##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 12.1 Build EDK_MS1.53c
# Thu Apr 22 11:24:17 2010
# Target Board: Xilinx Virtex 6 ML605 Evaluation Platform Rev D
# Family:    virtex6
# Device:    xc6vlx240t
# Package:   ff1156
# Speed Grade:  -1
# Processor number: 1
# Processor 1: microblaze_0
# System clock frequency: 100.0
# Debug Interface: On-Chip HW Debug Module
# ##############################################################################
 PARAMETER VERSION = 2.1.0


 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX_pin, DIR = I
 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_Clk_pin = fpga_0_DDR3_SDRAM_DDR3_Clk_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin = fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_CE_pin = fpga_0_DDR3_SDRAM_DDR3_CE_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_CS_n_pin = fpga_0_DDR3_SDRAM_DDR3_CS_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_ODT_pin = fpga_0_DDR3_SDRAM_DDR3_ODT_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin = fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin = fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_WE_n_pin = fpga_0_DDR3_SDRAM_DDR3_WE_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin = fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin, DIR = O, VEC
     = [2:0]
 PORT fpga_0_DDR3_SDRAM_DDR3_Addr_pin = fpga_0_DDR3_SDRAM_DDR3_Addr_pin, DIR = O, VEC = [12:0]
 PORT fpga_0_DDR3_SDRAM_DDR3_DQ_pin = fpga_0_DDR3_SDRAM_DDR3_DQ_pin, DIR = IO, VEC = [31:0]
 PORT fpga_0_DDR3_SDRAM_DDR3_DM_pin = fpga_0_DDR3_SDRAM_DDR3_DM_pin, DIR = O, VEC = [3:0]
 PORT fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin = fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin, DIR = O
 PORT fpga_0_DDR3_SDRAM_DDR3_DQS_pin = fpga_0_DDR3_SDRAM_DDR3_DQS_pin, DIR = IO, VEC = [3:0]
 PORT fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin = fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin, DIR = IO, VEC =
     [3:0]
 PORT fpga_0_Hard_Ethernet_MAC_TemacPhy_RST_n_pin =
     fpga_0_Hard_Ethernet_MAC_TemacPhy_RST_n_pin, DIR = O
 PORT fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin = fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin,
     DIR = I
 PORT fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin, DIR =
     O, VEC = [7:0]
 PORT fpga_0_Hard_Ethernet_MAC_GMII_TX_EN_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_TX_EN_0_pin,
     DIR = O
 PORT fpga_0_Hard_Ethernet_MAC_GMII_TX_ER_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_TX_ER_0_pin,
     DIR = O
 PORT fpga_0_Hard_Ethernet_MAC_GMII_TX_CLK_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_TX_CLK_0_pin,
     DIR = O
 PORT fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin, DIR =
     I, VEC = [7:0]
 PORT fpga_0_Hard_Ethernet_MAC_GMII_RX_DV_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_RX_DV_0_pin,
     DIR = I
 PORT fpga_0_Hard_Ethernet_MAC_GMII_RX_ER_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_RX_ER_0_pin,
     DIR = I
 PORT fpga_0_Hard_Ethernet_MAC_GMII_RX_CLK_0_pin = fpga_0_Hard_Ethernet_MAC_GMII_RX_CLK_0_pin,
     DIR = I
 PORT fpga_0_Hard_Ethernet_MAC_MDC_0_pin = fpga_0_Hard_Ethernet_MAC_MDC_0_pin, DIR = O
 PORT fpga_0_Hard_Ethernet_MAC_MDIO_0_pin = fpga_0_Hard_Ethernet_MAC_MDIO_0_pin, DIR = IO
 PORT fpga_0_Hard_Ethernet_MAC_PHY_MII_INT_pin = fpga_0_Hard_Ethernet_MAC_PHY_MII_INT_pin, DIR
     = I, SIGIS = INTERRUPT, SENSITIVITY = LEVEL_LOW, INTERRUPT_PRIORITY = MEDIUM
 PORT fpga_0_clk_1_sys_clk_p_pin = dcm_clk_s, DIR = I, SIGIS = CLK, DIFFERENTIAL_POLARITY = P,
     CLK_FREQ = 200000000
 PORT fpga_0_clk_1_sys_clk_n_pin = dcm_clk_s, DIR = I, SIGIS = CLK, DIFFERENTIAL_POLARITY = N,
     CLK_FREQ = 200000000
 PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 1
 PORT fpga_0_SysACE_CompactFlash_SysACE_MPA_pin = fpga_0_SysACE_CompactFlash_SysACE_MPA_pin,
     DIR = O, VEC = [6:0]
 PORT fpga_0_SysACE_CompactFlash_SysACE_CLK_pin = fpga_0_SysACE_CompactFlash_SysACE_CLK_pin,
     DIR = I
 PORT fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin =
     fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin, DIR = I
 PORT fpga_0_SysACE_CompactFlash_SysACE_CEN_pin = fpga_0_SysACE_CompactFlash_SysACE_CEN_pin,
     DIR = O
```

```
    PORT fpga_0_SysACE_CompactFlash_SysACE_OEN_pin = fpga_0_SysACE_CompactFlash_SysACE_OEN_pin ,
         DIR = O
55  PORT fpga_0_SysACE_CompactFlash_SysACE_WEN_pin = fpga_0_SysACE_CompactFlash_SysACE_WEN_pin ,
         DIR = O
    PORT fpga_0_SysACE_CompactFlash_SysACE_MPD_pin = fpga_0_SysACE_CompactFlash_SysACE_MPD_pin ,
         DIR = IO , VEC = [7:0]
    PORT fpga_0_FLASH_Mem_A_pin = fpga_0_FLASH_Mem_A_pin_vslice_7_30_concat , DIR = O, VEC =
         [7:30]
    PORT fpga_0_FLASH_Mem_OEN_pin = fpga_0_FLASH_Mem_OEN_pin , DIR = O
    PORT fpga_0_FLASH_Mem_WEN_pin = fpga_0_FLASH_Mem_WEN_pin , DIR = O
60  PORT fpga_0_FLASH_Mem_DQ_pin = fpga_0_FLASH_Mem_DQ_pin , DIR = IO , VEC = [0:15]
    PORT fpga_0_FLASH_CE_inverter_Res_pin = fpga_0_FLASH_CE_inverter_Res_pin , DIR = O


   BEGIN microblaze
65  PARAMETER INSTANCE = microblaze_0
    PARAMETER C_DEBUG_ENABLED = 1
    PARAMETER C_ICACHE_BASEADDR = 0x90000000
    PARAMETER C_ICACHE_HIGHADDR = 0x9fffffff
    PARAMETER C_CACHE_BYTE_SIZE = 32768
70  PARAMETER C_ICACHE_ALWAYS_USED = 1
    PARAMETER C_DCACHE_BASEADDR = 0x90000000
    PARAMETER C_DCACHE_HIGHADDR = 0x9fffffff
    PARAMETER C_DCACHE_BYTE_SIZE = 32768
    PARAMETER C_DCACHE_ALWAYS_USED = 1
75  PARAMETER HW_VER = 7.30.a
    PARAMETER C_USE_ICACHE = 1
    PARAMETER C_USE_DCACHE = 1
    PARAMETER C_DCACHE_USE_WRITEBACK = 1
    PARAMETER C_USE_BARREL = 1
80  PARAMETER C_DPLB_BUS_EXCEPTION = 1
    PARAMETER C_IPLB_BUS_EXCEPTION = 1
    PARAMETER C_ILL_OPCODE_EXCEPTION = 1
    PARAMETER C_UNALIGNED_EXCEPTIONS = 1
    PARAMETER C_OPCODE_0x0_ILLEGAL = 1
85  BUS_INTERFACE DPLB = mb_plb
    BUS_INTERFACE IPLB = mb_plb
    BUS_INTERFACE DXCL = microblaze_0_DXCL
    BUS_INTERFACE IXCL = microblaze_0_IXCL
    BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
90  BUS_INTERFACE DLMB = dlmb
    BUS_INTERFACE ILMB = ilmb
    PORT MB_RESET = mb_reset
    PORT INTERRUPT = microblaze_0_Interrupt
   END
95
   BEGIN plb_v46
    PARAMETER INSTANCE = mb_plb
    PARAMETER HW_VER = 1.04.a
    PORT PLB_Clk = clk_100_0000MHzMMCM0
100 PORT SYS_Rst = sys_bus_reset
   END

   BEGIN lmb_v10
    PARAMETER INSTANCE = ilmb
105 PARAMETER HW_VER = 1.00.a
    PORT LMB_Clk = clk_100_0000MHzMMCM0
    PORT SYS_Rst = sys_bus_reset
   END

110 BEGIN lmb_v10
    PARAMETER INSTANCE = dlmb
    PARAMETER HW_VER = 1.00.a
    PORT LMB_Clk = clk_100_0000MHzMMCM0
    PORT SYS_Rst = sys_bus_reset
115 END

   BEGIN lmb_bram_if_cntlr
    PARAMETER INSTANCE = dlmb_cntlr
    PARAMETER HW_VER = 2.10.b
120 PARAMETER C_BASEADDR = 0x00000000
    PARAMETER C_HIGHADDR = 0x0000ffff
    BUS_INTERFACE SLMB = dlmb
    BUS_INTERFACE BRAM_PORT = dlmb_port
   END
125
   BEGIN lmb_bram_if_cntlr
    PARAMETER INSTANCE = ilmb_cntlr
    PARAMETER HW_VER = 2.10.b
```

```
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = ilmb
 BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
 PARAMETER INSTANCE = lmb_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = ilmb_port
 BUS_INTERFACE PORTB = dlmb_port
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RX = fpga_0_RS232_Uart_1_RX_pin
 PORT TX = fpga_0_RS232_Uart_1_TX_pin
END

BEGIN mpmc
 PARAMETER INSTANCE = DDR3_SDRAM
 PARAMETER C_NUM_PORTS = 4
 PARAMETER C_MMCM_EXT_LOC = MMCM_ADV_X0Y9
 PARAMETER C_USE_MIG_V6_PHY = 1
 PARAMETER C_MEM_TYPE = DDR3
 PARAMETER C_MEM_PARTNO = MT4JSF6464HY-1G1
 PARAMETER C_MEM_ODT_TYPE = 1
 PARAMETER C_MEM_REG_DIMM = 0
 PARAMETER C_MEM_CLK_WIDTH = 1
 PARAMETER C_MEM_CE_WIDTH = 1
 PARAMETER C_MEM_CS_N_WIDTH = 1
 PARAMETER C_MEM_DATA_WIDTH = 32
 PARAMETER C_MEM_NDQS_COL0 = 3
 PARAMETER C_MEM_NDQS_COL1 = 1
 PARAMETER C_MEM_DQS_LOC_COL0 = 0x000000000000000000000000000000020100
 PARAMETER C_MEM_DQS_LOC_COL1 = 0x000000000000000000000000000000000003
 PARAMETER C_PIM0_BASETYPE = 1
 PARAMETER C_PIM1_BASETYPE = 1
 PARAMETER C_PIM2_BASETYPE = 3
 PARAMETER C_SDMA2_PI2LL_CLK_RATIO = 2
 PARAMETER HW_VER = 6.00.a
 PARAMETER C_MPMC_CLK0_PERIOD_PS = 5000
 PARAMETER C_IODELAY_GRP = DDR3_SDRAM
 PARAMETER C_SKIP_SIM_INIT_DELAY = 1
 PARAMETER C_PIM3_BASETYPE = 2
 PARAMETER C_MPMC_BASEADDR = 0x90000000
 PARAMETER C_MPMC_HIGHADDR = 0x9fffffff
 PARAMETER C_SDMA_CTRL_BASEADDR = 0x84600000
 PARAMETER C_SDMA_CTRL_HIGHADDR = 0x8460ffff
 BUS_INTERFACE XCL0 = microblaze_0_IXCL
 BUS_INTERFACE XCL1 = microblaze_0_DXCL
 BUS_INTERFACE SDMA_CTRL2 = mb_plb
 BUS_INTERFACE SDMA_LL2 = Hard_Ethernet_MAC_LLINK0
 BUS_INTERFACE SPLB3 = mb_plb
 PORT SDMA2_Clk = clk_100_0000MHzMMCM0
 PORT SDMA2_Rx_IntOut = DDR3_SDRAM_SDMA2_Rx_IntOut
 PORT SDMA2_Tx_IntOut = DDR3_SDRAM_SDMA2_Tx_IntOut
 PORT MPMC_Clk0 = clk_200_0000MHzMMCM0
 PORT MPMC_Clk_200MHz = clk_200_0000MHzMMCM0
 PORT MPMC_Rst = sys_periph_reset
 PORT MPMC_Clk_Mem = clk_400_0000MHzMMCM0
 PORT MPMC_Clk_Rd_Base = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT MPMC_DCM_PSEN = MPMC_DCM_PSEN
 PORT MPMC_DCM_PSINCDEC = MPMC_DCM_PSINCDEC
 PORT MPMC_DCM_PSDONE = MPMC_DCM_PSDONE
 PORT DDR3_Clk = fpga_0_DDR3_SDRAM_DDR3_Clk_pin
 PORT DDR3_Clk_n = fpga_0_DDR3_SDRAM_DDR3_Clk_n_pin
 PORT DDR3_CE = fpga_0_DDR3_SDRAM_DDR3_CE_pin
 PORT DDR3_CS_n = fpga_0_DDR3_SDRAM_DDR3_CS_n_pin
 PORT DDR3_ODT = fpga_0_DDR3_SDRAM_DDR3_ODT_pin
 PORT DDR3_RAS_n = fpga_0_DDR3_SDRAM_DDR3_RAS_n_pin
```

```
 PORT DDR3_CAS_n = fpga_0_DDR3_SDRAM_DDR3_CAS_n_pin
 PORT DDR3_WE_n = fpga_0_DDR3_SDRAM_DDR3_WE_n_pin
 PORT DDR3_BankAddr = fpga_0_DDR3_SDRAM_DDR3_BankAddr_pin
 PORT DDR3_Addr = fpga_0_DDR3_SDRAM_DDR3_Addr_pin
 PORT DDR3_DQ = fpga_0_DDR3_SDRAM_DDR3_DQ_pin
 PORT DDR3_DM = fpga_0_DDR3_SDRAM_DDR3_DM_pin
 PORT DDR3_Reset_n = fpga_0_DDR3_SDRAM_DDR3_Reset_n_pin
 PORT DDR3_DQS = fpga_0_DDR3_SDRAM_DDR3_DQS_pin
 PORT DDR3_DQS_n = fpga_0_DDR3_SDRAM_DDR3_DQS_n_pin
END

BEGIN xps_ll_temac
 PARAMETER INSTANCE = Hard_Ethernet_MAC
 PARAMETER C_NUM_IDELAYCTRL = 6
 PARAMETER C_IDELAYCTRL_LOC = IDELAYCTRL_X2Y1-IDELAYCTRL_X2Y2-IDELAYCTRL_X2Y3-IDELAYCTRL_X1Y1-
     IDELAYCTRL_X1Y2-IDELAYCTRL_X1Y3
 PARAMETER C_PHY_TYPE = 1
 PARAMETER C_TEMAC1_ENABLED = 0
 PARAMETER C_BUS2CORE_CLK_RATIO = 1
 PARAMETER C_TEMAC_TYPE = 3
 PARAMETER C_TEMAC0_PHYADDR = 0b00001
 PARAMETER HW_VER = 2.03.a
 PARAMETER C_TEMAC0_TXCSUM = 1
 PARAMETER C_TEMAC0_RXCSUM = 1
 PARAMETER C_TEMAC0_TXFIFO = 16384
 PARAMETER C_TEMAC0_RXFIFO = 32768
 PARAMETER C_BASEADDR = 0x89480000
 PARAMETER C_HIGHADDR = 0x894fffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE LLINK0 = Hard_Ethernet_MAC_LLINK0
 PORT TemacIntc0_Irpt = Hard_Ethernet_MAC_TemacIntc0_Irpt
 PORT TemacPhy_RST_n = fpga_0_Hard_Ethernet_MAC_TemacPhy_RST_n_pin
 PORT GTX_CLK_0 = clk_125_0000MHz
 PORT REFCLK = clk_200_0000MHzMMCM0
 PORT LlinkTemac0_CLK = clk_100_0000MHzMMCM0
 PORT MII_TX_CLK_0 = fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin
 PORT GMII_TXD_0 = fpga_0_Hard_Ethernet_MAC_GMII_TXD_0_pin
 PORT GMII_TX_EN_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_EN_0_pin
 PORT GMII_TX_ER_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_ER_0_pin
 PORT GMII_TX_CLK_0 = fpga_0_Hard_Ethernet_MAC_GMII_TX_CLK_0_pin
 PORT GMII_RXD_0 = fpga_0_Hard_Ethernet_MAC_GMII_RXD_0_pin
 PORT GMII_RX_DV_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_DV_0_pin
 PORT GMII_RX_ER_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_ER_0_pin
 PORT GMII_RX_CLK_0 = fpga_0_Hard_Ethernet_MAC_GMII_RX_CLK_0_pin
 PORT MDC_0 = fpga_0_Hard_Ethernet_MAC_MDC_0_pin
 PORT MDIO_0 = fpga_0_Hard_Ethernet_MAC_MDIO_0_pin
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_0
 PARAMETER C_COUNT_WIDTH = 32
 PARAMETER C_ONE_TIMER_ONLY = 0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BASEADDR = 0x83c00000
 PARAMETER C_HIGHADDR = 0x83c0ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER C_CLKIN_FREQ = 200000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_PHASE = 0
 PARAMETER C_CLKOUT0_GROUP = MMCM0
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT1_FREQ = 125000000
 PARAMETER C_CLKOUT1_PHASE = 0
 PARAMETER C_CLKOUT1_GROUP = NONE
 PARAMETER C_CLKOUT1_BUF = TRUE
 PARAMETER C_CLKOUT2_FREQ = 200000000
 PARAMETER C_CLKOUT2_PHASE = 0
 PARAMETER C_CLKOUT2_GROUP = MMCM0
 PARAMETER C_CLKOUT2_BUF = TRUE
 PARAMETER C_CLKOUT3_FREQ = 400000000
 PARAMETER C_CLKOUT3_PHASE = 0
 PARAMETER C_CLKOUT3_GROUP = MMCM0
 PARAMETER C_CLKOUT3_BUF = TRUE
 PARAMETER C_CLKOUT4_FREQ = 400000000
```

```
 PARAMETER C_CLKOUT4_PHASE = 0
 PARAMETER C_CLKOUT4_GROUP = MMCM0
 PARAMETER C_CLKOUT4_BUF = FALSE
 PARAMETER C_CLKOUT4_VARIABLE_PHASE = TRUE
 PARAMETER C_PSDONE_GROUP = MMCM0
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER HW_VER = 4.00.a
 PORT CLKIN = dcm_clk_s
 PORT CLKOUT0 = clk_100_0000MHzMMCM0
 PORT CLKOUT1 = clk_125_0000MHz
 PORT CLKOUT2 = clk_200_0000MHzMMCM0
 PORT CLKOUT3 = clk_400_0000MHzMMCM0
 PORT CLKOUT4 = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT PSCLK = clk_200_0000MHzMMCM0
 PORT PSEN = MPMC_DCM_PSEN
 PORT PSINCDEC = MPMC_DCM_PSINCDEC
 PORT PSDONE = MPMC_DCM_PSDONE
 PORT RST = sys_rst_s
 PORT LOCKED = Dcm_all_locked
END

BEGIN mdm
 PARAMETER INSTANCE = mdm_0
 PARAMETER C_MB_DBG_PORTS = 1
 PARAMETER C_USE_UART = 1
 PARAMETER C_UART_WIDTH = 8
 PARAMETER HW_VER = 1.00.g
 PARAMETER C_BASEADDR = 0x84400000
 PARAMETER C_HIGHADDR = 0x8440ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_mdm_bus
 PORT Debug_SYS_Rst = Debug_SYS_Rst
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER HW_VER = 2.00.a
 PORT Slowest_sync_clk = clk_100_0000MHzMMCM0
 PORT Ext_Reset_In = sys_rst_s
 PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
 PORT Dcm_locked = Dcm_all_locked
 PORT MB_Reset = mb_reset
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = sys_periph_reset
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT Intr = Hard_Ethernet_MAC_TemacIntc0_Irpt&xps_timer_0_Interrupt&
      DDR3_SDRAM_SDMA2_Rx_IntOut&DDR3_SDRAM_SDMA2_Tx_IntOut&
      fpga_0_Hard_Ethernet_MAC_PHY_MII_INT_pin
 PORT Irq = microblaze_0_Interrupt
END

BEGIN xps_sysace
 PARAMETER INSTANCE = SysACE_CompactFlash
 PARAMETER C_MEM_WIDTH = 8
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x83600000
 PARAMETER C_HIGHADDR = 0x8360ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT SysACE_MPA = fpga_0_SysACE_CompactFlash_SysACE_MPA_pin
 PORT SysACE_CLK = fpga_0_SysACE_CompactFlash_SysACE_CLK_pin
 PORT SysACE_MPIRQ = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin
 PORT SysACE_CEN = fpga_0_SysACE_CompactFlash_SysACE_CEN_pin
 PORT SysACE_OEN = fpga_0_SysACE_CompactFlash_SysACE_OEN_pin
 PORT SysACE_WEN = fpga_0_SysACE_CompactFlash_SysACE_WEN_pin
 PORT SysACE_MPD = fpga_0_SysACE_CompactFlash_SysACE_MPD_pin
END

BEGIN xps_hwicap
 PARAMETER INSTANCE = xps_hwicap_0
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_BASEADDR = 0x86800000
```

```
 PARAMETER C_HIGHADDR = 0x8680ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT ICAP_Clk = clk_100_0000MHzMMCM0
END

BEGIN dyn_interface
 PARAMETER INSTANCE = dyn_interface_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xcb000000
 PARAMETER C_HIGHADDR = 0xcb00ffff
 BUS_INTERFACE MPLB = mb_plb
 BUS_INTERFACE SPLB = mb_plb
END

BEGIN xps_mch_emc
 PARAMETER INSTANCE = FLASH
 PARAMETER C_NUM_BANKS_MEM = 1
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_WIDTH = 16
 PARAMETER C_MAX_MEM_WIDTH = 16
 PARAMETER C_INCLUDE_DATAWIDTH_MATCHING_0 = 1
 PARAMETER C_SYNCH_MEM_0 = 0
 PARAMETER C_TCEDV_PS_MEM_0 = 110000
 PARAMETER C_TAVDV_PS_MEM_0 = 110000
 PARAMETER C_THZCE_PS_MEM_0 = 35000
 PARAMETER C_TWC_PS_MEM_0 = 11000
 PARAMETER C_TWP_PS_MEM_0 = 70000
 PARAMETER C_TLZWE_PS_MEM_0 = 35000
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_MEM0_BASEADDR = 0x8c000000
 PARAMETER C_MEM0_HIGHADDR = 0x8dffffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RdClk = clk_100_0000MHzMMCM0
 PORT Mem_A = 0b0000000 & fpga_0_FLASH_Mem_A_pin_vslice_7_30_concat & 0b0
 PORT Mem_CEN = net_bsbassign0
 PORT Mem_OEN = fpga_0_FLASH_Mem_OEN_pin
 PORT Mem_WEN = fpga_0_FLASH_Mem_WEN_pin
 PORT Mem_DQ = fpga_0_FLASH_Mem_DQ_pin
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = FLASH_CE_inverter
 PARAMETER C_OPERATION = not
 PARAMETER C_SIZE = 1
 PARAMETER HW_VER = 1.00.a
 PORT Op1 = net_bsbassign0
 PORT Res = fpga_0_FLASH_CE_inverter_Res_pin
END
```

E Software Konfiguration des MicroBlaze Systems

Listing 16: *MSS-Datei zum Erstellen der Bibliotheken im SoC-Client*

```
0
 PARAMETER VERSION = 2.2.0


BEGIN OS
5 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 5.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
10 PARAMETER SYSTMR_SPEC = true
 PARAMETER SYSTMR_DEV = xps_timer_0
 PARAMETER SYSINTC_SPEC = xps_intc_0
 PARAMETER SCHED_TYPE = SCHED_PRIO
 PARAMETER PTHREAD_STACK_SIZE = 4096
15 PARAMETER ENHANCED_FEATURES = true
 PARAMETER CONFIG_KILL = true
 PARAMETER CONFIG_YIELD = true
 PARAMETER CONFIG_SEMA = true
 PARAMETER CONFIG_TIME = true
20 PARAMETER CONFIG_DEBUG_SUPPORT = true
 PARAMETER VERBOSE = true
 PARAMETER DEBUG_MON = true
 PARAMETER CONFIG_PTHREAD_MUTEX = true
 PARAMETER MAX_PTHREADS = 15
25 PARAMETER MAX_PTHREAD_MUTEX = 15
 PARAMETER MAX_PTHREAD_MUTEX_WAITQ = 15
 PARAMETER STATIC_PTHREAD_TABLE = ((main_thread,1))
END

30
BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.12.b
 PARAMETER HW_INSTANCE = microblaze_0
35 END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = mpmc
40 PARAMETER DRIVER_VER = 4.00.a
 PARAMETER HW_INSTANCE = DDR3_SDRAM
END

BEGIN DRIVER
45 PARAMETER DRIVER_NAME = lltemac
 PARAMETER DRIVER_VER = 3.00.a
 PARAMETER HW_INSTANCE = Hard_Ethernet_MAC
END

50 BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = RS232_Uart_1
END
55
BEGIN DRIVER
 PARAMETER DRIVER_NAME = sysace
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = SysACE_CompactFlash
60 END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 2.00.a
65 PARAMETER HW_INSTANCE = dlmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
70 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = ilmb_cntlr
END
```

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = mdm_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = hwicap
 PARAMETER DRIVER_VER = 4.00.a
 PARAMETER HW_INSTANCE = xps_hwicap_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = xps_intc_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = xps_timer_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = dyn_interface_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = emc
 PARAMETER DRIVER_VER = 3.00.a
 PARAMETER HW_INSTANCE = FLASH
END


BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = lwip130
 PARAMETER LIBRARY_VER = 2.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER API_MODE = SOCKET_API
 PARAMETER LWIP_DHCP = true
 PARAMETER DHCP_DOES_ARP_CHECK = true
END

BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = xilmfs
 PARAMETER LIBRARY_VER = 1.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
END
```

F Auszug aus dem PLB-Master Interface zum Lesen und Schreiben der Daten

Listing 17: *Zustandsautomat des PLB-Masters*

```
0   STATE_REG_PROCESS : process( Bus2IP_Clk ) is
    begin
      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
        if Bus2IP_Reset = '1' then
          state_reg <= IDLE;
5         read_counter_reg <= (others => '0');
          write_counter_reg <= (others => '0');
          started_reg <= '0';
        else
          state_reg <= state_sig;
10        read_counter_reg <= read_counter_sig;
          write_counter_reg <= write_counter_sig;
          started_reg <= started_sig;
        end if;
      end if;
15  end process;

    -- Das Signal Bus2IP_MstRd_src_rdy_n zeigt an, dass die Daten gültig sind
    data_in_fifo_write <= (not Bus2IP_MstRd_src_rdy_n) and  Bus2IP_Mst_Cmplt;

20
      READ_STATE_PROCESS : process( state_reg, slv_reg0, Bus2IP_Mst_CmdAck, Bus2IP_Mst_Cmplt,
                                     data_in_fifo_full_n, data_out_fifo_empty_n, write_counter_reg
                                     ,
                                     read_counter_reg, slv_reg4, slv_reg2, slv_reg1,
                                     slv_reg3, started_reg, write_reached_sig, read_reached_sig
25                                  ) is
    begin
    --Standard zuweisungen
    IP2Bus_MstRd_Req          <= '0';
    IP2Bus_MstWr_Req          <= '0';
30
    data_out_fifo_read <= '0';

    state_sig <= state_reg;
    IP2Bus_MstRd_Req <= '0';
35  req_rest <= '0';
    prm_reset_sig <= '0';

    read_counter_sig <= read_counter_reg;
    write_counter_sig<= write_counter_reg;
40  counter_seg <= '0';
    -- FSM
    started_sig <= started_reg;
      case state_reg is
      when IDLE =>
45      if(start_bit = '1') then
          req_rest <= '1';
          prm_reset_sig <= '1';
          -- Übernahme der Zählereingänge
          write_counter_sig <= unsigned(slv_reg3);
50        read_counter_sig <= unsigned(slv_reg1);
          started_sig <= '1';
          state_sig <= READ_REQ;
        end if;

55    when STARTED =>
        if(data_out_fifo_empty_n = '1'  and write_reached_sig = '0' ) then
          state_sig <= WRITE_REQ; -- Starten des Schreibens
        elsif(data_in_fifo_full_n = '1' and  read_reached_sig = '0' ) then
          state_sig <= READ_REQ;  -- Starten des Lesens
60      end if;

      when READ_REQ =>
        IP2Bus_MstRd_Req <= '1';
        if(Bus2IP_Mst_CmdAck = '1') then
65        state_sig <= WAIT_FOR_CMP;
        end if;

      when WAIT_FOR_CMP =>
        if(Bus2IP_Mst_Cmplt = '1') then
70        state_sig <= STARTED;
```

```
        read_counter_sig <= read_counter_reg + 4;
      end if;

    when WRITE_REQ =>
75    IP2Bus_MstWr_Req <= '1';
      counter_seg <= '1';
      if(Bus2IP_Mst_CmdAck = '1') then
        state_sig <= WAIT_FOR_WCMP;
      end if;
80
    when WAIT_FOR_WCMP =>
      counter_seg <= '1';
      if(Bus2IP_Mst_Cmplt = '1') then
        write_counter_sig <= write_counter_reg + 4;
85      data_out_fifo_read <= '1';
        state_sig <= STARTED;
      end if;

    when others => null;
90  end case;
end process;

-- Generien des Signals zum Anzeigen, dass die maximale Adresse erreicht wurde
reached_proc:process(write_counter_reg, slv_reg4,read_counter_reg , slv_reg2,started_reg) is
95  begin
    write_reached_sig <= '0';
    read_reached_sig <= '0';

    if(write_counter_reg = unsigned(slv_reg4) and started_reg = '1') then
100     write_reached_sig <= '1';
    end if;

    if(read_counter_reg = unsigned(slv_reg2) and started_reg = '1') then
      read_reached_sig <= '1';
105 end if;
end process;

-- Mux zur ausgabe der Adresse
IP2Bus_Mst_Addr <= std_logic_vector(read_counter_reg)  when counter_seg = '0' else
110                std_logic_vector(write_counter_reg) when counter_seg = '1' else
                   (others => '0');
```

G DVD mit Soft- und Hardware Projekten

1. SoC-Client:

   a) EDK-Projekt:
   ../Hardware/EDK/

   b) DYN_Interface:
   ../Hardware/DYN/

   c) PRM-Projekte:
   ../Hardware/PRM/

   d) PlanAhead Projekt:
   ../Hardware/PlanAhead/

2. Broker:
   ../Software/Broker/

3. Beispiel-Projekt:

   a) Java:
   ../Software/Beispiel_Projekt/

   b) Matlab:
   ../Software/Matlab/

4. Arbeit:
   ../Master_Arbeit/ma.pdf

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. November 2011

Ort, Datum

Unterschrift