Thread basierte partielle Rekonfiguration von SoC Systemen

Frank Opitz

Ausarbeitung

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Frank Opitz

Thread basierte partielle Rekonfiguration von SoC Systemen

Ausarbeitung eingereicht im Rahmen der Veranstaltung Anwendung 1
im Masterstudiengang Informatik
im Studiendepartment Informatik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor : Dr.-Ing. Bernd Schwarz

Gutachter : Prof. Dr.rer.nat. Kai von Luck
Gutachter : Prof. Dr.rer.nat. Gunter Klemke

Inhaltsverzeichnis

1 Einleitung

Der Einsatz der partiellen Rekonfiguration ermöglicht es die Ressourcen eines FPGAs mehrfach während des Betriebes zu verwenden. Im Gegensatz zu herkömmlichen Methoden kann der FPGA auf Änderungen des Umfeldes reagieren ohne entsprechende Module dauerhaft geladen zu haben. Dies ist unter der Voraussetzung möglich, dass nicht alle "Intellectual Property" (IP) -Blöcke gleichzeitig benötigt werden. In diesem Fall kann der FPGA von außerhalb oder innerhalb rekonfiguriert und somit oftmals der Einsatz von mehreren FPGAs vermieden werden.

Eingesetzt wird diese Technologie unter anderem in kryptografischen Systemen. Die Ausführung der Verschlüsselungsalgorithmen in Software führt zu einer hohen Auslastung der CPU, in Hardware werden im Gegenzug viele Ressourcen verbraucht. Ismaili und Moussa beschreiben ein System, welches die kryptografischen Module selbständig rekonfiguriert [Ismaili und Moussa (2009)]. Es stehen Module für unterschiedliche Längen der Schlüssel bereit, welche, über einen FPGA internen MicroBlaze Softcore-Prozessor gesteuert, rekonfiguriert werden. Die Parameter sowie die zu entschlüsselnden Datensätze werden den rekonfigurierten Modulen vom MicroBlaze zur Verfügung gestellt. Dieser ist statisch und wird beim initialen Konfigurieren des FPGAs geladen. Im Gegensatz zur herkömmlichen Implementation wird hier die Summe der Anzahl der Slices der kleinsten Module gespart, da die zu rekonfigurierende Fläche größer oder gleich des Moduls mit dem höchsten Verbrauch an Slices ist.

Ein weiteres Beispiel der Verwendung der partiellen Rekonfiguration ist der Aufbau eines Schwarms von unbemannten Flugzeugen (UAV). Dieser Schwarm verteilt die entsprechenden IPs innerhalb eines FPGAs oder über ein Netzwerk an die anderen Teilnehmer [Kearney und Jasiunas (2006)]. Unterschieden wird hier in zwei Typischen Scenarien.

- Single Mission:
  In diesem Scenarion hat jeder Schwarm Teilnehmer eine spezielle Aufgabe. Ist der Treibstoff eines Teilenehmers aufgebraucht, so kehren alle zur Basis zurück und nehmen neuen Treibstoff auf.
- Continuous Mission:
  Im Gegensatz zur Single Mission, stehen statt N Teilnehmern $N + M$ zur Verfügung. N ist hierbei die Anzahl der benötigten und M die darüber hinaus zur Verfügung stehenden Teilnehmern. Ist der Treibstoff eines Teilnehmers aufgebraucht, so übernimmt ein anderer seinen Platz und er kehrt alleine zurück zur Aufnahme von neuem Treibstoff.

Im zweiten Scenario werden die benötigten Hardware-Beschleuniger innerhalb des Schwarms unter den entsprechenden Teilnehmern verteilt.

Im Rahmen des Masterstudiums ist geplant diese Technologie, aufbauend auf einer Masterarbeit, auf einem "System on Chip" (SoC) basierten autonomen Fahrzeug einzusetzen [Mamegani (2010)]. Die hierfür verwendete Technologie sowie das Testsystem und die nächsten Schritte beschreibt diese Ausarbeitung. Die Steuerung der Rekonfiguration sowie des Fahrzeuges erfolgt über ein Real "Time Operating System" (RTOS), welches auf MicroBlaze Instanziierungen auf den FPGAs läuft. Die Auswahl dieses RTOS erfolgt ebenfalls in dieser Ausarbeitung.

Diese Ausarbeitung beschreibt in Kapitel 1.1 den bisherigen Aufbau des SoC Fahrzeuges. Kapitel 2 beschäftigt sich mit den Vor- und Nachteilen verschiedener Re- Konfigurations-Methoden und Anforderungen an die Rekonfigurationsdesigns bei Xilinx FPGAs. Die Auswahl des RTOS

wird in Kapitel 3 beschrieben. Eine Beschreibung des Testsystems sowie die nächsten Arbeitsschritte erfolgt in Kapitel 4. Anschließend folgt eine Zusammenfassung der Ausarbeitung und ein Ausblick auf weiterführenden Themenstellungen.

1.1 Das SoC Carolo-Cup Fahrzeug

Der Carolo Cup ist ein von der TU Braunschweig ausgeschriebener Wettbewerb, welcher sich mit dem Thema autonome Fahrzeuge beschäftigt [Braunschweig (2010)]. Die HAW-Hamburg stellt bisher ein Software basiertes Fahrzeug zur Teilnahme an diesem Wettbewerb. Dieses Fahrzeug enthält seine Umbegunbsdaten beispielsweise über eine Front-Kamera und Infrarot-Sensoren. Die Verarbeitung dieser Daten erfolgt über eine Acer Aspire One mit einem Atom Prozessor und drei Atmel Prozessoren. Parallel zu diesem Fahrzeug, erfolgt der Aufbau eines Fahrzeuges, welches mit Hilfe von 2 Spartan 3 FPGAs gesteuert wird. Der Vorteil dieses Aufbaufs ist, dass zum einen Gewicht eingespart wird zum anderen die Leistungsaufnahme verringert wird. Abbildung 1 zeigt den Aufbau des SoC Fahrzeuges.

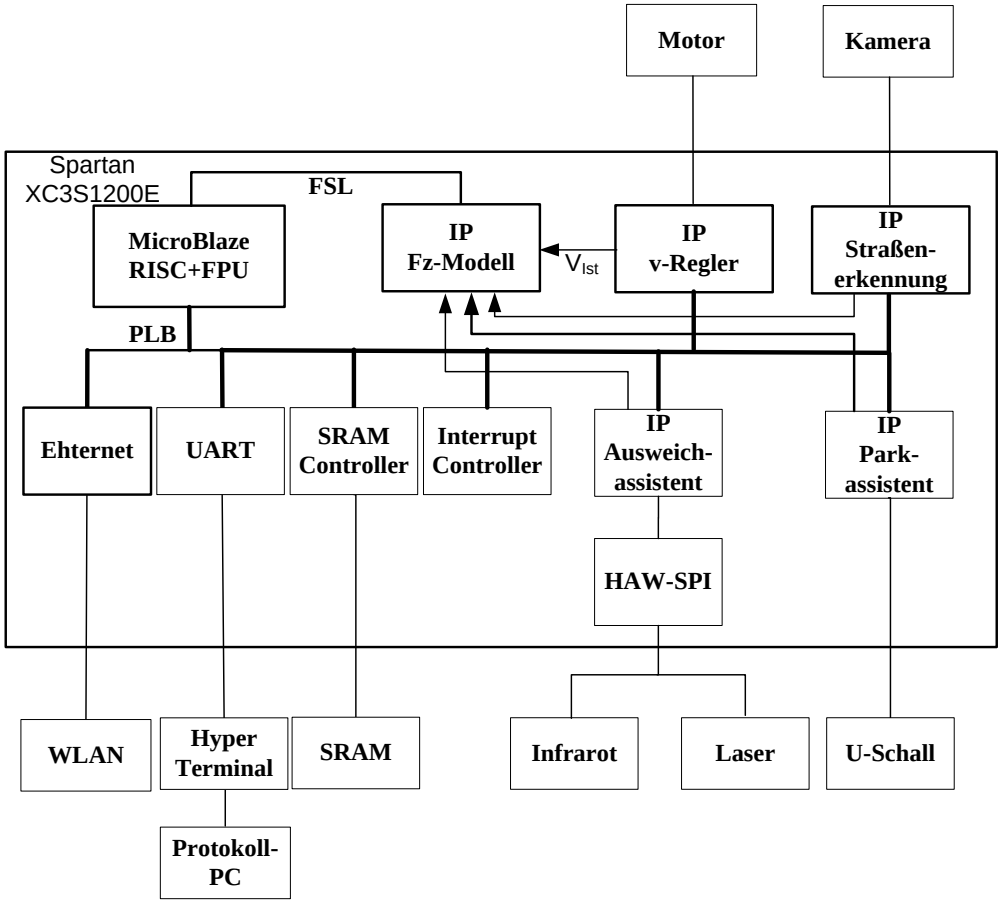

Abb. 1: *FPGA-basierte SOC-Steuerplattform für ein autonomes Fahrzeug sowie Bildverarbeitungs- und Sensorauswertungseinheiten.*

Die Module haben folgende Aufgaben:

- MicroBlaze
  Der MicroBlaze RISC Prozessor ist ein 32 Bit Softcore-Prozessor welcher Interrupt gesteuert die Daten der Module verarbeitet [Xilinx (2009b)]. In einem der nächsten Schritte

soll ein RTOS die Verarbeitung der Daten, auf dem MicroBlaze und das Starten der Rekonfiguration beim Kontext wechsel übernehmen.

- FZ-Modell
  Das Fahrzeug-Modell gibt den Modulen den aktuellen Systemzustand wieder. Die Steuereinheiten, welche die Aktoren wie Motor und Lenk-Servo steuern, geben ihre aktuellen Parameter an dieses Modul weiter [Mellert (2010)].

- V-Regler
  Der V-Regler ist für die Geschwindigkeitsregelung des Fahrzeuges zuständig. Die vorgegebene Geschwindkeit wird mit den Daten eine Inkrementalgebers verglichen und die entsprechende pulsweitenmodulierte Spannung an den Motor ausgegeben. [Bordasch (2009)].

- Straßenerkennung
  Die Erkennung der Straße erfolgt anhand von Bildverarbeitungsalgorithmen. Zum Einsatz kommen hier beispielsweise der Sobeloperator und die Houghtransformation [Mellert (2010)].

- Ausweich- und Parkassisten
  Die beiden Assistenten übernehmen jeweils die Aufgabe zur Erkennung eines Hindernisses auf der Straße und das anschließende Ausweichen oder das Abmessen einer Parklücke und darauffolgendes Einparken.

- Ethernet, UART, SRAM-Controller
  Diese Module dienen der Kommunikation zwischen dem Fahrzeug und einem Protokoll-System während der Testphase, sowie der Ansteuerung eines Speichers.

Aufgrund der aktuellen Aufgabenstellung des Carolo Cups, werden nicht alle Module gleichzeitig benötigt, da das Einparken und das Fahren auf dem Rundkurs nicht gleichzeitig stattfindet [Braunschweig (2010)]. Hier wird im späteren Verlauf des Projekts die dynamische Rekonfiguration eingesetzt. Die aktuelle Planung sieht vor, das der Park- und der Ausweichassistent geladen werden, wenn die entsprechende Aufgabe zu erledigen ist. Die anderen Module werden in der Standardkonfiguration dauerhaft im FPGA gehalten.

2 Konfiguration von FPGAs

Die Konfiguration von FPGAs wird in zwei Kategorien unterheilt, die volle und die partielle Konfiguration. Die volle Konfiguration wird zumeist beim Starten des Systems ausgeführt und konfiguriert den FPGA bis zu seinem Abschalten. Diese “typische Konfiguration“ wird in den meisten FPGA Systemen angewandt. Es wird hier die komplette Konfiguration beim Start in den FPGA geschrieben, so dass die Größe des FPGAs die Anzahl und Komplexität der Module bestimmt [Xilinx (2008b)].

Überschreiten die Module die vorhandenen Ressourcen, so bleiben dem Entwickler zwei Möglichkeiten. Die erste ist das System mit einem oder mehreren FPGAs zu erweitern oder zu überprüfen, ob einige Module zu unterschiedlichen Zeiten laufen. Ist dies der Fall so kann der FPGA während des Betriebes der Plattform mit einer zweiten Konfiguration überschrieben werden. Bei dieser Art der Rekonfiguration gehen die Systemzustände verloren und wichtige Daten sind außerhalb des FPGAs zu speichern [Kao (2005)] .

Die Re-Konfiguration des Systems benötigt mehrere Millisekunden, in denen der FPGA reaktionsunfähig ist. Diese Zeit kann verkürzt werden indem nicht der komplette FPGA rekonfiguriert wird. Diese Rekonfiguration, welche auch partielle Rekonfiguration genannt wird, kann in eine “online“ und “offline“ Variante unterteilt. Die offline Variante verändert die Konfiguration von Teilblöcken, während der FPGA wieder reaktionsunfähig ist, verkürzt aber die Zeit der Rekonfiguration, da nur Teilkomponenten neu beschrieben werden. In sicherheitskritischen Systemen, die keinen Ausfall des Systems in Kauf nehmen können, wird die “Dynamische Partielle Rekonfiguration“ angewandt, welche die online Variante der partiellen Rekonfiguration darstellt. Diese Technik wird bisher nur von Xilinx unterstützt und erfordert das Einsetzen von speziellen Bus Makros bei der Kommunikation zwischen den Rekonfigurationsmodulen und anderen. Weiterhin sind sogenannte “partially reconfigurable regions“ (PRR) durch AREA_GROUP Constraints festzulegen. Diese PRR stecken die Bereiche ab, die rekonfiguriert werden können [Xilinx (2008a)]. Abbildung 2 zeigt den Aufbau eines Systems mit einer PRR in die unterschiedliche “partial reconfigurable modules“ (PRM) geladen werden. Diese können nun zur Laufzeit, in die PRR geschrieben werden, wobei das Starten der Rekonfiguration FPGA intern oder von außerhalb erfolgen kann. Der Aufbau eines so selbstrekonfigurierenden Systems ist in Kapitel 4 dargestellt.

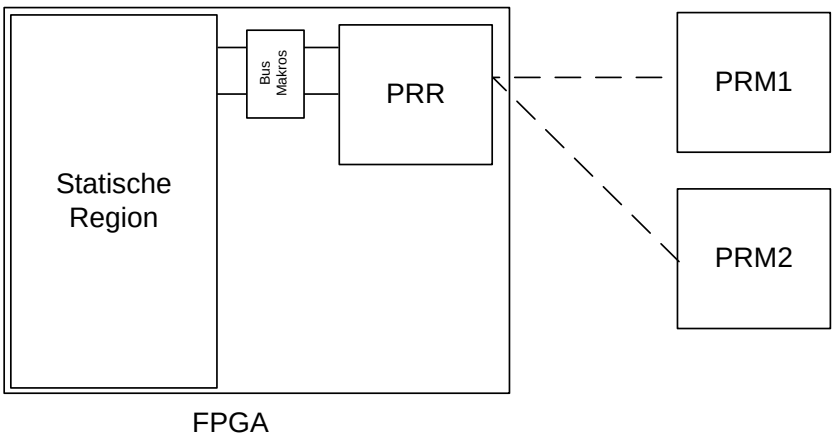

Abb. 2: *System mit Statischem- und Rekonfigurationsblock*

2.1 Bus Makros

Die Kommunikation zwischen Rekonfigurationsbereichen zu statischen Bereichen oder anderen Rekonfigurationsbereichen erfolgt immer über statische Routen. Damit diese Anforderung erfüllt wird, werden auf Xilinx FPGAs sogenannte Bus Makros verwendet. Diese setzen die Schnittstellen fest und regeln die Kommunikation zwischen den Modulen [Xilinx (2009a)].

Das Festsetzen der Kommunikationspunkten ist erforderlich, da eine Abweichung eine nicht funktionierende Kommunikation zur Folge hätte. Die Bus Markos sind dazu schon vorgeroutet, so dass bei jeder Kompilation die Schnittstellen an den gleichen Punkten liegen. In der aktuellen Implementation der Bus Makros werden pro instattiiertem Bus Marko 4 Signale übergeben. In Richtung des dynamischen Moduls werden Bus Makros instatiiert, welche die Signale vom statischen Modul dauehaft übergeben, da hier die Übernahme der Signale über Enables gesteuert wird (vgl. Abbildung 3 Data_out1/2). Datenflüsse, die aus der PRR kommen, werden vom statischen Modul während der Rekonfiguration auf ’0’ gesetzt (vgl. Abbildung 3 Datan_in 1/2). Die Steuerung der Kommunikation ist hier erforderlich, da bei einer Rekonfiguration teilweise unbestimmte Zustände auftreten können, welche sich auf die aktiven Komponenten auswirken [Xilinx (2009a)].

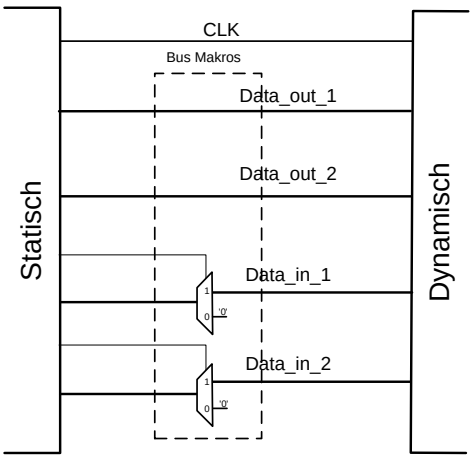

Abb. 3: *Kommunikation zwischen statischem und dynamischen Modul über Bus Makros*

3 Auswahl des RTOS

Die Steuerung der Rekonfiguration sowie der Bearbeitung der SoC-Plattform Daten erfolgt mit der Hilfe eines “Real Time Operating Systems“ (RTOS). Dieses übernimmt zum einen das Scheduling der Tasks und zum anderen die Steuerung der Rekonfiguration. Das Starten der Rekonfiguration kann an zwei Punkten erfolgen. Der erste Punkt liegt vor dem Ausführen des Tasks, so dass die Rekonfiguration abgeschlossen ist, wenn der Task gestartet wird. Der zweite Punkt ist beim ersten Zugriff auf das Modul, so dass die Bearbeitung der Aufgabe für die Zeit der Rekonfiguration unterbrochen wird. Die erste Variante des Zeitpunktes zur Rekonfiguration erfordert die Quelloffenheit des RTOS, da Komponenten des RTOS, wie z.B. der Scheduler, verändert werden. Zu dieser Anforderung an das RTOS kommen folgende hinzu:

- Geringer Speicherbedarf
  Das RTOS wird auf einem Spartan 3 FPGA laufen. Dieser FPGA besitzt ein Intern Block-Ram, welches eine Tiefe von insgesamt 64 kByte hat. Dieses Block-Ram wird auch von den anderen Modulen verwendet, so dass für den internen Speicher ca. 31 kByte zur Verfügung stehen, welche vom RTOS genutzt werden können [Xilinx (2009c)].

- Multiprozessorfähigkeit
  Im weiteren Verlauf wird die SoC-Plattform erweitert, so dass eine Multiprozessor-Plattform entsteht [Bordasch (2009)]. Das RTOS ist im Hinblick auf diese Erweiterung zu wählen.

- Vorhersagbares Verhalten
  Das Fahrzeug wird im Hard-Real-Time Kontext betrieben, da eine zu späte Bearbeitung von Hindernissen auf der Fahrbahn zu einem Unfall führen kann. Das Bearbeiten von Interrupts und das Scheduling der Task muss also in vorher definierten Zeiträumen ablaufen.

Die Wahl des RTOS für das CaroloCup Fahrzeug stützt sich auf eine Studie der Canadian Space Agency. Diese Studie untersucht den Aufbau der Systeme und bewertet sie je nach Aufbau (vgl. Tabelle 1)

Tabelle 1: *Ausschnitt der Kriterien der Canadian Space Agency für RTOS. W_i stellt die Gewichtung der Kategorie dar und W_j die Gewichtung innerhalb der Kategorie [Melanson und Tafazoli (2003)].*

Kategorie	W_i	Kriterium	W_j	$W_i * W_j$
Kernel	13%	Architektur	35%	5%
		Multi-processor support	25%	3%
Scheduling	20%	Algorithmen	40%	8%
		Priorität	20%	4%
Process/Thread/ Task Modell	12%	Anzahl der Prioritäten Level	26%	3.12%
		Max. Anzahl der Tasks	18%	2.16%

Diese Studie wurde anfänglich mit 48 RTOS durchgeführt. Vor der eigentlichen Bewertung wurden 28 Systeme aus der Studie gestrichen, da diese z.B. von dem gleichen Hersteller produziert werden oder nur für bestimmte Prozessoren gedacht waren. Die Anwendung der Kriterien auf die einzelnen RTOS ergab eine Rangliste der verbleibenden 20 Systeme [Melanson und Tafazoli (2003)]. Diese Rangliste wurde mit den Anforderungen des Fahrzeugs verglichen. Dieser

Vergleich ergab, dass das uC/OS der Firma Micrium die Anforderungen des Fahrzeuges erfüllt. Ein weiter Punkt der für dieses RTOS spricht, ist dass bereits eine Implementierung für den MicroBlaze existiert [Micrium (2009a)].

3.1 uC/OS2

Das uC/OS2, was für "Micro-Controller Operating System" steht, wurde im Jahr 2000 zertifiziert die Anforderungen des RTCA DO-178B zu erfüllen. Dieser Standard ist für den Gebrauch von Software im Luftfahrt Bereich und legt besonderen Wert auf die Robustheit und Sicherheit des Codes [Labrosse (2002)]. Der Code des uC/OS ist größtenteils in ANSI-C geschrieben. Die Verwendung von Assembler-Aufrufen ist so reduziert worden, dass das System durch das kleine Änderungen im Assembler Code auf die verschiedenen Prozessoren portiert werden kann. Die Ausführung des uC/OS erfordert auf den Prozessoren einen Stack-Pointer, das "Push" und "Pop" der CPU-Register auf den Stack und das Abschalten der Interrupts per C oder Assembler Anweisung. Diese Anforderungen werden von den meisten Prozessoren erfüllt, so dass es viele Portierungen gibt. Portiert wurde es unter anderem für den Xilinx MicroBlaze oder den Altera Nios [Micrium (2009b), Micrium (2009a)].

Das System ist so aufgebaut, dass es komplett im ROM gehalten werden kann. Die Größe des benötigten Speichers variiert zwischen 6 kByte und 24 kByte. Das Scheduling der Tasks erfolgt preemptiv und ist in der uC/OS2 Version auf 256 begrenzt. Jeder Task hat hier eine im System einzigartige Priorität, wodurch immer der Task mit der höchsten Priorität läuft, der lauffähig ist, und es keine Round-Robin Verteilung der Laufzeit gibt [Micrium (2009b), Micrium (2009a)].

4 Das PR-Testsystem mit Petalinux

Die Rekonfiguration der PRR erfolgt von innerhalb des FPGA oder über ein externes Modul. Je nach Methode sind unterschiedliche Erweiterungen des Systems erforderlich.

- Externe Rekonfiguration:
  Die Rekonfiguration erfolgt hier z.B. über eine JTAG Schnittstelle und wird über einen Mikrocontroller oder PC gesteuert. So sind zusätzlich zu den vorhandenen I/O-Pins, z.B. für einen Speicher, weitere Anschlüsse zur Verfügung zu stellen [Mamegani (2010)].
- Interne Rekonfiguration:
  Die interne Rekonfiguration erfordert das Ansprechen der "Internen Rekonfigurationsschnittstellen" (ICAP) des FPGA. Xilinx stellt dazu ein IP zur verfügung welches diese Hardware Schnittstellen anspricht wobei die PRM dabei aus einem Speicher oder über verschiedenen Busse geladen werden können [Xilinx (2009d)]. Gesteuert wird die Rekonfiguration von Software oder über eine "Finite State Machine" (FSM).

Abbildung 4 zeigt das Testsystem zur Selbstrekonfiguration, welches zur Zeit an der HAW zur Rekonfiguration eingesetzt wird. Unterteilt ist dies System in den statischen Part mit einem MicroBlaze sowie dem ICAP-IP und dem dynamischen Part mit einer PRR. Die Kommunikation zwischen den Bereichen erfolgt durch ein Interface, welches auf der einen Seite die Anbindung an den System-Bus (PLB) und auf der anderen Seite die Kontrolle der Bus Makros erledigt. Auf dem MicroBlaze läuft das Petalinux als OS welches die Rekonfiguration der PRR steuert.

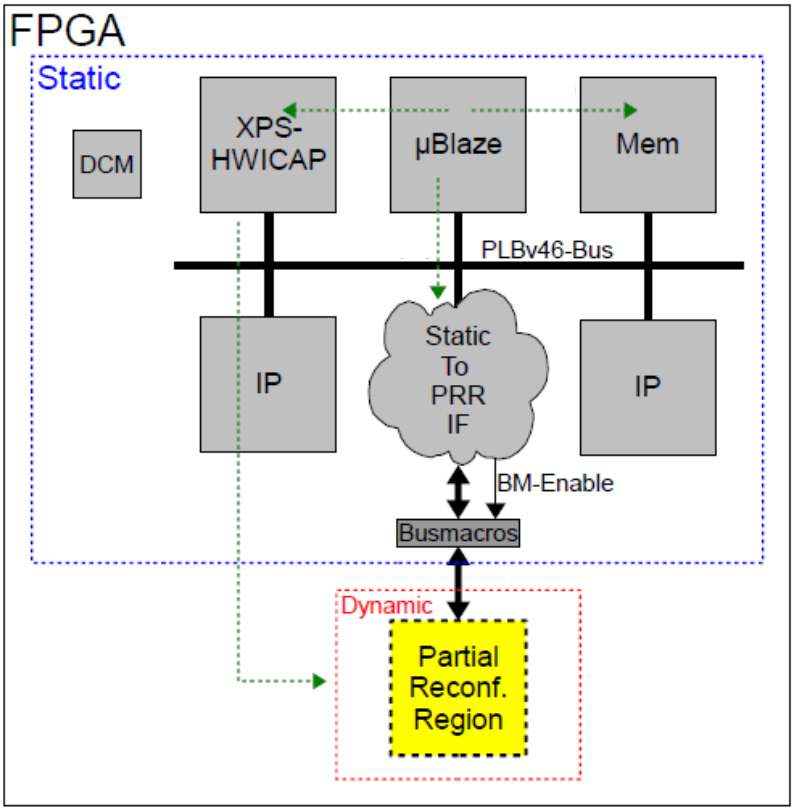

Abb. 4: *Modularer Aufbau des PR-Testsystems mit Petalinux [Mamegani (2010)]*

4.1 Die nächsten Arbeitsschritte

Aufbauend auf der Testumgebung sind folgenden Arbeitsschritte durchzuführen:

- Timinganalyse:
  Das Zeitverhalten der Rekonfiguration ist zu untersuchen. Zum einen ist das verhalten des bisher verwenden Operating System (OS) zu ermitteln. Zum anderen ist zu untersuchen wie sich die Rekonfiguration bei Module unterschiedlicher Größe verhält.
- Einbinden des RTOS:
  In dem existierenden PR-Testsystem wird Petalinux zur System Steuerung verwendet [Mamegani (2010)]. Dieses Linux OS wird durch das uC/OS RTOS ersetzt (vgl. Kapitel 3).

Abschließend ist die Rekonfigrationstechnologie auf der SoC-Plattform zu implementieren. Hierzu ist die Unterteilung des vorhandenen Systems in statische und rekonfigurierbare Module zu unterteilen und die Kommunikation anzupassen.

5 Zusammenfassung und Ausblick

Diese Ausarbeitung beschreibt die verschiedenen Varianten der Re-/Konfiguration von FPGA Systemen und die Steuerung dieser mit der Hilfe eines RTOS im Kontext eines autonomen Fahrzeuges. Die partielle Rekonfiguration bietet die Möglichkeit den FPGA während des Betriebes teilweise neu zu beschreiben. Bei dieser speziellen Rekonfigurationsmethode laufen die Berechnungen der anderen Module weiter und das System bleibt reaktionsfähig. Dies ist bei den üblicherweise verwendenten Rekonfigrationsmethoden nicht der Fall. Diese unterbrechen die Berechnungen des FPGAs und versetzen ihn in den Reset-Zustand. Bei der partiellen Rekonfiguration erfolgt die Kommunikation zwischen den PRR und den statischen Regionen des Systems über Bus-Markos. Zudem ist die zu rekonfigurierende Fläche über Constraints festzulegen.

Die Steuerung des Fahrzeuges erfolgt anhand eines RTOS. Die Anforderungen an dieses System werden vor allem durch den FPGA bestimmt, sowie durch das Fahrzeug. Diese sind z.B. eine maximale Größe von 31 kByte sowie ein vorhersagbares Verhalten im Bereich des Schedulings. Die Auswahl des Systems erfolgte mit Hilfe einer Studie der Canadian Space Agency. Das Ergebnis dieser Studie wurde mit den Anforderungen des Fahrzeuges verknüpft. Das zum jetzigen Zeitpunkt passende RTOS, ist das uC/OS 2. Dieses System erfüllt die Anforderungen des Fahrzeuges und wurde schon auf einem MicroBlaze portiert.

Die Analyse und Implementation erfolgt auf einem existierendem Testsystem. Die nächsten Arbeitsschritte bestehen aus der Analyse der Rekonfigurationszeiten im Verhältnis zum Kontextwechsel des OS sowie in Abhängigkeit der zu rekonfigurierenden Module. Anschließend ist das bisherige OS durch das gewählte RTOS zu ersetzen und das System auf der SoC-Plattform zu implementieren.

5.1 Weiterführende Projekte

Aufbauend auf den vorgestellten Techniken stehen im weiteren Verlauf des Studiengangs nicht nur Themen für das SoC Fahrzeug zur Verfügung. Denkbar sind hier, z.B. die Verteilung von IPs auf einem Multi-Core System oder die Verteilte Berechnung ähnlich des SETI@Home Netzwerkes.

Das SETI@Home Netzwerk, ist ein Verbund privat und gewerblich genutzter Computer, welche sich an der Suche nach außerirdischer Intelligenz des “Search for extraterrestrial intelligence“ (SETI) Projekts beteiligen. Die Teilnehmer melden sich bei den SETI@Home Servern an und stellen freie Rechner-Kapazitäten zur Verfügung, so dass die Untersuchung der Signale in einem großen Verbund stattfindet, aber keine Anschaffung vieler Computer und deren Bündelung an einem Bestimmten Ort notwendig ist (vgl. Abbildung 5) [Korpela u. a. (2001)]. Bei der Berechnung aufwendiger Algorithmen in Hardware, wie z.B. dem Untersuchen von Verschlüsselungsalgorithmen, werden große Netzwerke von FPGAs verwendet [PicoComputing (2010)]. Die Netzwerke werden innerhalb eines geschlossenen Systems aufgebaut und sind meist schwer zu erweitern. Mit der Verwendung der partiellen Rekonfigurationstechnologie ist es möglich die Berechnungen weltweit so zu verteilen, das ungenutzte FPGA-Ressourcen sich an der Lösung beteiligen. In diesem Fall würde auf den einzelnen FPGA ein Standardsystem laufen und die IPs bzw. Parameter würden an diese verteilt werden. Auf Basis dieses Netzwerkes wäre es nicht mehr notwendig eine große Anzahl von FPGAs an einem Ort zu koppeln, was das Berechnen solcher Algorithmen preiswert im Bezug auf Anschaffungskosten und Unterhalt macht.

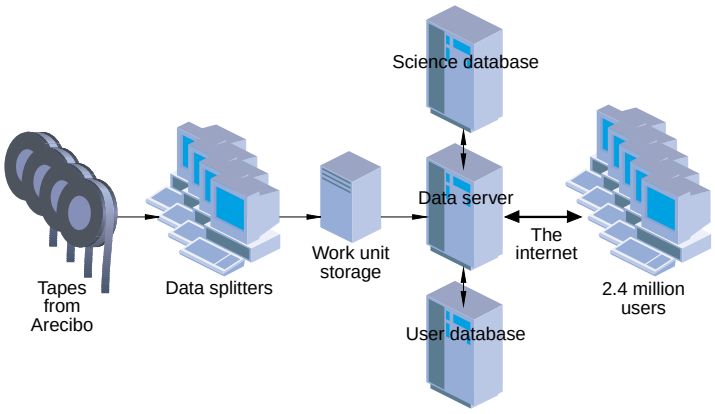

Abb. 5: *Topologie des SETI@Home Netzwerkes [Korpela u. a. (2001)]*

Software, welche mit spezialisierten Hardware-Beschleunigern arbeitet, ist auf bisherigen Systemen auf einen Core angewiesen. Bei der Verwendung der partiellen Rekonfigurationstechnologie besteht auch hier die Möglichkeit die Software auf die anderen Cores zu verteilen, ohne eine Instanz des Hardware Beschleunigers permanent geladen zu haben. Eine Anpassung des Schedulings hätte zum einen die aktuellen Belegungen der PRRs zu beachten, zum anderen die Verteilung der IPs zu organisieren. Weiterhin ist im Bereich der Multi-Core Systeme denkbar ein System zu Implementieren, welches die Anzahl der Cores selbst reguliert. Diese Regulierung würde anhand der Auslastung der bereits instanziierten Cores erfolgen.

Literatur

[Bordasch 2009] BORDASCH, Heiko: *VHDL-Modellierung einer Geschwindigkeitsregelung für ein autonomes Fahrzeug implementiert auf einer SoC - Plattform*, HAW-Hamburg, Diplomarbeit, 2009

[Braunschweig 2010] BRAUNSCHWEIG, TU: *Carolo-Cup Regelwerk 2010*. 2010

[Ismaili und Moussa 2009] ISMAILI, Zine El Abidine A. ; MOUSSA, Ahmed: Self-Partial and Dynamic Reconfiguration Implementation for AES using FPGA. In: *IJCSI International Journal of Computer Science Issues* 2 (2009)

[Kao 2005] KAO, Cindy: Benefits of Partial Reconfiguration. In: *Xcell Journal* (2005)

[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)

[Korpela u. a. 2001] KORPELA, Eric ; WERTHIMER, Dan ; ANDERSON, David ; COBB, Jeff ; LEBOFSKY, Matt: *SETIatHOME MASSIVELY DISTRIBUTED COMPUTING FOR SETI*. 2001. – Verfügbar Online unter `http://setiathome.berkeley.edu/sah_papers/CISE.pdf`; 01.02.10

[Labrosse 2002] LABROSSE, Jean J.: *MicroC/OS-II The Real-Time Kernel*. Bd. 2. Auflage. CMPBooks, 2002

[Mamegani 2010] MAMEGANI, Armin J.: *Erprobung und Evaluierung von Methoden zur partiellen und dynamischen Rekonfguration eines SoC-FPGAs*, HAW-Hamburg, Diplomarbeit, 2010

[Melanson und Tafazoli 2003] MELANSON, Philip ; TAFAZOLI, Siamak: *A Selection Methodology for the RTOS Market*. 2003

[Mellert 2010] MELLERT, Dennis: *Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx System Generator für eine SoC-Plattform*, HAW-Hamburg, Diplomarbeit, 2010

[Micrium 2009a] MICRIUM: *uC/OS-II and the Xilinx MicroBlaze Processor*, 2009

[Micrium 2009b] MICRIUM: *uC/OS-II Real-Time Operating System*, 2009

[PicoComputing 2010] PICOCOMPUTING: *FPGA Cluster Demonstrates Massively Parallel, Hardware-Accelerated DES Cracking*. 2010. – Verfügbar Online unter `http://www.picocomputing.com/pdf/PR_Pico_DES_BH_Jan_29_2010.pdf`; 01.02.10

[Xilinx 2008a] XILINX: *Early Access Partial Reconfiguration User Guide*. v. 1.2, 2008

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

[Xilinx 2009a] XILINX: *Development System Reference Guide*, 2009. – Verfügbar Online unter `http://www.xilinx.com/itp/xilinx8/books/docs/dev/dev.pdf`; 11.02.10

[Xilinx 2009b] XILINX: *MicroBlaze Processor Reference Guide*, 2009

[Xilinx 2009c] XILINX: *Spartan-3 FPGA Family Data Sheet*, 2009

[Xilinx 2009d] XILINX: *XPS HWICAP*. v.3.0, 2009. – Verfügbar Online unter `http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf`; 11.02.10

Abbildungsverzeichnis