Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung AW1

Felix Kolbe

System-on-a-Chip Konzepte für die Telemetrie- und Steuerungskomponenten in einem Formula Student Rennwagen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Die Entwicklung von eingebetteten Systemen unterliegt umfangreichen Anforderungen, wie geringer Stückpreis, geringer Energieverbrauch, hohe Rechenleistung und Einhaltung von Zeitgrenzen. Um diese häufig gegenläufigen Anforderungen erfüllen zu können, sind abgestimmte und kombinierte HW/SW-Lösungen erforderlich (vgl. Wolf, 2007, S. 383).

Eine reine Software-Lösung setzt voraus, dass die Rechengeschwindigkeit mit der Taktfrequenz soweit erhöht wird, dass alle Auswertungen, Berechnungen und Steuerungsaufgaben z. B. in den geforderten Messzyklen durchgeführt werden können. Eine Steigerung der Taktfrequenz ist jedoch nicht immer möglich und bewirkt einen höheren Energieverbrauch.

Bei einer Hardware-Lösung wird die Funktionalität direkt in reprogrammierbarer Hardware implementiert, etwa in einem Field Programmable Gate Array (FPGA). Dieses kann kombinatorisch oder getaktet agieren. Kombinatorische Logikkreise bilden die Eingänge innerhalb kürzester Zeit auf die Ausgänge ab und sind daher für arithmetrisch-logische Blöcke geeignet. Getaktete Gatter reagieren nur mit dem Taktsignal und sind für zeitabhängige Funktionen und Steuerungen wie etwa Zustandsautomaten geeignet. Ein solcher Hardware-Teil wird auch als Beschleuniger (engl. accelerator) bezeichnet, da dieser durch parallele Rechenblöcke spezielle Operationen schneller ausführen kann, als der Prozessor, dessen ALU auf einen Rechenschritt pro Takt und simple Operationen beschränkt ist (vgl. Ashenden, 2008, S. 393).

Durch eine kombinierte Lösung aus Software und Hardware lassen sich die Vorzüge beider Lösungen übernehmen und die großen Nachteile übergehen (vgl. Vahid und Stitt, 2008, S. 540). In der Partitionierung entscheidet sich, welche Teile der Funktionalität in SW bzw. in HW implementiert werden (vgl. Abb. 1.1 auf der nächsten Seite). Die Partitionierung kann zu einem überaus komplexen Problem werden, wenn der Funktionsumfang eine unüberschaubare Anzahl von Aufteilungsmöglichkeiten bewirkt. Eine solche Kombination wird mit HW/SW-Codesign bezeichnet. Die Integration von Prozessor und diversen anderen Komponenten auf einem einzigen Halbleiterchip wird System-on-a-Chip (SoC) genannt.

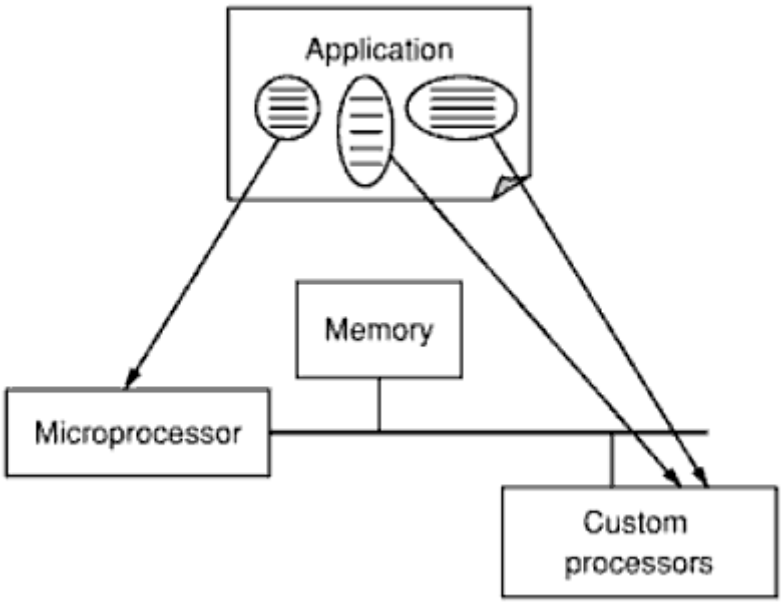

Abbildung 1.1: Partitionierung: Teilung der Applikation in Mikroprozessor-Komponenten (Software) und Beschleuniger-Komponenten (Hardware) (Vahid und Stitt, 2008)

1.2 Ziel und Anwendung

Diese Ausarbeitung gibt einen Überblick über die Aspekte der Partitionierung im Rahmen des HW/SW-Codesigns. Zum einen sollen dadurch Themen, die sich für eine nähere Auseinandersetzung eignen, aufgedeckt werden.

Zum anderen besteht für dieses Thema ein direkter Anwendungsbezug: Der Formula Student Rennwagens des Hawks Racing Teams wird entwickelt und gebaut von Studenten der Hochschule für Angewandte Wissenschaften Hamburg, sowie mit einem Telemetrie- und Fahrassistenzsystem ausgerüstet. Dazu sind eine umfassende Auswahl von Sensoren und Mikrocontrollern integriert. Letztere übernehmen das Ermitteln, Speichern und Fernübertragen der Sensorwerte. Andreae (2008) untersucht in seiner Ausarbeitung, ob die Integration von FPGAs bzw. die Verwendung eines SoC-Konzeptes im Rennwagen eine vorteilhafte Alternative zum bestehenden System ist. Diese Arbeit dagegen nimmt die Verwendung eines SoC-Designs als Ausgangspunkt und untersucht die Technologien der SoC-Partitionierung.

1.3 Die erste SoC-Lösung

Ein erster Ansatz, die zuvor genannten Verbesserungen zu erreichen, wäre die spontane Umsetzung einer SoC-Platform. Auf einem aktuellen Entwicklungsboard wie dem Nexys-2 von Digilent Inc. (Digilent Nexys-2, 2008) sind die wichtigsten Komponenten vorhanden. So enthält das Nexys-2 einen Xilinx Spartan-3E FPGA, diverse Schnittstellen sowie Erweiterungs-Anschlüsse (vgl. Abb. 1.2). Diese können flexibel mit den jeweiligen Erweiterungsmodulen zu den benötigten Schnittstellen umfunktioniert werden, um auch selten genutzte Schnittstellen betreiben zu können.

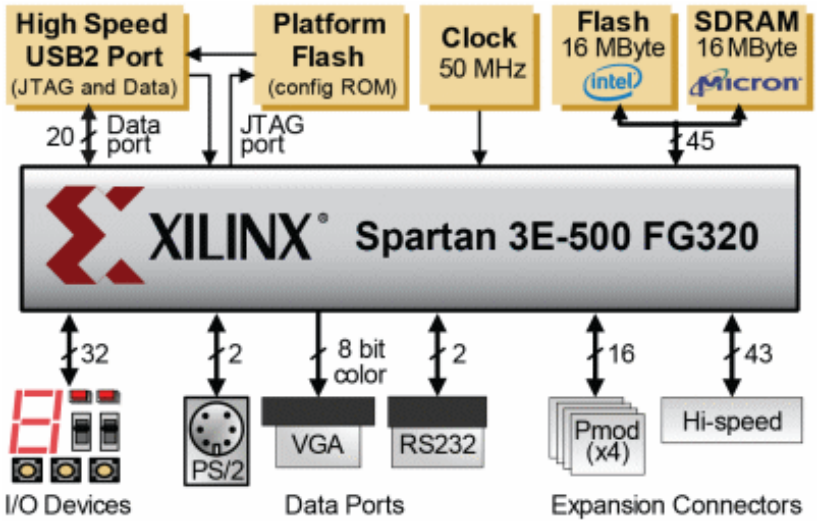

Abbildung 1.2: Schnittstellendiagramm des Spartan-3E FPGAs des Nexys-2-Entwicklungsboard. (Digilent Nexys-2, 2008)

Die vorhandene Software der zu ersetzenden Komponenten würde experimentell daraufhin untersucht werden, welche Teile des Programmes in Hardware realisiert werden könnten. Eine solche Lösung kann durchaus funktional und effizienter sein als die aktuelle Implementierung. Jedoch hängt die Qualität der Umsetzung von der Erfahrung oder dem Schicksal des Programmierers ab.

1.4 Gliederung dieser Ausarbeitung

Um Kompetenz in dem Bereich der Partitionierung zu erlangen, werden im Kapitel 2 Grundlagen die wichtigsten Aspekte der Partitionierung geschildert. Das Kapitel 3 Ausblick zeigt daraufhin mögliche Vertiefungsrichtungen. Eine Zusammenfassung dieser Ausarbeitung erfolgt im Kapitel 4 Zusammenfassung.

2 Grundlagen

Dieses Kapitel zeigt die grundlegenden Aspekte der Hardware/Software-Partitionierung.

2.1 Amdahl's Law - maximale Rechenzeitverkürzung

Wenn ein reines Softwareprogramm in ein CoS-Design umgewandelt wird, gibt es ein rechnerisches Maximum des Rechenzeitverkürzungsfaktors. Diese Formel wurde von Gene Amdahl (vgl. Vahid und Stitt, 2008, S. 542) aufgestellt:

$$max\_speedup = \left(s + \frac{p}{n}\right)^{-1} \tag{2.1}$$

Dabei gibt s den prozentualen Anteil des Programmes an, der nach der Umwandlung in Software verbleibt und p den in Hardware realisierten Anteil. n ist die Anzahl der Hardware-Beschleuniger, die sich parallel arbeitend die Hardware-Aufgaben teilen. Beispielsweise ergibt ein Programm, welches zu 75% parallelisierbar ist, folgende Gleichung:

$$max\_speedup(n) = \left(25\% + \frac{75\%}{n}\right)^{-1} \tag{2.2}$$

Bei lediglich einem Beschleuniger ergibt sich kein Vorteil, da auch alle Hardware-Aufgaben sequentiell verarbeitet werden müssen:

$$max\_speedup(1) = \left(25\% + \frac{75\%}{1}\right)^{-1} = \frac{1}{100\%} = \underline{1} \tag{2.3}$$

Stehen dagegen ausreichend Beschleuniger zur Verfügung, und lassen sich die Hardware-Aufgaben entsprechend fein verteilen, konvergiert die benötigte Ausführungszeit der Hardware-Aufgaben gegen Null. Hierbei wird eine messbare Gesamtbeschleunigung erreicht, deren Ausmaß vom Software-Anteil abhängt.

$$max\_speedup(\infty) = \left(25\% + \frac{75\%}{\infty}\right)^{-1} = \frac{1}{25\% + \sim 0} = \underline{4} \tag{2.4}$$

Die Rechenzeitverkürzung ist also begrenzt durch die Regionen (siehe Abschnitt 2.2), die nicht in HW verschoben werden, da diese nicht parallelisiert werden können. Es sollte also für ein Maximum an Rechenzeitverkürzung ein Maximum an Funktionalität in die Hardware verschoben werden (vgl. Vahid und Stitt, 2008, S. 542). Hierbei wird zugrunde gelegt, dass Software-CPU und Hardware-FPGA nicht parallel arbeiten, sondern alternierend. Dies ist mit etwas Mehraufwand auch mit Überschneidungen möglich, wie der Abschnitt 2.4 auf der nächsten Seite zeigt.

2.2 Regionen und Granularität

Zur Partitionierung wird die Funktionalität in Regionen aufgeteilt. Jede Region wird also entweder in Hardware, oder in Software implementiert. Daraus ergeben sich bei n Regionen theoretisch 2^n Partitionierungen. Diese exponentielle Komplexität macht bei entsprechend vielen Regionen die Betrachtung aller Kombinationen in der Praxis unmöglich. Je komplexer das Problem ist, desto länger dauert die Synthese, oder je raffinierter müssen die Heuristiken sein, um den Partitionierungsprozess in absehbarer Zeit vollenden zu können.

Die Anzahl der Regionen einer Funktionalität lassen sich durch die Größe der Regionen beeinflussen. So kann man ganze Funktionen, einzelne Code-Blöcke oder besondere Konstrukte wie Schleifen als Regionen betrachten. Größere Regionen verringern zwar die Komplexität des Problems. Jedoch lässt sich mit feineren Regionen gegebenenfalls eine optimalere Partitionierung erreichen. Um ein gutes Verhältnis von Partitionierungsaufwand zu Rechenzeitverkürzung zu erreichen, ist eine Aufteilung in unterschiedlich umfangreiche Regionen möglich. Dabei werden die Programmteile feiner aufgeteilt, die am zeitaufwändigsten sind, also relativ zu anderen Regionen den Prozessor länger belegen oder öfters aufgerufen werden, sowie jene Teile, die bei Verlagerung in die Hardware die größere Rechenzeitverkürzung bewirken würden (vgl. Vahid und Stitt, 2008, S. 545).

2.3 Alternative Implementations

Insbesondere im Bereich der Algorithmen gibt es viele Wege, ein Verhalten in Hardware zu implementieren. Diese unterschiedlichen Wege lassen sich zudem jeweils besser oder schlechter in Hardware bzw. in Software umsetzen. Die weiteren Metriken außer Acht lassend, müssen die Kriterien der geringen Gatterzahl und der kurzen Berechnungszeit bewertet werden. Da diese gegensätzlich zueinander stehen, muss zwischen der Lösung mit der geringsten Gatterzahl, der mit der kürzesten Berechnungszeit und einer der kompromissbildenden Lösungen entschieden werden.

Zur beispielhaften Veranschaulichung diene ein Grafikfilter, der durch die Verrechnung der Quellbildpunkte ein Ausgabebild erzeugt (vgl. Ashenden, 2008, S. 405).

Lösung mit geringster Gatterzahl
Es gibt eine einzige Verrechnungseinheit, die alle Ausgabebildpunkte sequentiell berechnet. Die dabei benötigte Gatterzahl ist die geringstmögliche, die Berechnungszeit für ein Bild benötigt dagegen ein vielfaches der folgenden Lösung und ist von der Bildgröße abhängig.

Lösung mit geringster Durchlaufzeit
Es gibt für jeden Ausgabebildpunkt eine Verrechnungseinheit, sodass die Ausgabebildpunkte alle in einem Schritt nebenläufig berechnet werden können. Die Berechnungszeit ist die geringstmögliche, die benötigte Gatterzahl beträgt jedoch ein vielfaches der vorigen Lösung und ist von der Bildgröße abhängig.

Kompromisslösungen
Eine Kompromisslösung besteht beispielsweise aus zwei nebenläufig arbeitenden Verrechnungseinheiten, die sich das Bild teilen. Alternativ gibt es für jede Bildzeile eine Verrechnungseinheit. Mit der Spannweite zwischen den beiden obigen Lösungen variiert die Menge der Kompromisslösungen.

Da eine Vielzahl der Alternativen wiederum die Komplexität erhöht, ist es zweckmäßig, sich hierbei auf wenige, aber repräsentative Alternativen zu beschränken, etwa die beiden extremen Varianten und ein bis drei Lösungen aus dem Kompromissbereich (vgl. Vahid und Stitt, 2008, S. 548). Eine weiteres Beispiel ist die elementweise Multiplikation von zwei 100er-Arrays mit einer oder hundert Multiplikationseinheiten (vgl. Abb. A.1 auf Seite 16).

2.4 CPU und FPGA alternierend oder überlappend

Bei der Verwendung von CPU und FPGA in einem CoS-Design kann man diese exklusiv alternierend, oder parallel arbeiten lassen (vgl. Abb. 2.1 auf der nächsten Seite). Die parallele Ausführung verkürzt den Programmzyklus zusätzlich zur Rechenzeitverkürzung durch den Beschleuniger selbst. Wenn etwa jeweils die Hälfte der Programmausführungszeit auf CPU und FPGA fällt, lässt sich durch eine parallele Ausführung die Zykluszeit halbieren.

Jedoch ist gegebenenfalls ein Mehraufwand durch Datensynchronisation zu bewältigen, um Dateninkonsistenzen zu verhindern. Dieser Mehraufwand ist im Allgemeinen nicht gerechtfertigt, wenn der Hardware-Teil nur einen unwesentlichen Anteil der benötigten Gesamtzykluszeit beansprucht und so der Zeitgewinn durch parallele Ausführung nur einen unwesentlichen Vorteil bringen würde (vgl. Vahid und Stitt, 2008, S. 550).

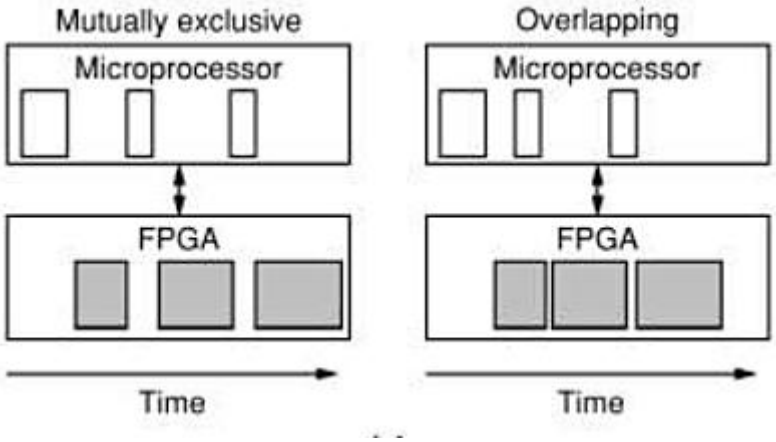

Abbildung 2.1: Implementation models: exklusiv alternierende oder überlappende Ausführung. (Vahid und Stitt, 2008)

2.5 Kopplung von CPU und FPGA

Für die Anbindung der Beschleuniger an den Prozessor gibt es verschiedene Möglichkeiten. Die direkteste Variante bildet die Kommunikation über Register des Prozessors. Dies ist aus Sicht des Prozessors die simpelste und schnellste Möglichkeit, Daten mit dem Beschleuniger auszutauschen. Da diese Register begrenzt zur Verfügung stehen, sollten hier die am häufigsten vom Prozessor angesprochenen Beschleuniger angebunden werden.

Als Variante kann der Beschleuniger am Adress-Daten-Bus des Prozessors, oder über eine Bridge auch an einem weiteren Bus angebunden sein. Dies ist ausreichend, wenn der Prozessor nur selten mit dem Beschleuniger Daten austauschen muss und der Beschleuniger für die Berechnungen intensiv über den entsprechenden Bus kommuniziert (vgl. Vahid und Stitt, 2008, S. 552).

2.6 Optimierungsziele

Durch die Vielzahl der möglichen Optimierungsziele lässt sich das Partitionierungsergebnis und der Partitionierungsaufwand den gewünschten Umständen anpassen (vgl. Vahid und Stitt, 2008, S. 547).

Performanz

Der Grundgedanke der Partitionierung ist die Beschleunigung der Programmausfüh-

rung, um weitere Programmteile in den gegebenen Zeitrahmen hinzufügen oder diesen verkleinern zu können.

Chipfläche
Ein in der Serienproduktion wichtiges Kriterium ist die benötigte Chipfläche, also die Menge an Logikschaltkreisen. Meistens steht dieses Kriterium zu dem vorigen, der Performanz, in Konkurrenz, wie der Abschnitt 2.3 auf Seite 7 zeigt.

Energieverbrauch
Der Energieverbrauch ist insbesondere bei batteriebetriebenen Geräten beachtenswert. Er steigt etwa mit der Größe der aktiven Schaltkreise, vor allem aber mit der Taktgeschwindigkeit. Es ist daher vorteilhaft, wenn jeder Teil des Systems mit einem jeweils minimalen Takt betrieben werden kann.

2.7 Partitionierungsmethoden

Für die Partitionierung selbst gibt es viele Strategien. Sie basieren auf Algorithmen, wie dem Knapsack, oder auf heuristischen Ansätzen.

Für diese beiden gilt, dass eine Ausgangskonstellation gewählt werden muss. So kann mit einer all-in-software-Basis begonnen werden, sodass im Sinne des Algorithmus Regionen aus der Hardware in die Software verschoben werden. Umgekehrt kann von einer all-in-hardware-Konstellation ausgegangen werden. Eine dritte Variante startet mit einer leeren Verteilung, bei der die Regionen erstmals der Hardware oder der Software zugewiesen werden.

Weiterhin kann das Maximum einer Metrik, oder das Optimum aller Metriken gesucht werden. Auch können zu erreichende Grenzwerte für Metriken das Verfahren vereinfachen (vgl. Vahid und Stitt, 2008, S. 547).

2.8 Softcore MicroBlaze als aktuelles SoC-Design

Der MicroBlaze ist ein spezieller Softcore-Mikrocontroller von Xilinx. Als Softcore ist er nicht als Hardwarbaustein, sondern nur als Quellcode verfügbar, welcher wiederum für verschiedene Xilinx-FPGAs synthetisiert werden kann. Dieser Mikrocontroller in dem FPGA lässt sich dann wie ein gewöhnlicher Mikrocontroller programmieren und benutzen. Zusätzlich ist es möglich, weitere Peripherie-Komponenten in den FPGA zu synthetisieren, und diese dabei direkt an den MicroBlaze anzubinden (vgl. Abb. 2.2 auf Seite 12).

Die Integration des Mikrocontrollers in den FPGA hat diverse Vorteile: durch die Kombination entfällt ein Teil der Platinenleiterbahnen, und es wird weniger Platinenfläche belegt. Die Beschleuniger und Peripherie im FPGA mit den jeweiligen Anbindungen an den Mikrocontroller können jederzeit ohne Änderung an der Hardware modifiziert werden. Die Anbindung dieser Komponenten an den Mikrocontroller geschieht über Register oder den „Processor Local Bus“ (PLB).

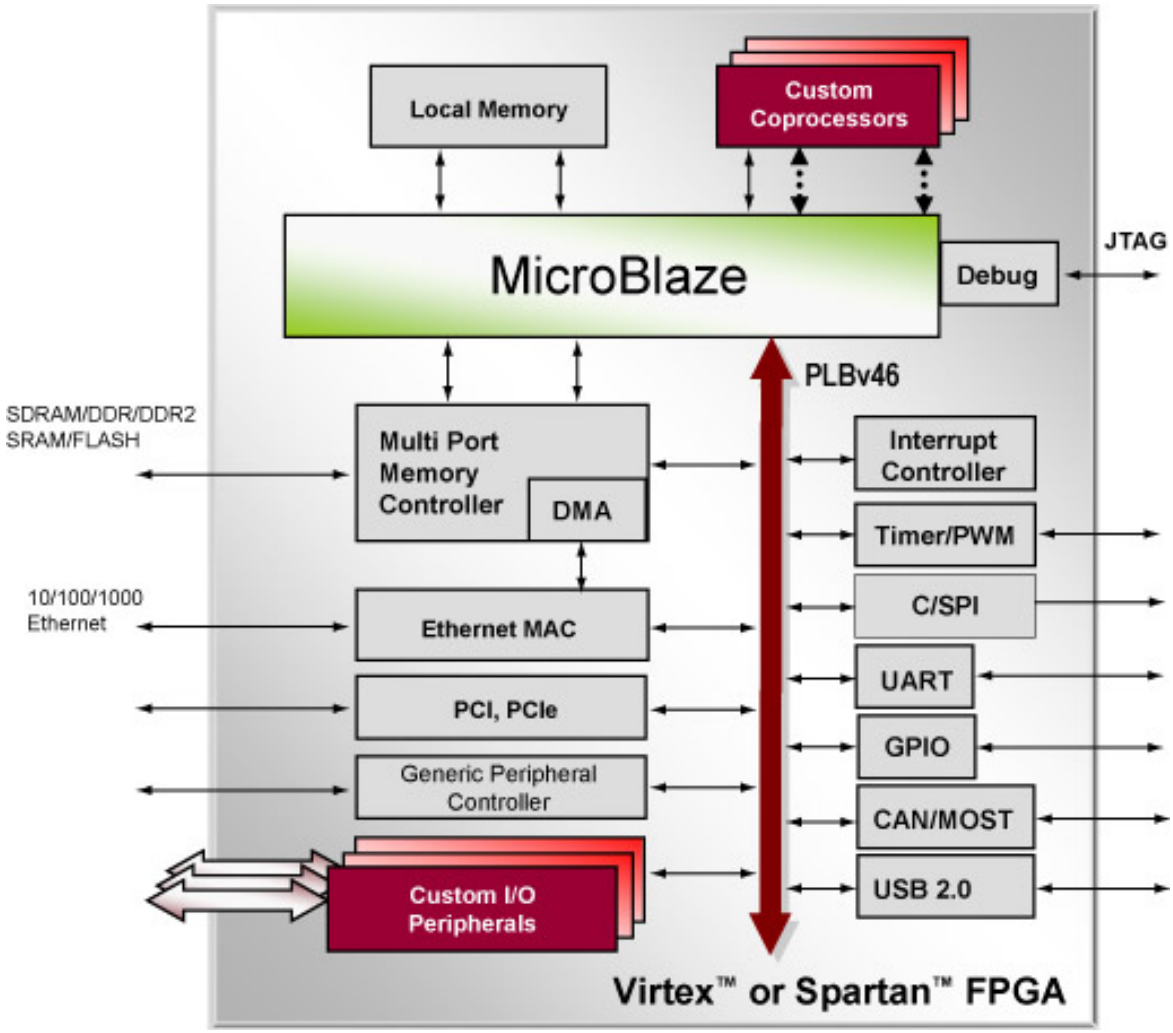

Abbildung 2.2: Busanbindungen des MicroBlaze-Prozessors. Beschleuniger entsprechen den 'Custom Coprocessors' des oberen roten Kastens. (Xilinx MicroBlaze, 2008)

3 Ausblick

Der Ausblick ist unterteilt in einen methodischen und einen operativen Teil, um die theoretischen Überlegungen und die praktische Umsetzung getrennt zu betrachten.

3.1 Methodischer Ausblick

Um tiefer in das Thema einzusteigen, stellen die Algorithmen und Heuristiken ein umfangreiches Themengebiet dar. Um eine Hardware/Software-Partitionierung zu entwickeln müssen Granularität, alternative Implementierungen sowie Implementierungsmodelle wie Überlappung und Beschleunigeranbindung bedacht werden. Die automatisierte Partitionierung ist ein nur im Ansatz erschlossenes Gebiet. Die Schwierigkeit hierbei liegt in den Unterschieden zwischen den Quelltexten von Hardware und Software, sodass es keine triviale Konvertierung zwischen diesen Implementierungen gibt. Um dieses Dilemma zu umgehen sind abstrakte Modellierungssprachen wie SystemC entworfen worden. Diese beschreiben die gesamte Funktionalität unabhängig von einer favorisierten Implementierung in Hardware oder in Software. Die Partitionierung soll nun aus diesem Modell eine optimale Aufteilung ermitteln, sodass die Funktionalität autonom kompiliert bzw. synthetisiert werden kann. Mit automatisierter Partitionierung lässt sich der Entwicklungsprozess für eingebettete Systeme beschleunigen, was wegen der Verbreitung von SoC-Systemen in den verschiedensten Anwendungsbereichen sicherlich auf Interesse der Industrie stößt (vgl. Vahid und Stitt, 2008, 5558).

3.2 Operativer Ausblick

Um die Umsetzbarkeit der Erkenntnisse zu überprüfen sollte auf Basis eine SoC-Konzeptes eine Alternativlösung zu dem bestehenden System des Rennwagens entwickelt werden. Eine dafür geeignete Platform wurde mit dem Nexys2-Entwicklungsboard vorgestellt. In der praktischen Anwendung kommen gegebenenfalls unerwartete Hindernisse zu Tage, die für ein Vertiefung interessant sind.

4 Zusammenfassung

Diese Ausarbeitung hat einen Überblick über die Partitionierung im Bereich des Hardware/Software-Codesigns gegeben. Dazu wurden die wichtigsten Aspekte für einen Einstieg in den Themenbereich erklärt. Algorithmen und Heuristiken konnten wegen des begrenzten Umfangs nur angedeutet werden. Mit dem MicroBlaze wurde eine aktuelle Technologie angeführt, mit der die gewonnenen Kenntnisse umgesetzt werden können. Anschließend wurden im Ausblick Vorschläge zur weiteren Vertiefung des Themas genannt.

Literaturverzeichnis

[Andreae 2008] ANDREAE, Johann-Nikolaus: *Konzeptperspektive für die Sensorik- und Mikrocontrollerplattform im Formula Student Rennwagen*, Hamburg: Hochschule für Angewandte Wissenschaften, Dep. Informatik, Ausarbeitung, Dezember 2008

[Ashenden 2008] ASHENDEN, Peter: *Accelerators*. Kap. 9. In: *Digital Design : An Embedded Systems Approach Using VHDL*, Morgan Kaufmann, 2008. – ISBN 0-12-369528-7

[Digilent Nexys-2 2008] DIGILENT INC.: *Nexys-2*. 2008. – URL `http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS2`. – Abruf: 2008-12-14

[Vahid und Stitt 2008] VAHID, F. ; STITT, G.: *Hardware/Software Partitioning*. Kap. 26, S. 539–560. In: HAUCK, S. (Hrsg.) ; DEHON, A. (Hrsg.): *Reconfigurable Computing : The Theory and Practice of FPGA-Based Computation*, Morgan Kaufmann/Elsevier, 2008. – ISBN 0-12-370522-3

[Wolf 2007] WOLF, Wayne: *Hardware and Software Co-design*. Kap. 7. In: *High Performance Embedded Computung : Architectures, Applications, and Methodologies*. San Francisco : Elsevier, 2007. – ISBN 0-12-369485-X

[Xilinx MicroBlaze 2008] XILINX, INC.: *MicroBlaze Processor*. 2008. – URL `http://www.xilinx.com/products/design_resources/proc_central/microblaze.htm`. – Abruf: 2008-12-14

A Alternative Imlementations

Die Abbildung A.1 zeigt die elementweise Multiplikation von zwei 100er-Arrays als Beispiel für Alternative Implementations, siehe auch Abschnitt 2.3 auf Seite 7.

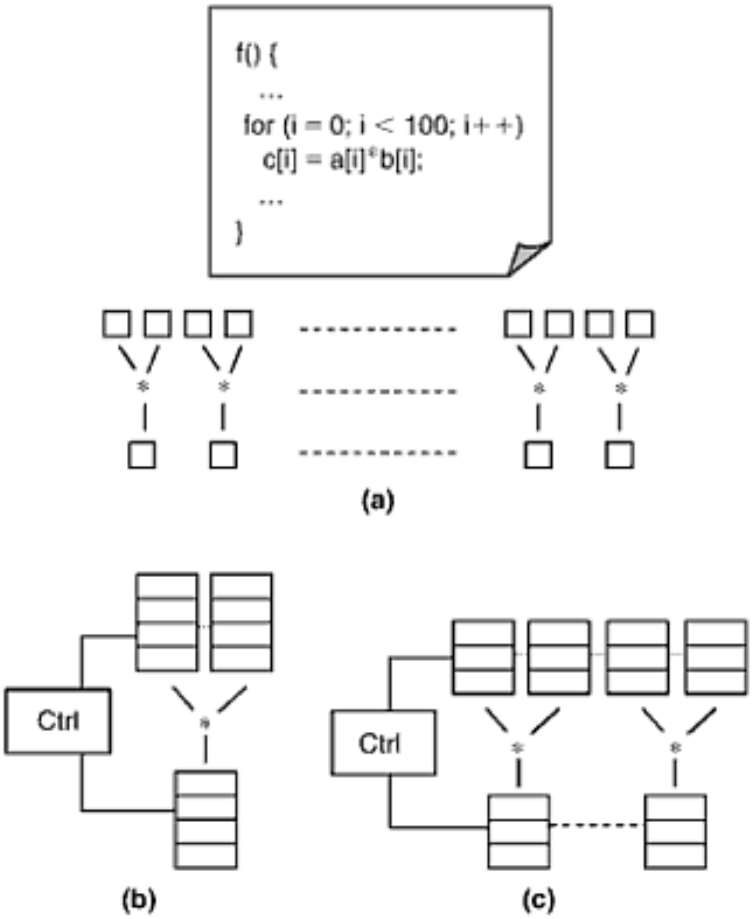

Abbildung A.1: Alternative Regions-Implementierungen einer 100 Multiplikationen benötigenden Applikation (oben): (a) 100 Multiplizierer; (b) 1 Multiplizierer; (c) 2 Multiplizierer (Vahid und Stitt, 2008).

Abbildungsverzeichnis

Abkürzungsverzeichnis

Bezeichnung	Beschreibung	Seiten
ALU	Arithmetical Logical Unit, die Recheneinheit einer CPU, üblicherweise fähig zu arithmetischen, logischen und schiebe-Operationen	3
CPU	Central Processing Unit, zentrale Rechen- und Steuereinheit zur Softwareausführung	7
FPGA	Field Programmable Gate Array, ein auf unterster Ebene programmierbarer Logikbaustein	3, 7, 10
SoC	System-on-a-chip ist die Kombination vieler Systemkomponenten, wie etwa Prozessor, Controller, Speicher, Beschleuniger und Schnittstellen auf einem Halbleiterchip	3–5, 13