FPGA basierter Prototyper für die Verifikation von Hardware-Software Systemen

Th. Benner, R. Ernst, I. Könenkamp, U. Holtmann,
P. Schüler, H.-C. Schaub, N. Serafimov

Technische Universität Braunschweig
Institut für Datenverarbeitungsanlagen
D–38106 Braunschweig, Germany

15. April 1994

Erweiterte Kurzfassung

Einführung

COSYMA (COSYnthesis of eMbedded Architectures) ist eines der ersten Systeme für die automatisierte Cosynthese von Hardware und Software [1]. Es ist für den Entwurf von kleinen eingebetteten Controllern bestimmt. Aus einem in C^x, einem C-Dialekt, der gegenüber ANSI-C um Zeitrestriktionen und einem Task-Konzept erweitert wurde, beschriebenen Entwurf bildet COSYMA soviel Software wie möglich auf einen gegebenen Prozessor ab. Falls Zeitbedingungen durch eine reine Softwarelösung nicht erfüllt werden können, wird aus einem Teil der C^x Beschreibung automatisch ein anwendungsspezifischer Coprozessor generiert, um mit minimalem Hardwareaufwand die Zeitbedingungen erfüllen zu können.

Abbildung 1 gibt einen Überblick über den Entwurfsprozeß. Basierend auf einer Abschätzung des Speedups, der Kommunikationszeit und des Harwareaufwandes erfolgt die Partitionierung auf der Ebene von Grundblöcken mit Hilfe des Simulated Annealings in einen Hardware- und einen Softwareanteil. Die Software wird um einen Kommunikationsmechanismus ergänzt und in ausführbaren Code übersetzt. Mit Hilfe der High-Level Synthese wird die Hardware generiert. Der Ansatz erlaubt es, hierzu verschiedene Synthesesysteme einzusetzen. Derzeit setzen wir OLYMPUS [2], das HardwareC als Eingabesprache verwendet, sowie unser eigenes System BSS [3] ein, das einen CDFG als Eingabe benötigt. Die Struktur des Datenpfades und des Contollers des Coprozessors ermöglichen eine exakte Abschätzung der Laufzeit des resultierenden HW/SW-Systems auf der Basis von Taktzyklen [2], um festzustellen, ob die Zeitbedingungen erfüllt sind. Die dabei verwendeten Schätzungen bauen auf Scheduling Informationen und dem ausführbaren Code auf. Derzeit ist eine Adaptierung in der Entwicklung, die mit Hilfe dieser Schätzungen das Partionierungsergebnis korrigieren wird.

Sobald alle Zeitbedingungen erfüllt sind, generiert die High-Level Synthese eine hierarchische Netzliste auf Logikebene, welche eine Beschreibung des Datenpfades und des Controllers im SLIF-Format (Stanford Intermediate Logic Format) enthält. Diese Netzliste verifizieren wir durch Simulation auf der Logikebene, wodurch wir selbst dann auf sehr kleine Beispiele beschränkt sind, wenn das Prozessormodell als Register-Transfer Beschreibung vorliegt. Aus diesem Grunde haben wir uns für die Verwendung eines Hardware-Prototypers entschieden.

Anforderungen an den Prototyper

Da die Architektur des erzeugten HW/SW-Systems auf Systeme mit Speicherkopplung beschränkt ist, schien es sinnvoll, an der Stelle eines kostspieligen kommerziellen Prototypers, der alternativ um einen SPARC-Prozessor hätte erweitert werden müssen, ein eigenes System zu entwickeln. Auf Grund der einfachen Architektur des Coprozessors läßt sich erreichen, daß das System beinahe unter Echtzeitbedingungen betrieben werden kann (33MHz getakteter SPARC-Prozessor). Eine Hauptanforderung war, daß der gesamte Entwurfsprozeß, von der High-Level Synthese ausgehend, automatisiert ohne die Notwendigkeit manueller Eingriffe ablaufen können soll.

Abbildung 2 gibt einen Überblick über den Prototyper. Der SPARC-Prozessor und das FPGA-Board kommunizieren über einen gemeinsamen Speicher. Ein Microcontroller MC 68332 überwacht das System und bildet gleichzeitig die Schnittstelle zum Hostrechner.

Die Architektur des Prototypers

Die Hauptaufgabe des Prototypers besteht in der Emulation eines anwendungsspezifischen Coprozessors. Ein solcher Copropessor kann aus bis zu fünf ALUs und 20 Registern bestehen, und ist somit nicht mehr auf einem einzigen FPGA zu realisieren. Aus diesem Grund wird der Entwurf in mehrere Teile aufgespalten. Um ein Maximum an Parallelität zu erreichen, müssen die ALUs und Register vollständig miteinander verbunden sein. Die Anzahl der Pins der erhältlichen FPGAs erlaubt es nicht, komplette ALUs und Register auf einzelne FPGAs abzubilden und extern miteinander zu verbinden. Als Alternative hierzu bietet sich eine Bit-Slice Architektur (Abb. 3) an. Die Coprozessor-Platine enthält vier XILINX XC4010, welche jeweils auf einem acht Bit breiten Slice arbeiten. Auf jedem XC4010 befinden sich fünf 8-Bit-ALUs, ein Registersatz und ein Barrelshifter. Multiplikationen werden extern auf einem AMD AM29C323 ausgeführt.

Derzeitig arbeiten der SPARC und der Coprozessor im gegenseitigen Ausschluß und kommunizieren miteinander über einen gemeinsamen Speicher. Um den Grad an Parallelität zu erhöhen, greift jeder Slice auf zwei getrennte Speicherbänke zu. Somit kann der Coprozessor gleichzeitig zwei 32-Bit-Werte lesen oder schreiben. Die beiden Speicherbänke werden in den Adreßraum des SPARCs eingeblendet.

Bislang wird der SPARC mit Hilfe eines HOLD-Signales angehalten, solange der Coprozessor arbeitet. Durch die parallele Ausfürung des SPARCs und des Coprozessors beabsichtigen wir, die Geschwindigkeit weiter zu erhöhen. Aus diesem Grunde haben wir dem Prozessor einen eigenen Befehlsspeicher spendiert, um insgesamt bis zu drei parallele Speicherzugriffe zu ermöglichen.

Wie bereits erwähnt, setzen wir einen Microcontroller zur Zeitmessung und zur Fehlersuche ein. Einer der Zeitgeber des MC68332 wird dazu durch ein Tag-RAM gesteuert, welches die Adressen von bis zu acht maskierbaren Breakpoints enthält. Sobald eine der ausgewählten Adressen auf dem Bus erscheint, empfängt der Microcontoller eine Unterbrechung und der Prozessor und der Coprozessor werden im selben Taktzyklus angehalten. Anschließend kann der Benutzer den Speicher auslesen.

Der Entwurfsprozess

Als Ausgabe liefert die High-Level Synthese ([3,4]) eine SLIF Netzliste. Diese beschreibt unabgängig von der verwendeten Technologie den anwendungsspezifischen Coprozessor. Nach ihrer Erzeugung wird die Netzliste in die Beschreibungen von vier Slices partitioniert (Abb. 4). Jeder Slice verwendet eine lokale Kopie der hierarchischen Netzliste, aus denen mittels eines in „perl“ geschrieben Programmes alle Signale entfernt wurden, die zu anderen Slices gehören. Der vollständige Controller liegt in allen Slices vor.

Die Beschreibungen von Modulen wie ALUs, RAMs enthalten auch jene Signale, die für die externe Verbindungen der Slices notwendig sind. Auf Grund der festen Verbindungsstruktur, ist die Anzahl der Signale zwischen dem Datenpfad und dem Controller beschränkt.

Nach der Übersetzung nach Verilog, wird jede FPGA-Netzliste getrennt synthetisiert und optimiert. Hierzu werden VHDL-Modelle der ALUs und des Barrelschifters verwendet. Die synthetisierten Slices werden auf die FPGAs gemappt. Die Xilinx Tools liefern als Ausgabe einen Bitstrom, der direkt auf die FPGAs geladen werden kann.

Ausblicke und Danksagungen

Wir beabsichtigen, COSYMA um kleine heterogene Multiprozessor-Systeme zu erweitern. Als ersten Schritt werden wir den Prototyper um den DSP 96002 erweitern. Dieses Board soll unter Verwendung von programmierbaren Verbindungsstrukturen (Aptix) entwickelt werden.

J. Herrmann und P. Meier waren für den mechanischen Aufbau verantwortlich und H.-O. Leilich gab sehr hilfreiche Hinweise bei der Fehlersuche.

Literatur

[1.] R. Ernst, J. Henkel und Th. Benner, Hardware/Software Co-Synthesis for Microcontrollers, IEEE Design & Test of Computers, S. 64–75, Dez. 1993.

[2.] W. Ye, R. Ernst, Th. Benner, J. Henkel, Fast Timing Analysis for Hardware-Software Co-Synthesis, Proc. of ICCD 1993, Cambridge, S. 452–457, 1993.

[3.] G. De Micheli et al., The Olympus Synthesis System, IEEE Design & Test of Computers, S. 37–53, Okt. 1990.

[4.] U. Holtmann, Hierarchical Behavioural Representation in the Braunschweig Synthesis System BSS, IFIP Workshop on Application of Synthesis and Simulation, Lenggries, 25.-28.8.1993.

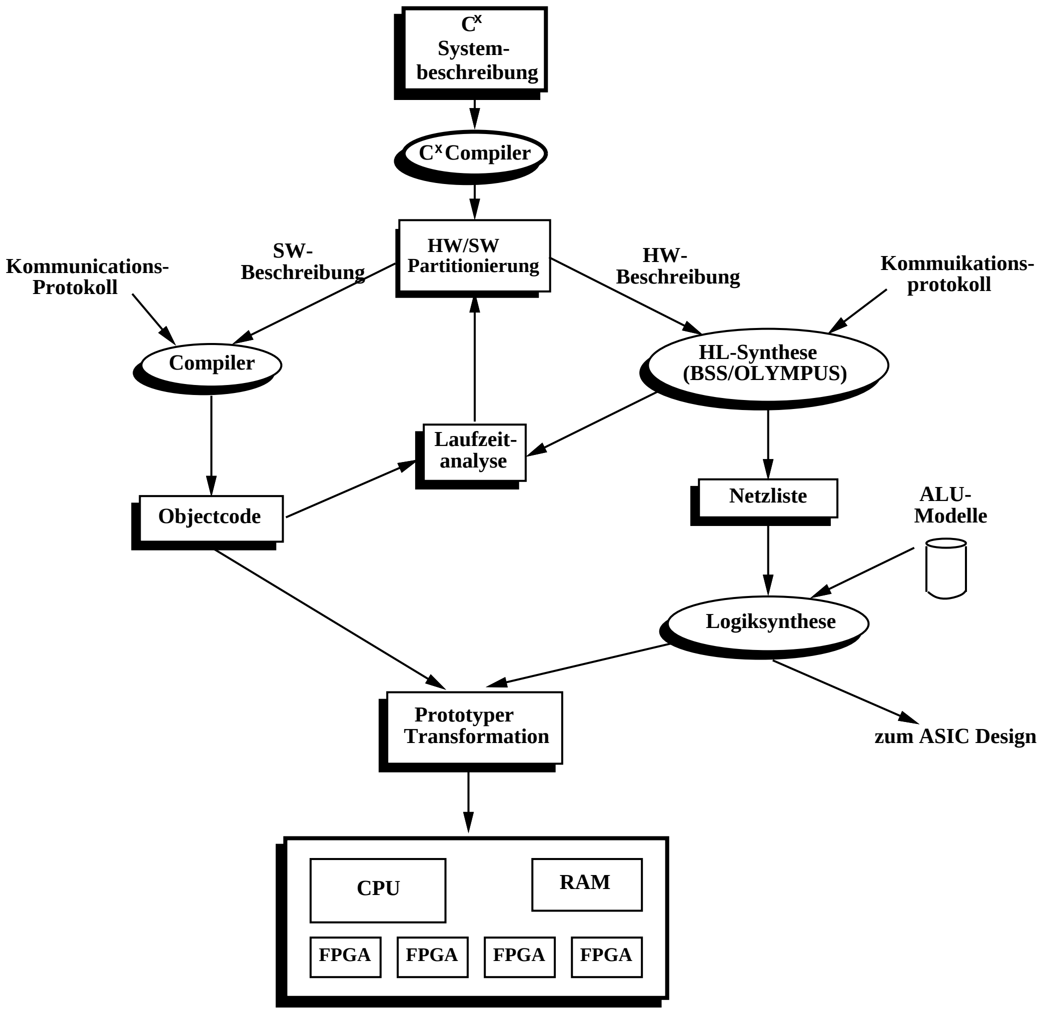

Abbildung 1: Das Experimentalsystem COSYMA.

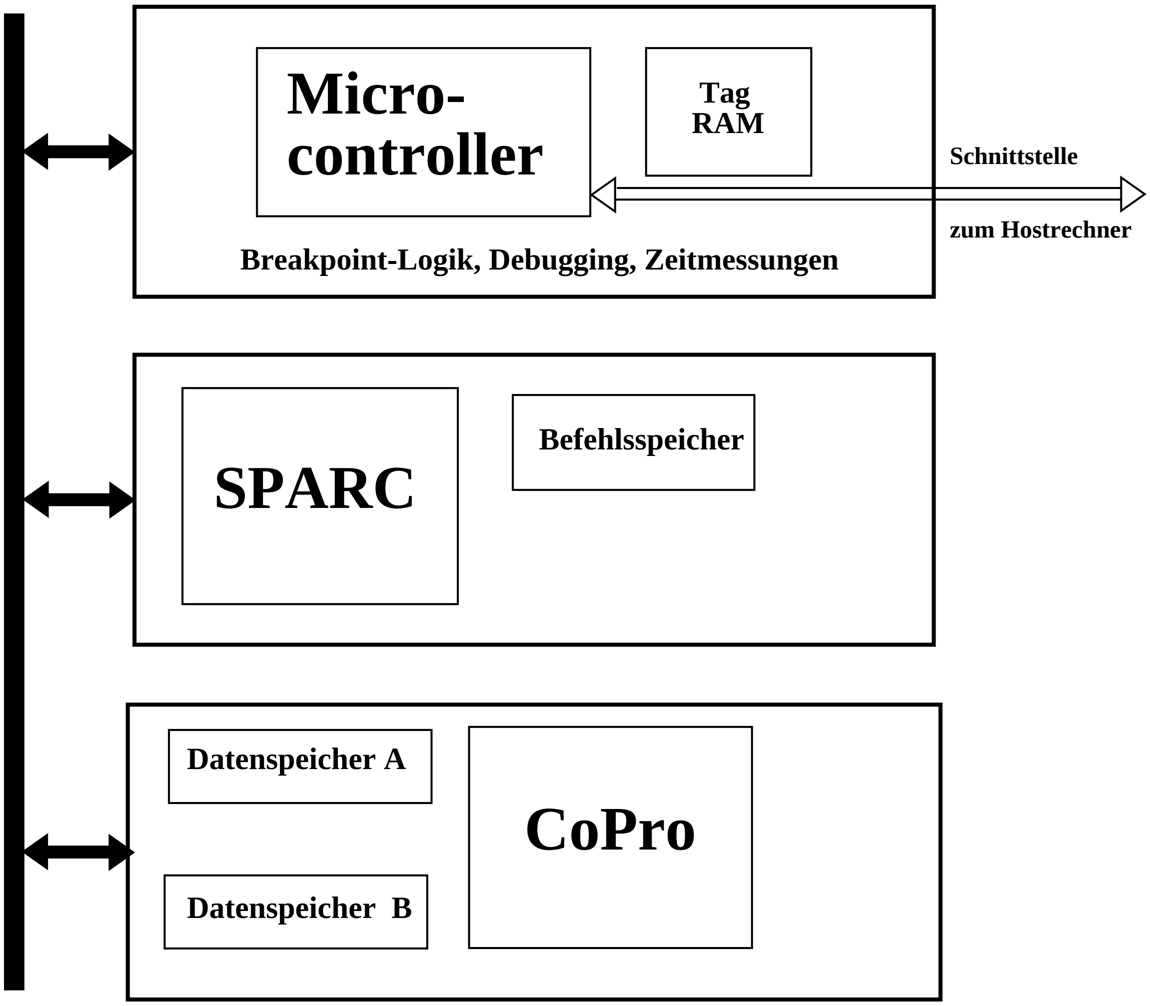

Abbildung 2: Die Architectur des Prototypers.

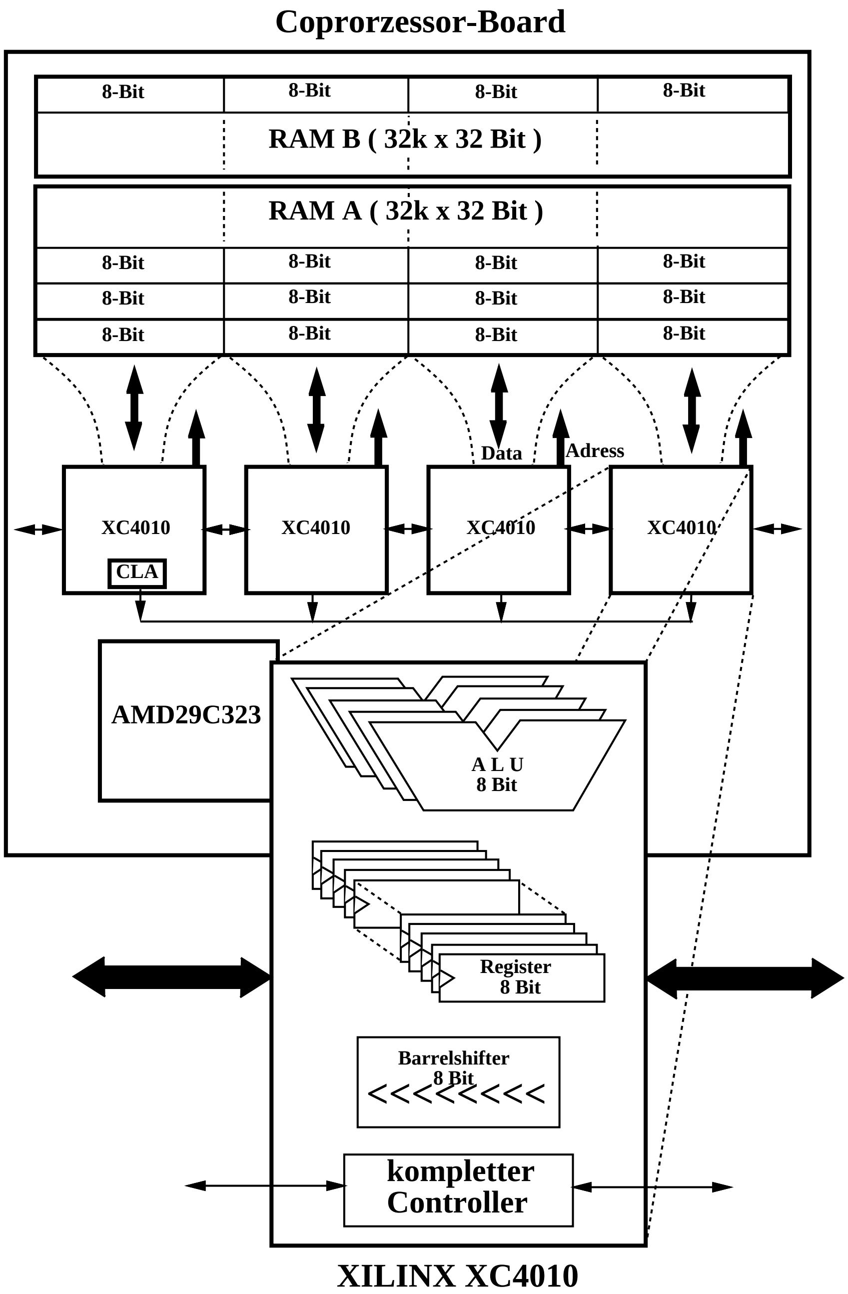

Abbildung 3: Die FPGA Platine.

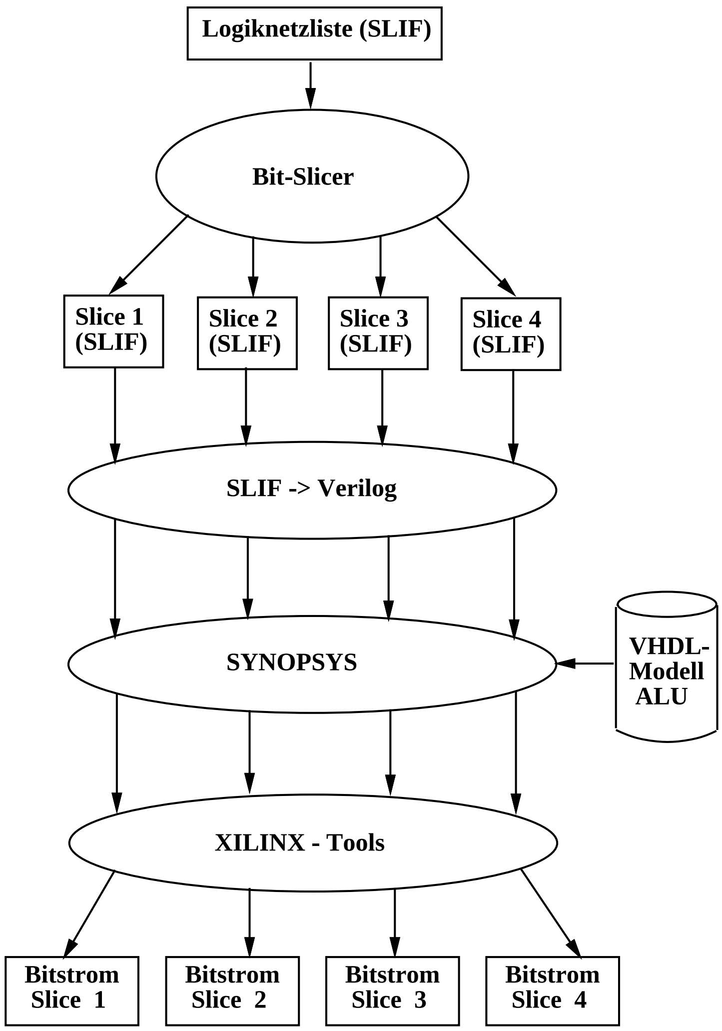

Abbildung 4: Der Entwurfsprozeß.