Adaptive Rechensysteme und ihre Entwurfswerkzeuge

Dr. ing. Andreas Koch, Abt. Entwurf integrierter Schaltungen, TU Braunschweig

Kurzfassung

Adaptive Rechensysteme mit reprogrammierbaren und rekonfigurierbaren Chips sind eine Möglichkeit, das Problem von immer weiter wachsenden Gatter-Kapazitäten bei ebenso steigenden Herstellungskosten zu beherrschen. Wir haben Architekturen für solche Systeme untersucht und zu Demonstrationszwecken eine konkrete Hardware/Software-Ausprägung als diskrete Lösung realisiert (board-level). Die praktische Anwendbarkeit eines adaptiven Rechners wird daneben aber auch durch seine Entwurfswerkzeuge bestimmt. Wir beschreiben einen experimentellen Entwurfsfluß, bestehend aus Compilern, Datenpfadsynthese und Modulgeneratoren, der aus konventionellen C-Programmen die Rekonfigurierbarkeit ausnutzende kombinierte Hardware/Software-Lösungen erstellt.

1 Einleitung

Moderne Fertigungstechniken erlauben zwar die Herstellung immer größerer Chips, auf der anderen Seite wird aber jeder einzelne Fertigungslauf (von der Maskenerstellung zum Test) auf einem solchen Prozeß immer aufwendiger und damit für immer weniger Neuentwicklungen rentabel [1].

Ein Ausweg aus diesem Dilemma kann der Einsatz von programmierbaren bzw. konfigurierbaren Schaltungen sein, bei denen eine feste Chip-Plattform durch nachträgliche Anpassung auf die konkrete Anwendung zugeschnitten werden kann. Neben der allgemeinen Kostensenkung durch Zugriff auf einen Massenbaustein können durch das so erreichbare hohe Maß an Anpaßbarkeit Änderungen (z.B. während eines Normungsprozesses) ohne Risiko in den laufenden Entwurf einfließen.

Die hohe durch den Einsatz von Standardprozessoren erreichbare Flexibilität (Anwendung rein durch Software definiert) wird häufig durch den Zwang zur Überdimensionierung des Prozessors erkauft: Einzelne Funktionen der Anwendung, die sich möglicherweise effizient direkt in Hardware implementieren lassen, können rein in Software weniger gut realisierbar sein und erfordern zum Ausgleich einen insgesamt leistungsfähigeren Prozessor. Eine universell einsetzbare Lösung sollte daher neben der Software-Flexibilität auch in Hardware-Aspekten anpaßbar sein.

Solche *adaptiven* Rechensysteme basieren auf Technologien, die heute neben den klassischen FPGAs (die mittlerweile ausreichende Kapazitäten für reale Rechenanwendungen bereitstellen) und anwendungs-spezifischen Prozessoren (ASIPs) insbesondere auch sogenannte Network Processors (z.B. [2] [3]) umfassen. Letztere haben als konfigurierbare Einheiten in der Regel keine 1-Bit breiten Logikblöcke, sondern bauen auf Multi-Bit-Operatoren (wie ALUs) auf.

Entscheidende Leistungssteigerungen mittels adaptiver Rechner lassen sich nur erreichen, wenn die Werkzeuge im Entwurfsfluß schon auf hoher Ebene auf die Zielplattform hin optimieren. Dabei darf ein praktisch anwendbares System nicht die Anforderungen seines potentiellen Nutzerkreises aus den Augen verlieren: Zur Eingabe sollten dem Nutzer vertraute Notationen zum Tragen kommen, im DSP-Bereich also beispielsweise C, Fortran und MATLAB. Die Verwendung exotischer Eingabesprachen (und dazu zählen aus Sicht eines Software-Entwicklers auch HDLs!) wäre der breiten Anwendung hier eher hinderlich.

Seit 1997 kooperiert die Abteilung E.I.S. im Rahmen des Projekts “Nimble Compiler for Agile Hardware” unter anderem mit der Fa. Synopsys (Advanced Technology Group) und der Universität Berkeley (BRASS) mit dem Ziel, einen durchgehenden Entwurfsfluß zu entwickeln, der ohne Einschränkungen C in kombinierte HW/SW-Systeme übersetzen kann. Diese Arbeit beschreibt als Zwischenbilanz einen ersten, aus Hard- und Software bestehenden Systemprototypen.

2 Ziel-Architektur

Die *ideale* Ziel-Hardware für den Compiler ist in Abbildung 1.a skizziert. Sie besteht aus einem eng gekoppelten rekonfigurierbaren Datenpfad DP und einem konventionellen Prozessor CPU, die auf einen gemeinsamen Speicher zugreifen (DRAM, Cache). Optional kann der DP auch noch lokalen Speicher SRAM und eigene Peripherie I/O haben.

Der Datenpfad sollte dabei einige Dutzend bis einige Hundert ALU-Operationen gleichzeitig realisieren können und ausreichend Flip-Flops für den Aufbau von Pipelines enthalten. Je kürzer die Rekonfigurationszeit, desto effizienter kann der DP für unterschiedliche Teile der Anwendung wiederverwendet werden.

Entscheidend für die effiziente Ausführung von Algorithmen in konventionellen Programmiersprachen (C, Fortran etc.) ist der homogene Speicher, der sowohl CPU als auch

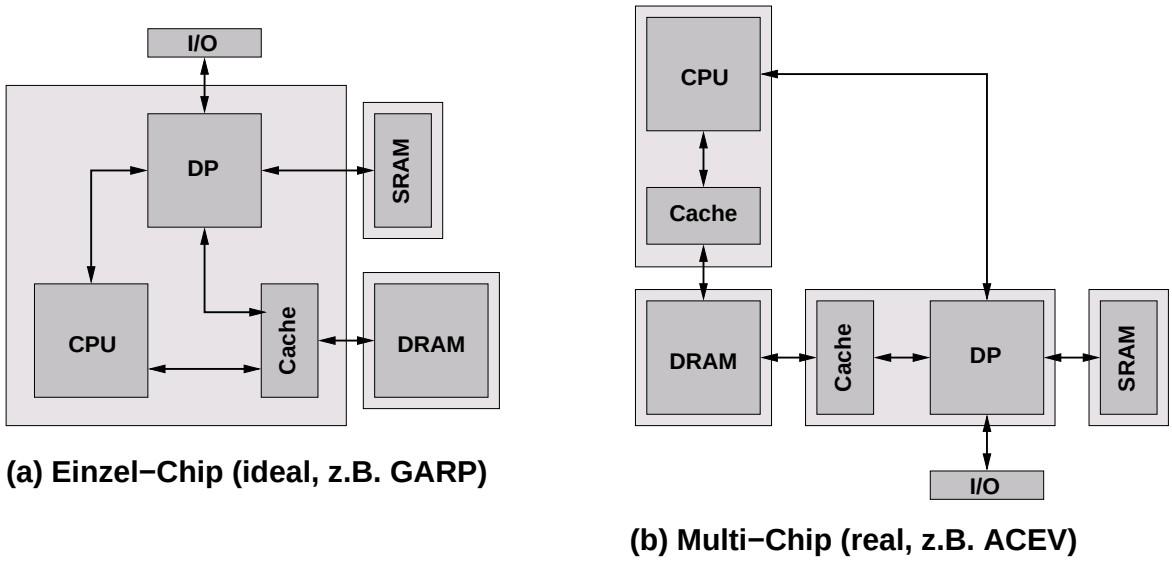

Abbildung 1: Hardware-Architekturen

DP zur Verfügung steht[1]. Ohne diese Fähigkeit müssten alle benötigten Daten vor Start einer Hardware-Anwendung explizit in den DP-eigenen Speicher kopiert werden. Neben einer allgemeinen Verkomplizierung der Speicherverwaltung (es muß immer zwischen CPU- und DP-zugänglichem Speicher unterschieden werden), wird dies gerade bei komplexeren Algorithmen, in denen der DP selbstständig Zeigerketten auflöst, hochgradig ineffizient (hier müssen sogar alle überhaupt *referenzierbaren* Daten kopiert werden).

Das gesamte Forschungsprojekt Nimble war von vorneherein auf die praktische Erprobung der theoretischen Ansätze ausgelegt. Da geeignete Bausteine mit den oben genannten Fähigkeiten erst seit kurzem verfügbar sind bzw. sein werden [4] [5], wurde in Form der Test-Plattform ACEV ein adaptiver Rechner aufgebaut, der diskrete Chips für die einzelnen Komponenten verwendet (Abbildung 1.b).

Im einzelnen wird dabei zur Realisierung der Datenpfade ein Baustein Xilinx Virtex XCV1000 [6] verwendet. Ein RISC microSPARC-IIep dient als CPU. Als Speicher stehen DP und CPU gemeinsam 64MB DRAM, dem DP lokal 4x1MB ZBT SRAM zur Verfügung. Die Kommunikation zwischen DP und CPU erfolgt über einen PCI-Bus. Um die Latenzen der vom DP ausgehenden DRAM-Zugriffe zu minimieren, verwendet auch der DP einen eigenen Cache. Die komplette Testplattform [7] steckt als PCI-Karte in einem Wirtsrechner, läuft aber von diesem unabhängig unter einem eigenen Betriebssystem (RTEMS [15]). Lediglich für Dateioperationen wird auf den Wirtsrechner zugegriffen.

Diese Umgebung erlaubt die praktische Erprobung von Entwurfswerkzeugen, Hardware-Architekturen und konkreten adaptiven Anwendungen auf Systemebene schneller als dies durch Simulationen möglich wäre. Dabei ist allerdings die absolute Rechenleistung bedingt durch den diskreten Aufbau stark beschränkt: Neben der niedrigen Bandbreite und hohen Latenz der PCI-Anbindung ist dafür auch die sehr langsame Rekonfigurationszeit der Virtex-FPGAs verantwortlich. Bei der zukünftigen Verwendung der oben genannten höher integrierten Lösungen (single-chip) mit schneller konfigurierbarer Struktur (ALU statt Logikblock) werden diese Einschränkungen aber aufgehoben.

Eine realistischere Einschätzung der tatsächlich erreichbaren Beschleunigung durch einen solchen Baustein wird durch Ausführung der HW/SW-Anwendung auf einem geeigneten Simulator erreicht. Dabei wird als Ziel der GARP-Chip [10] angenommen. Dieser kommt der in Abbildung 1.a gezeigten idealen Architektur sehr nahe: Neben einem MIPS-II-Kern als Prozessor steht ein Feld aus konfigurierbaren 32-Bit-ALUs sowie ein Cache für vier komplette Konfigurationen auf dem Chip zur Verfügung.

Neben der bisher diskutierten Hardware-Architektur auf Systemebene gilt es nun noch, die Architektur auf den rekonfigurierbaren Elementen selbst zu untersuchen. So ist es für unseren Ansatz, der ja die Programmierung durch eine Hochsprache unterstützen soll, nicht sinnvoll, einzelne Logikblöcke als primitive Elemente anzusehen. Stattdessen werden die Datenpfade aus Multi-Bit-Operatoren und Speichern aufgebaut. Diese werden mittels parametrisierter Modulgeneratoren (siehe unten) regulär vorplaziert und dann in eine feste Hardware-Umgebung eingebunden. Auf dem GARP-Chip ist diese Umgebung als feste Schaltung von vorneherein Bestandteil der Architektur. Für die ACEV-Plattform wird sie als Netzliste mit den synthetisierten Datenpfaden zusammengebunden.

Die Datenpfadumgebung (Abb. 2) stellt verschiedene Dienste bereit: Für den Datenpfad werden Standardschnittstellen zum lokalen SRAM sowie zum gemeinsamen

[1] Für andere 'models of computation', insbesondere die verschiedenen Datenfluß-Modelle, ist dies *nicht* erforderlich.

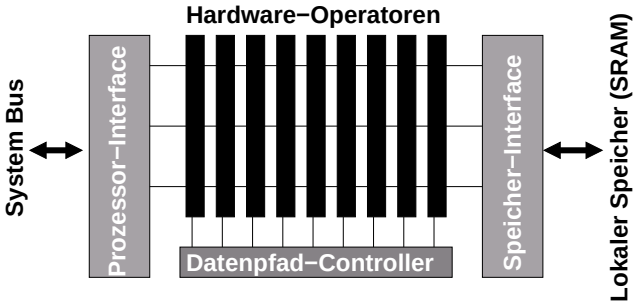

Abbildung 2: Umgebung für synthetisierten Datenpfad

DRAM aufgebaut, die mit einem allgemeinen System von On-Chip-Bussen an alle speicherzugreifenden Module angeschlossen werden. Die CPU kann durch die Prozessor-Schnittstelle beliebige Datenpfad-Register lesen und schreiben. Auf diese Weise können die Werte von Software-Variablen in Hardware-Register und zurück übertragen werden. Daneben werden auch noch administrative Funktionen wie das Auslösen eines Interrupts (Ende der Hardware-Ausführung) und die Identifikation der Interrupt-Ursache (auslösender Operator) unterstützt.

Die bisherigen Standardschnittstellen auf der ACEV-Plattform beinhalten einen einfachen Direct-Mapped-Cache für DRAM-Zugriffe (Vermeiden der langen PCI-Latenzen). Mittelfristig ist aber die Umstellung auf die mächtigere MARC-Schnittstelle [8] geplant. Diese stellt eine konfigurierbare Zahl von parallel verwendbaren Speicher-Ports zur Verfügung und realisiert neben verschiedenen Caches für irreguläre auch noch Streams für reguläre Zugriffe.

3 Entwurfsfluß

Der gesamte Nimble Entwurfsfluß ist in Abbildung 3 skizziert. Im Kern besteht er aus einem ANSI/ISO-kompatiblen C-Compiler und einem datenpfad-orientierten Plazierungswerkzeug.

Der in C vorliegende Quelltext wird für die weitere Bearbeitung zunächst in ein geeignetes Zwischenformat [9] übersetzt. Im folgenden wird das Programm mit verschiedenen klassischen architekturunabhängigen Optimierungsverfahren bearbeitet. Dazu zählen beispielsweise die Eliminierung der Berechnung gleicher Teilausdrücke und redundanter Speicherzugriffe. Gegebenenfalls kann an dieser Stelle ein dynamisches Profiling des Programmes mit anschließenden Visualisierungen der Ergebnisse vorgenommen werden.

Nun werden effizient in konfigurierbarer Hardware realisierbare Schleifen erkannt. Blöcke, die Fließkommaoperationen oder I/O-Anweisungen enthalten, werden dabei explizit ausgeschlossen. Diese Operationen sind entweder nicht oder nur mit hohem Flächenaufwand in rekonfigurierbarer Hardware ausführbar. Es werden aber Blöcke berücksichtigt, bei denen diese “illegalen” Operationen nur in bedingten Anweisungen mit ausreichend niedriger Ausführungswahrscheinlichkeit (z.B. nur bei der Fehlerbehandlung) auftreten. Hier wird der Block für die Hardware-Realisierung zugelassen, aber mit der Möglichkeit versehen, bei Auftreten der “illegalen” Operation zur Laufzeit diese nahtlos wieder in Software auszuführen [10]. Als Grundlage für diese Selektion dienen die Ergebnisse des dynamischen Profiling, daß neben den Ausführungszeiten auch die Ausführungsfrequenzen der Blöcke bestimmt.

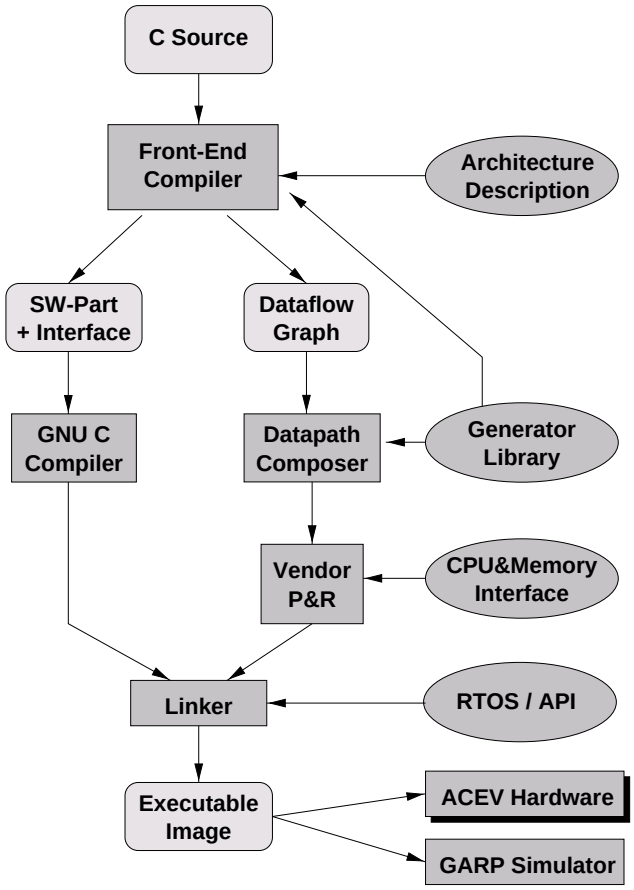

Abbildung 3: Nimble-Entwurfsfluß

Für die in Frage kommenden Blöcke werden nun verschiedene Hardware-Architekturen erzeugt (z.B. variiert durch unrolling, software pipelining etc.). So wird versucht, einen möglichst hohen Grad an Parallelität auf Instruktionsebene (ILP) [11] zu erreichen. Aus allen diesen Varianten werden nun diejenigen ausgewählt, deren abgeschätzter Bedarf an Hardware-Fläche noch in die zur Verfügung stehende freie Kapazität paßt. Für diese Abschätzung wird auf zielarchitektur-spezifische Angaben aus einer parametrisierten Modulbibliothek [12] [13] zurückgegriffen. Die konkrete Auswahl beruht auf expliziten Benutzerangaben

und wiederum auf den Profiling-Daten. Dabei wird neben dem Anteil des Kandidaten an der Ausführungszeit auch das dynamische Rekonfigurationsverhalten berücksichtigt. Durch letzteres wird das schnelle Wechseln zwischen mehreren benachbarten Blöcken ("configuration thrashing") vermieden [14].

Die letztendlich ausgewählten Architekturen werden dann als CDFG zur Hardware-Realisierung an den "Datapath Composer" übergeben. Dieser greift auf die Modulbibliothek zurück, um für jede der Datenpfadoperationen eine geeignete Instanz zu erzeugen (parametrisiert beispielsweise durch Bit-Breite und Datentypen der Operanden). Die erzeugten Module sind bereits in sich vorplaziert (RPM). Anschließend baut der Datapath Composer alle Instanzen zu einem ebenfalls vorplazierten kompletten Datenpfad zusammen. Die hersteller-spezifischen EDA-Werkzeuge (im Falle von Xilinx also das M3-Paket) fügen den Datenpfad dann mit dem festen CPU/Speicher-Interface-Makroblock zusammen und nehmen eine abschließende Verdrahtung vor.

Der als Software verbleibende Teil des ursprünglichen C-Programmes wird um automatisch generierte Schnittstellen zu den Hardware-Blöcken angereichert und auf konventionellem Wege übersetzt. Zu den Aufgaben dieser Schnittstellen gehören auch die Realisierung des nahtlosen Wechsels zwischen Hard- und Software-Blöcken. Dabei wird bei Beendigung eines Hardware-Blockes entsprechend seines Rückgabewertes (Interrupt-Quelle) als nächstes der passende Software-Zweig ausgeführt. Neben dem erfolgreichen Abschluß (Ausführung bis zum Ende des Blocks) kann so auch auf unterschiedliche Ausnahmezustände innerhalb des Hardware-Blocks reagiert werden, um diese dann in Software zu behandeln. In allen Fällen werden die innerhalb des Blocks veränderten Werte aus den Hardware-Registern wieder in ihre zugehörigen Software-Variablen zurückgeschrieben.

Alle Soft- und Hardware-Teile werden schließlich zusammen mit dem Betriebssystemkern [15] und den Laufzeitbibliotheken zu einem auf die Zielplattform ladbaren Programm zusammengebunden. Dabei werden auf der ACEV-Plattform die unhandlichen FPGA-Konfigurationsdateien (über 700KB je Konfiguration) komprimiert (auf einige Dutzend KB) und direkt in Standard-Objektmodule konvertiert. Auf diese Weise wird die häufig verwendete Kompilierung von nach C übersetzten Konfigurationsdateien vermieden.

4 Erfahrungen und Ergebnisse

Der Weg zur Realisierung eines durchgehenden Flusses von C zu einer adaptiven HW/SW-Lösung hat sich als steinig und mühsam erwiesen. Neben dem signifikanten Aufwand, der in die Entwicklung der Prototyping-Plattform ACEV investiert wurde, war dafür auch die Intention verantwortlich, so früh wie möglich komplette *reale* Anwendungen (SPEC, UCLA Mediabench, Honeywell Suite) zu untersuchen. Auch wurde die Vermutung bestätigt, daß FPGAs zwar als Rechenkomponenten verwendet werden können, ihre oft feinkörnige Struktur aber für arithmetische Operationen (8-32 Bit Breite) eher ineffizient ist. Ein praktischer Einsatz unserer Techniken scheint eher auf grobkörnigeren Architekturen mit sehr rascher Rekonfigurierbarkeit sinnvoll.

Anwendung	Anzahl von Schleifen und % Laufzeit			
	Gesamt	$t > 10\%$ Σt in %	$t > 1\%$ Σt in %	HW Σt in %
Skipjack	6	1 98.1	2 99.5	3 98.1
DES Keysearch	12	1 87.0	3 99.1	4 95.3
Wavelet Enc	25	5 69.8	12 98.8	21 98.9
Wavelet Dec	63	3 42.3	15 96.7	42 92.9
MPEG Enc	165	3 66.7	15 84.7	100 65.9
MPEG Dec	115	2 27.9	14 61.2	61 40.8
JPEG Dec	470	4 83.2	7 95.0	321 95.9
ADPCM 1	14	2 43.1	9 63.8	12 39.0
ADPCM 2	5	2 87.4	4 97.6	3 95.2
G.721 Dec	9	1 38.4	5 57.3	7 49.6
G.721 Enc	10	1 38.8	5 56.6	7 49.4

Tabelle 1: Beispielanwendungen

Die Tabelle 1 zeigt eine Analyse von verschiedenen Beispielen. Wie oben angedeutet handelt es sich dabei nicht um synthetische Benchmarks, sondern um reale Anwendungen. Zu jedem Programm ist dabei die Gesamtzahl der Schleifen, die Zahlen von Schleifen, die mehr als 10% bzw. mehr als 1% der Rechenzeit abdecken und die Zahl der durch Nimble in HW realisierbaren Schleifen angegeben. Die zweite Zeile gibt den Anteil der Laufzeit an, der jeweils in diesen Schleifen verbracht wird. Beispiel: Die Wavelet-Encoding-Anwendung hat insgesamt 25 Schleifen. Davon haben 5 Schleifen Einzelanteile an der Ausführungszeit von je $> 10\%$. Zusammen machen diese 5 Schleifen 69.8% der Gesamtausführungszeit aus. 12 Schleifen haben eine Einzelausführungszeit, die mehr als 1% der Gesamtlaufzeit ausmacht. Insgesamt verbringt die Anwendung 98.8% ihrer Ausführungszeit in diesen 12 Schleifen. Durch den Nimble-Compiler sind 21 der 25 Schleifen in Hardware realisierbar, ohne Rekonfigurationszeiten würde die Anwendung dann 98.9% ihrer Rechenzeit in Hardware verbringen.

Neben der bekannten Tatsache, daß in der Regel nur wenige Schleifen das Gros der Programmlaufzeit ausmachen, wird hier deutlich, daß sich mit dem in Nimble verwendeten Ansatz auch aus einer Sprache wie C ein nennenswerter

Teil der Anwendung (40%-99%) in HW übersetzen läßt.

Da der aktuelle Prototyp erst wenige Optimierungsschritte enthält, ist es noch zu früh, eine umfassende Beurteilung seiner Leistungsfähigkeit zu geben. An dieser Stelle seien nur zwei Zwischenergebnisse genannt: Die Wavelet-Bildkompression wird bei Auslagerung von nur 10 Schleifen in HW auf der GARP-Architektur um den Faktor 3.25 schneller als die reine SW-Lösung ausgeführt. Die DES-Schlüsselsuche ist ein Sonderfall. Hier läßt sich durch Verwendung von anwendungsspezifischen Modulgeneratoren, die mittels eines einfachen Funktionsaufrufs von C aus angesprochen werden, sogar auf der realen ACEV-Plattform eine Beschleunigung um den Faktor 137 (inklusive FPGA-Konfiguration) erreichen.

5 Ausblick

Der bisherige Stand von Entwurfsfluß und Hardware wird nun als Ausgangspunkt für weitere Optimierungen sowohl auf der Compiler- als auch auf der Back-End-Seite dienen. Dabei kann eine ganze Palette von Techniken aus den Bereichen der Optimierung für Vektor- und VLIW-Prozessoren ebenso wie aus dem klassischen VLSI-Entwurf (Synthese, physical design) zum Einsatz kommen: Die adaptiven Recheneinheiten können somit immer optimal an die Anforderungen des speziellen Problems angepaßt werden.

Literatur

[1] Santarini, M., “ASIPs: Get ready for reconfigurable silicon”, EE Times, 20 Nov 2000

[2] Chameleon Systems Inc., “CS2000 Reconfigurable Communications Processor”, `http://www.cmln.com/what/RCP_Product_Brief.pdf`, 2000

[3] PACT GmbH, “The eXtreme Processor Platform”, `http://www.pactcorp.de`, 2000

[4] Triscend Corp., “Triscend A7 CSoC Family”, `http://www.triscend.com/products/indexA7.html`, 2000

[5] Altera Corp., “ARM-based Embedded Processor Device Overview”, `http://www.altera.com/document/ds/ds_arm.pdf`, 2000

[6] Xilinx Corp., “Virtex 2.5V Field Programmable Gate Arrays”, `http://www.xilinx.com/partinfo/ds003.pdf`, 2000

[7] Koch, A., “A Comprehensive Prototyping Platform for Hardware-Software Codesign”, Workshop on Rapid Systems Prototyping, Paris, 2000

[8] Lange, H., Koch, A., “Memory Access Schemes for Configurable Processors”, Workshop on Field Programmable Logic 2000, Villach

[9] Stanford University, “The SUIF Compiler System”, `http://suif.stanford.edu`, 2000

[10] Callahan, T., Hauser, J., Wawrzynek, J., “The Garp Architecture and C Compiler”, IEEE Computer, 4/2000

[11] Hennessy, J., Patterson, D., “Computer Architecture: A Quantitative Approach”, section 4.7, Morgan-Kaufmann, 1996

[12] Koch, A., “Enabling Automatic Module Generation for FCCM Compilers”, IEEE Symp. on Field-Programmable Custom Computing Machines, 1999

[13] Koch, A., “Creation and Embedding of Complex Parameterized Hardware Objects”, 2nd Workshop on Engineering of Reconfigurable Hardware/Software Objects, 2000

[14] Li, Y.B., Harr, R., et al. “Hardware-Software Co-Design of Embedded Reconfigurable Architectures”, Design Automation Conference, 2000

[15] Rock, M., “Porting the RTEMS RTOS to the ACE adaptive computing environment”, Studienarbeit, Braunschweig, 2000