Einsatz programmierbarer Logik und digitaler Signalprozessoren in Forschung und Lehre

Eugen Brenner, Christian Kreiner, Claudia Mathis, Bernhard Rinner,
Martin Schmid, Reinhard Schneider, Christian Steger, Reinhold Weiss

Institut für Technische Informatik
Technische Universität Graz
{Nachname}@iti.tu-graz.ac.at
Inffeldgasse 16
8010 Graz

Zusammenfassung

In diesem Beitrag wird der Einsatz von programmierbarer Logik und digitalen Signalprozessoren (DSPs) am Institut für Technische Informatik in Forschung und Lehre beschrieben. Es werden kurz die Forschungsgebiete im Bereich eingebetteter Systeme erläutert: Hardware/Software Codesign, Echtzeit-KI-Architekturen und Rekonfigurierbare Rechnerarchitekturen. Danach werden realisierte Spezialrechner-Architekturen vorgestellt. Ziel aller Implementierungen ist der Entwurf von leistungsstarken Architekturen für den Einsatz in einer Echtzeitumgebung. Am Ende des Beitrages werden Lehrveranstaltungen im Bereich programmierbarer Logik (CPLDs) und DSPs vorgestellt.

Keywords: Multi-DSP, rekonfigurierbare Logik, Hardware/Software Codesign, Echtzeit-KI, Spezialrechner-Architekturen;

1 Einleitung

Das Institut für Technische Informatik befaßt sich schwerpunktmäßig mit verteilten Echtzeit-Rechnerarchitekturen, die in technische Prozesse *eingebettet* sind, diese also steuern, überwachen usw. Primäres Ziel ist es, leistungs- und zuverlässigkeitsorientierte Spezialrechner für industrielle Anwendungen zu entwerfen und aufzubauen. Zu diesen Spezialrechner-Architekturen zählen Multi-DSP-Systeme, die aus parallel arbeitsfähigen, digitalen Signalprozessoren (DSP) bestehen. In unseren Anwendungen werden meist DSPs vom Typ TMS320C40 oder TMS320C6x von Texas Instruments verwendet. Der Befehlssatz solcher Signalprozessoren stellt einen Querschnitt durch häufig benötigte Grundfunktionen der digitalen Signalverarbeitung dar, z.B. akkumulierende Multiplikation und zirkulare Adressierung. In vielen Anwendungen ist es sinnvoll, diesen Befehlssatz durch mächtige Spezialbefehle zu erweitern. Dies führt auf das Konzept des *Coprozessors*. Der Coprozessor (auf Basis von FPGAs) soll diese Spezialbefehle um ein bis zwei Größenordnungen schneller ausführen, als ein universellerer Signalprozessor.

2 Programmierbare Logik und DSPs in der Forschung

2.1 Forschungsschwerpunkte

2.1.1 Hardware/Software Codesign

Den Entwurfsvorgang heterogener Systeme (bestehend aus Hardware- und Software-Modulen) bezeichnet man als Hardware/Software Codesign. Diese Methode ermöglicht eine Beurteilung der gegenseitigen Beeinflussung (Realisierung einer Funktion in HW oder SW) im Entwurfsprozeß, eine kontinuierliche Verifikation über den gesamten Entwurfsprozeß und die Untersuchung des Entwurfsraums (Design-Space Exploration). Die einzelnen Entwurfsschritte werden durch eine Cosimulation [1] verifiziert. Schlüsselkomponente ist der Einsatz von verschiedenen Berechnungsmodellen (models of computations). Für die Simulation des Teilsystems wird immer der Simulator verwendet, welcher für das verwendete Berechnungsmodell am besten geeignet ist. Das Gesamtsystem wird durch die Interaktion zwischen diesen Einzelsimulatoren [2] simuliert.

2.1.2 Echtzeit-KI-Architekturen

Methoden aus dem Bereich der künstlichen Intelligenz (KI) werden zum Lösen von schwierigen Aufgaben, wie z.B. Sprachverarbeitung oder Diagnose [9], eingesetzt. Die KI-Methoden basieren meist auf komplexen Algorithmen mit unregelmäßiger Verarbeitungsabfolge. Die Systemgröße, unvollständige bzw. fehlerhafte Eingabedaten sowie beschränkte Ressourcen limitieren den Einsatzbereich dieser intelligenten Methoden.

Im Forschungsschwerpunkt *Echtzeit-KI-Architekturen* des Instituts werden die Einsatzgrenzen untersucht und spezielle Rechnerarchitekturen zur Leistungssteigerung der KI-Methoden entwickelt [7], [9]. Durch Parallelverarbeitung, hardwarenaher Implementierung sowie Anpassung der Rechnerarchitektur an das Verarbeitungsmodell der KI-Methode werden die Ausführungszeiten um Größenordnungen verringert und dadurch auch der Einsatz der intelligenten Verfahren in eingebetteten Systemen ermöglicht. Als Implementierungsplattformen werden Multi-DSP-Systeme und FPGAs verwendet.

2.1.3 Rekonfigurierbare Rechnerarchitekturen

Durch die Möglichkeit der Hardwareprogrammierbarkeit (z.B. von FPGAs) kann man nun einerseits Algorithmen parallelisieren und andererseits mehrere unterschiedliche, parallele Tasks zeitversetzt auf ein und demselben Chip implementieren. Grundsätzlich kann man drei Arten der Rekonfigurierbarkeit unterscheiden [6]: (i) Statische Rekonfiguration, (ii) Run-Time Rekonfiguration und (iii) Dynamische Rekonfiguration. Bei der **statischen Rekonfiguration** (compile-time reconfiguration CTR) wird das System nur einmal, vor Programmablauf konfiguriert.

Globale Run-Time Rekonfiguration (RTR) bietet die Möglichkeit, die FPGA-Architektur während der Ausführung einer Applikation neu zu konfigurieren. Damit verfügt man zeitversetzt über mehrere unterschiedliche Rechensysteme auf einem einzigen Chip.

Unter **lokaler Run-Time Rekonfiguration** versteht man das selektive Konfigurieren einzelner Teilsektionen der programmierbaren Logik eines FPGAs, während die restlichen Teile ohne Unterbrechung weiterarbeiten können. Dadurch muß der Programmablauf nicht angehalten werden, um eine geringfügige Rekonfiguration vorzunehmen. Die Konfiguration kleiner, selektiver Flächen eines FPGAs erfordert wesentlich weniger Konfigurationsbits und damit deutlich geringere Konfigurationszeiten.

2.2 Forschungsprojekte: Spezialrechner-Architekturen

Es folgt eine kurze Beschreibung von Forschungsprojekten des Instituts, in denen Spezialrechner-Architekturen mit programmierbaren Logikbausteinen und/oder Multi-DSP-Systemen entwickelt worden sind.

2.2.1 Coprozessor für schnelles Suchen

In Zusammenarbeit mit dem Europäischen Kernforschungszentrum CERN im Rahmen des Projektes CMS (Compact Muon Solenoid) wurde für einen Dual-Port-Pufferspeicher, auf den mit 100 Mbyte/s zugegriffen werden muß, um Detektordaten in Echtzeit zwischenzuspeichern, ein Hardware-Suchprozessor aufgebaut [4]. Primäre Aufgabe dieses Suchprozessors ist die Verwaltung von Speicherblöcken für die Einlese- und die Ausleseoperation, damit ein wahlfreier Zugriff auf jeden beliebigen Speicherblock schritthaltend möglich ist. Die technische Lösung für den Suchprozessor beruht auf einem statischen Hash-Algorithmus mit Verkettung der Überläufe und einer Organisation der Überläufe durch alphabetische Suchbäume. Ein besonderer Vorteil dieser Lösung ist, daß die Anzahl der Suchschritte im ungünstigsten Fall nur linear mit der Schlüsselmenge wächst.

2.2.2 Verteilte Rechnerarchitektur für schnelle qualitative Simulation

Für die modellbasierte Online-Diagnose von Prozeßautomatisierungssystemen spielt die qualitative Simulation eine immer wichtigere Rolle; ihr praktischer Einsatz in einer Echtzeit-Umgebung scheitert jedoch bislang an der hohen Rechenzeit.

In einer mehrjährigen Arbeit [7] wurde am Institut für Technische Informatik eine Spezialrechner-Architektur entwickelt, die die rechenzeitintensivsten Kernfunktionen des bekannten qualitativen Simulators QSIM um ein bis zwei Größenordnungen beschleunigt und dadurch die Laufzeit des gesamten Simulationsalgorithmus erheblich verkürzt.

Abbildung 1 zeigt das Blockschaltbild für den Spezialbefehl *qualitative Multiplikation*. Die Teilfunktionen des Spezialbefehls (SF1 ... SF3) werden parallel verarbeitet. Getrennte Controller für Ein- und Ausgabe ermöglichen einen erhöhten Datentransfer zwischen Host- und Coprozessor. Der Coprozessor wurde mit Hilfe von XC4013 FPGAs von Xilinx implementiert und erzielt eine Geschwindigkeitssteigerung von mehreren Größenordnungen im Vergleich zu Software-Implementierungen [7].

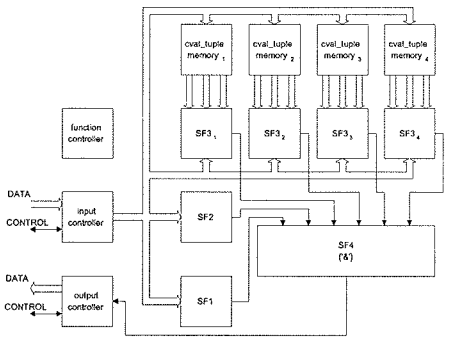

Abbildung 1: Blockschaltbild eines Coprozessors für die qualitative Multiplikation

2.2.3 Paralleler Prozessor für Simulated Annealing in Echtzeit-Systemen

Simulated Annealing ist ein bekannter Lösungsalgorithmus für kombinatorische Probleme. Der größte Nachteil dieses Algorithmus ist seine große Laufzeit. Das Ziel dieses Forschungsprojektes ist es, einen Spezialprozessor [8] (Blockdiagramm Abbildung 2) für Simulated Annealing zu entwickeln. Dieser Prozessor soll auch Zeitbedingungen erfüllen, um in Echtzeit-Systemen eingesetzt werden zu können. Die wichtigsten Eigenschaften dieses Prozessors sind: Parallelverarbeitung zur Leistungssteigerung, Unterbrechbarkeit mit der Möglichkeit, von der letztbesten Lösung wieder fortzufahren, sowie eine möglichst einfache Programmierung. Anwendungsgebiete dieses Prozessors sind: Online task-mapping, Hardware/Software Codesign und andere Optimierungsaufgaben.

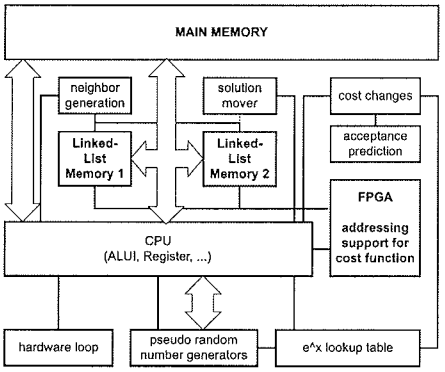

Abbildung 2: CPU Kern mit Erweiterungen für Optimierungsaufgaben.

2.2.4 Hardware-Scheduler für Echtzeit-Systeme

In Echtzeit-Systemen ist nicht nur die Richtigkeit eines berechneten Ergebnisses ausschlaggebend, sondern auch dessen rechtzeitige Fertigstellung. Daher ist neben der Rechenleistung die Vorhersagbarkeit über die rechtzeitige Abarbeitung der Task für ein Echtzeit-System von zentraler Bedeutung. Diese Problematik führte zur Entwicklung von Echtzeit-Scheduling Algorithmen (Rate Monotonic, Earliest Deadline First) auf der Task-Ebene. Auf der Hardware-Ebene stellt man jedoch fest, daß Standard-Prozessoren keine Unterstützung für Echtzeit-Scheduling bieten. Deshalb wird ein neuartiges Prozessorkonzept am Institut erarbeitet, welches die Echtzeitabarbeitung durch einen eingebetteten Hardware-Scheduler unterstützt [5]. Die wesentlichen Entwurfsziele dieses Prozessorkonzeptes sind *transparente Taskausführung*, *Taskswitching ohne Overhead*, gute *Skalierbarkeit* und eine möglichst *enge Kopplung zur CPU* um Kommunikations-Overhead zu vermeiden. Abbildung 3 zeigt das Blockschaltbild des neuartigen Prozessorkonzeptes.

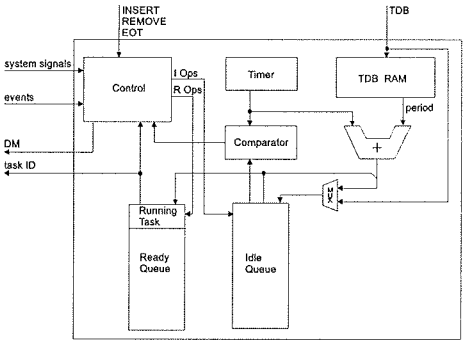

Abbildung 3: Konzept eines Schedulers in Hardware, der den Rate Monotonic Echtzeit-Scheduling Algorithmus unterstützt.

2.2.5 Spezialprozessor für die probabilistische Online-Diagnose

Dieses Projekt [3] behandelt den Entwurf, die Implementierung und die Evaluierung eines Spezialprozessors für die probabilistische Online-Diagnose eines technischen Prozesses. Der Spezialprozessor stellt eine neue Architektur zur intelligenten Prozeßdiagnose dar, welche harte Echtzeitanforderungen unter Verwendung von KI-Methoden erfüllt. Die Diagnosevorgänge werden vom KI-Subsystem des Spezialprozessors durchgeführt.

3 Programmierbare Logik und DSPs in der Lehre

3.1 Laborübung: Programmierbare Logik

Die CPLD-Programmierung ist ein Teil des Pflichtlabors Technische Informatik für Elektrotechnik-Studenten. Das Labor Technische Informatik dient zur Vertiefung der Inhalte der gleichnamigen Vorlesungen Technische Informatik (1 und 2), die das Schichtenmodell eines Computers vorstellt. Die CPLD-Programmierung repräsentiert die digitale Hardware-Ebene und hat als Teil eines größeren Labors einführenden Charakter. Nach der Eingabe werden die Schaltungen zuerst simuliert um deren Funktion zu überprüfen. Anschließend werden sie auf dem Zielbaustein - XC9572 CPLD abgebildet. Das CPLD wird *in circuit* programmiert und die entworfene Schaltung darauf demonstriert.

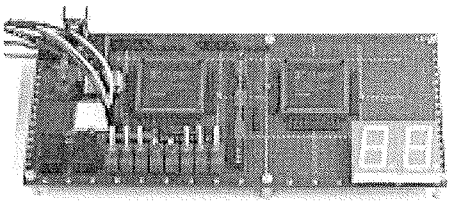

Abbildung 4: Testplatine zum Testen von Schaltungen für die Laborübung Programmierbare Logik.

Das Testboard (Abbildung 4) besteht aus 2 CPLDs (XC9572), eines für die Verwaltungsaufgaben (Generierung der Takte, Abtasten und Entprellen von Schaltern, Einlesen und Anzeigen von Signalen), das andere zur Programmierung durch den Benutzer. Der Benutzer hat somit ein programmierbares CPLD zur Verfügung, welches bereits in eine Testumgebung integriert ist.

Die Laborübung besteht aus drei Beispielen. Zur Lösung dieser Beispiele ist jeweils eine der drei folgenden Schaltungseingaben bzw. Entwurfsmethoden zu verwenden: Schema, Hardwarebeschreibungssprache (ABEL) oder Zustandsdiagramme (State Diagrams).

Beispiel 1 - Schema: In Beispiel 1 ist mittels Schema-Eingabe eine kombinatorische Schaltung zu entwerfen. Gegeben sind vier Städte, Verbindungswege zwischen den Städten und die Weglängen. Werden zwei Städte mit Schaltern ausgesucht, so soll der kürzeste Weg zwischen diesen beiden Städten als Ausgabe angezeigt werden (Jedem Verbindungsweg entspricht eine Leuchtdiode).

Beispiel 2 - Hardwarebeschreibungssprache: Beispiel 2 besteht ebenfalls aus einem kombinatorischen Schaltungsentwurf, der zum Unterschied zu Beispiel 1 mit der Hardwarebeschreibungssprache ABEL umgesetzt wird. Eine BCD-kodierte Zahl, die mit vier Schaltern eingestellt wird, ist auf der 7-Segmentanzeige darzustellen.

Beispiel 3 - State Diagram: In Beispiel 3 wird eine sequentielle Schaltung mit einem Moore bzw. Mealy Automaten entworfen, welcher dann direkt mit einem State Diagram umgesetzt wird. Als Eingabe ist ein Bitstrom gegeben (wird mit Schalter und Clock eingegeben), aus dem z.B. die Bitreihenfolge 011 zu detektieren ist.

3.2 Seminar: Rekonfigurierbare Rechnerarchitekturen

Die in Abschnitt 2 angesprochenen Forschungsarbeiten zum Thema *Rekonfigurierbare Rechnerarchitekturen* werden durch ein ganzjährig angebotenes Studentenseminar unterstützt, in dem Studentengruppen aktuelle Fallstudien zu theoretischen und/oder experimentellen FPGA-Architekturen detailliert studieren. Im Wintersemester 98/99 beispielsweise standen ausgewählte Rekonfigurierbare Rechnerarchitekturen für die Anwendungsgebiete *Computer Vision, Pattern Matching* und *Fehlertoleranz* im Mittelpunkt.

3.3 Hardwarebeschreibungssprachen

Am Institut für Technische Informatik wird die Lehrveranstaltung Hardwarebeschreibungssprachen [11] angeboten, die aus einem Vorlesungs- und einem Übungsteil besteht. Die Vorlesung behandelt die theoretischen Grundlagen von Hardwarebeschreibungssprachen (HDLs) und wählt als Beispiel die Standard-Industriesprache VHDL (VHSIC Hardware Description Language, VHSIC steht für Very High Speed Integrated Circuit) aus. Dabei werden Einsatzgebiete, Anforderungen an HDLs, Beispiele von HDLs und die Grundkonzepte von VHDL näher erläutert. Im Übungsteil lernen die Studenten mit Hilfe eines interaktiven Lernsystems die Syntax von VHDL kennen und können sofort eigene Beispiele programmieren, simulieren und analysieren. HDLs sind ein wesentliches Hilfsmittel in einem integrierten und automatisierten Entwurfsprozeß und finden Anwendung in der Spezifikation (Verhalten, Struktur,..), Dokumentation (Sprache, Interface-Spezifikation, Protokolle), Verifikation (Simulation) und Synthese (schrittweise Verfeinerung bis zur Zielarchitektur) von Systemen. Damit wird in der Ausbildung der Studenten der steigenden Komplexität eingebetteter Systeme Rechnung getragen.

3.4 Multi-DSP Labor

Ziel des Multi-DSP Labors [10] ist es, den Studenten sowohl Theorie als auch praktische Anwendungsmöglichkeiten von modernen DSP-Prozessoren zu vermitteln. Dieses Labor wird als Teil der Lehrveranstaltung *VLSI Prozessoren* angeboten und ist in vier Einzelübungen unterteilt. Die Übungen reichen von einfachen Audioanwendungen bis hin zu Anwendungen aus der parallelen Bildverarbeitung. Als Entwicklungsplattformen stehen TMS320C30 und TMS320C40 DSP-Prozessoren von Texas Instruments samt ihrer Entwicklungsumgebungen zur Verfügung.

Dieses Labor ist für Studenten der Elektrotechnik und Telematik im zweiten Studienabschnitt vorgesehen. Diese Studenten haben bereits ein grundlegendes Wissen über digitale Signalverarbeitung sowie Programmiererfahrung in Assembler und C.

Übung #1 dient zur Einführung in DSP-Prozessoren. Die Studenten experimentieren mit einfachen Audioanwendungen, wie z.B. Signalverstärkung und Filterung. Als Implementierungsplattform wird das DSP Starter Kit C3x (DSK) und eine einfache Audio-Box mit Signalgenerator und Lautsprecher verwendet. Die DSP-Algorithmen werden in Assembler programmiert. DSK und Audio-Box werden den Studenten zur Verfügung gestellt, um ein selbständiges Experimentieren mit einem DSP-Prozessor zu ermöglichen.

Übung #2 verwendet das TMS320C30 Evaluation Module (EVM) und den Code Composer von Go-DSP als Programmierumgebung. In dieser Übung wird die Nutzung der DSP-Spezifika, wie beispielsweise zirkulare Adressierung, verzögerte Sprünge oder Hardware-Schleifen, mittels Assemblerprogrammierung demonstriert. Es werden typische DSP-Algorithmen, wie FIR- und IIR-Filter optimiert und mit compiler-generiertem Code verglichen.

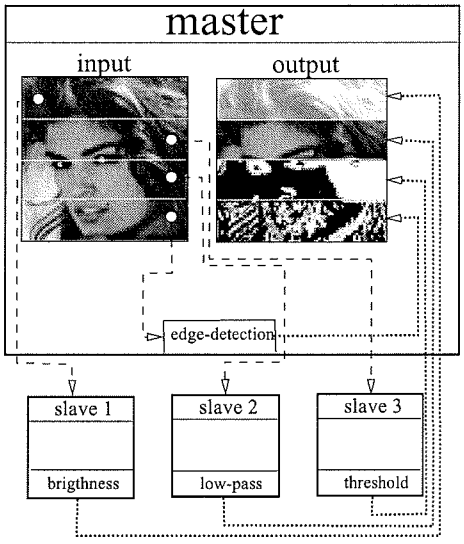

Abbildung 5: Parallele Bildverarbeitung in Übung #4.

Übung #3 behandelt komplexere DSP-Anwendungen aus dem Bereich der Bildverarbeitung. Die Bildverarbeitungsalgorithmen werden in C auf dem TMS320C40 Parallel Programming System (PPDS) implementiert. Es werden einfache Pixel-Algorithmen, wie Helligkeits- und Schwellwertfunktion, als auch komplexere Box-Filter entwickelt. Mit Hilfe von unterschiedlichen Box-Matrizen werden verschiedene Filtercharakteristiken realisiert.

Übung #4 führt in die Parallelverarbeitung mit DSP-Prozessoren ein. Es werden dazu die Bildverarbeitungsalgorithmen von Übung #3 parallelisiert und auf dem PPDS mit Hilfe des Code Com-

posers implementiert. Der Box-Filter wird anhand einer Master/Slave Struktur auf n Prozessoren parallelisiert. Der Master-Prozessor teilt das Bild in n Partitionen, überträgt $n-1$ Partitionen an die Slave Prozessoren und filtert die n-te Partition selbst. Anschließend empfängt der Master die gefilterten Partitionen von den Slaves und vereinigt alle Partitionen. Abbildung 5 zeigt die Master/Slave Struktur für vier Prozessoren (mit unterschiedlichen Filtermatrizen).

4 Schlußbetrachtung

In diesem Beitrag wurden die Forschungs- und Lehraktivitäten des Instituts für Technische Informatik auf dem Gebiet der programmierbaren Logik und der digitalen Signalprozessoren vorgestellt. Dieser Bereich unterliegt einem stetigen Wandel und einem rasanten (kommerziellen) Wachstum. Aktuelle Trends müssen frühzeitig erkannt und möglichst rasch in der Lehre vermittelt werden.

Literatur

[1] Christian Kreiner, Christian Steger, and Reinhold Weiss. A Hardware/Software Cosimulation Environment for DSP Applications. In *Euromicro Workshop on Digital System Design (EUROMICRO'99)*, Milan, Italy, September 1999.

[2] Christian Kreiner, Christian Steger, and Reinhold Weiss. Combination of Different Models of Computation for Cosimulation of Heterogeneous Systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, USA, June 1999.

[3] Johannes Lauber, Christian Steger, and Reinhold Weiss. Applied Probabilistic AI for Online Diagnosis of a Safety-Critical System Based on a Quality Assurance Problem. In *Proceedings of the 14th ACM Symposium on Applied Computing (ACM-SAC)*, San Antonio, TX, February 1999.

[4] Christian Mandl, Eugen Brenner, and Adolfo Fucci. Echtzeit-Suchprozessor-Architekturen. *e&i Elektrotechnik und Informationstechnik, ÖVE Verbandszeitschrift*, 115(3):137–143, 1998. special issue on Technische Informatik.

[5] Claudia Mathis and Reinhold Weiss. A Real-Time Hardware Scheduler Embedded in a Processor Core. In *14th International Conference on Computers and Their Applications*, pages 374–377, Cancun, Mexico, April 7-9 1999. ISCA.

[6] Marco Platzner. Reconfigurable computer architectures. *e&i Elektrotechnik und Informationstechnik, ÖVE Verbandszeitschrift*, 115(3):143–148, 1998. special issue on Technische Informatik.

[7] Marco Platzner, Bernhard Rinner, and Reinhold Weiss. A Computer Architecture to Support Qualitative Simulation in Industrial Applications. *e&i Elektrotechnik und Informationstechnik, ÖVE Verbandszeitschrift*, 114(1):13–18, 1997. special issue on Technische Informatik.

[8] Reinhard Schneider and Reinhold Weiss and Martin Schmid. Parallel Simulated Annealing on a Flexible Multi-DSP System. In *Proceedings of the International Conference on Signal Processing Applications & Technologies (ICSPAT)*, volume 2, pages 1135–1140. Miller Freeman, September 13-16 1998.

[9] Bernhard Rinner and Benjamin Kuipers. Monitoring Piecewise Continuous Behaviors by Refining Semi-Quantitative Trackers. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1080–1086, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[10] Bernhard Rinner, Reinhard Schneider, Christian Steger, and Reinhold Weiss. A Multi-DSP Laboratory Course. In *The Second European DSP Education and Research Conference*, pages 350–356, Paris, 1998. Texas Instruments.

[11] Christian Steger. Einführung in die Hardwarebeschreibungssprache VHDL mit Hilfe eines interaktiven Lernprogramms. In *Interaktives computerunterstütztes Lernen, ICL98*, pages 141–148, Villach, October 1998.