2.2 Eingebettete Systeme
Rolf Ernst; Universität Braunschweig

Institut für Datentechnik und Kommunikationsnetze Technische Universität Braunschweigr.ernst@tu-bs.de

Zusammenfassung

Entwurf und Architektur eingebetteter Systeme sind ein wissenschaftlich herausforderndes und industriell hoch relevantes Gebiet der Technischen Informatik. Dieser Beitrag gibt einen Einblick in die Aufgabenstellung, die wissenschaftliche Methodik und die Entwicklungsperspektiven.

2.2.1 Aufgabenstellung

Der Entwurf und die Architektur eingebetteter Systeme sind ein neues fachübergreifendes Schwerpunktthema der Technischen Informatik, der Elektrotechnik und des Maschinenbaus. Unter einem **eingebetteten System** versteht man dabei ein (Mikro-)Computersystem, das in ein technisches System „eingebettet" ist, das selbst nicht als Computer erscheint. Beispiele hierfür sind Vermittlungssysteme oder Endgeräte in der Telekommunikation, Geräte der Unterhaltungselektronik, sowie verteilte Systeme in der Automobilelektronik, der Automatisierungstechnik oder der Luft- und Raumfahrtechnik. Es gibt kaum noch moderne Industrieprodukte, die ohne diese Mikrocomputer und ihre Software noch konkurrenzfähig realisierbar wären.

Umgekehrt haben eingebettete Systeme einen großen Einfluss auf die Halbleiterindustrie, vor allem in Europa und speziell in Deutschland. So entfielen in Deutschland im Jahr 2001 lediglich 22% des Umsatzes auf Schaltungen für die Datentechnik [ZVE01]. Die verbleibenden 78% teilten sich die Kfz-Elektronik mit 28%, die Industrieelektronik mit 20%, die Telekommunikation mit 24% und die Konsumelektronik mit 6% auf.

Der Entwurf und die Hardware- und Software-Architekturen eingebetteter Systeme unterscheiden sich wesentlich von denen der klassischen Datentechnik. Während es noch bis in die 90er Jahre üblich war, ältere und damit kostengünstige Prozessoren aus der Datentechnik weitgehend unverändert zu übernehmen, lässt sich bei den eingebetteten Systemen heute eine Diversifizierung in eine Vielzahl von spezialisierten Architekturen beobachten, von (4 und) 8 bit Mikrocontrollern, die noch immer einen großen Marktanteil besitzen, bis zu digitalen Signalprozessoren (DSP), die hohe Rechenleistung bei geringen Kosten und geringer Verlustleistung bieten. Auch Vorgaben der Softwareentwicklung beeinflussen die Auswahl, etwa die Forderung nach Binärkompatibilität zum x86- bzw. IA (Intel Architecture)-Befehlssatz. Ähnliche Entwicklungen sind im Bereich der Kommunikation in verteilten eingebetteten Systemen zu beobachten. Der **Entwurfsraum**, d.h. die Menge der möglichen Lösungen, ist also viel größer als im klassischen Bereich der Workstations oder PCs. Komplexere eingebettete Systeme bilden meist heterogene Architekturen aus Mikrocontrollern, DSPs, Coprozessoren, analogen und digitalen kundenspezifischen Schaltungen und Speichern. Bereits heute werden heterogene Multiprozessoren mit unterschiedlichen Busstrukturen und Speicherarchitekturen auf einer VLSI-Schaltung zu sogenannten Multiprocessor Systems-on-Chip (MpSoC) integriert. Ein Beispiel hierfür ist der Viper von Philips, ein MpSoC für den Einsatz in Settop-Boxen [DJR01]. Abbildung 1 zeigt eine schematische Darstellung dieser Schaltung. Sie kombiniert zwei Typen von Prozessoren, einen VLIW-Prozessor (Very Long Instruction Word) für die Bild- und Videosignalverarbeitung und einen MIPS RISC-Prozessor für Steuer- und allgemeine Aufgaben. Beide sind mit getrennten Cache-Speichern für Daten und Programme ausgerüstet. Diese frei programmierbaren Prozessoren werden durch Coprozessoren ergänzt, von denen einige mehrfach instantiiert wurden. Die Prozessoren haben teils feste Funktionen, wie eine Videoformat-Konvertierung oder sie sind „schwach" programmierbar, d.h. sie verfügen über einen Satz von konfigurierbaren Funktionsmakros, z.B. für den Transport von Videoströmen oder für die Generierung von Graphiken. Der Viper verwendet mehrere Typen von Bussen unterschiedlicher Bandbreite, um diese Komponenten zu verbinden und um einen externen größeren Speicher, ein SDRAM, anzuschließen. Die Basissoftware für diesen MpSoC umfasst unterschiedliche Betriebssysteme, sowie ein Application Programmer Interface (API), das unter anderem Softwarekomponenten für die Bildverarbeitung und Softwarecodecs enthält.

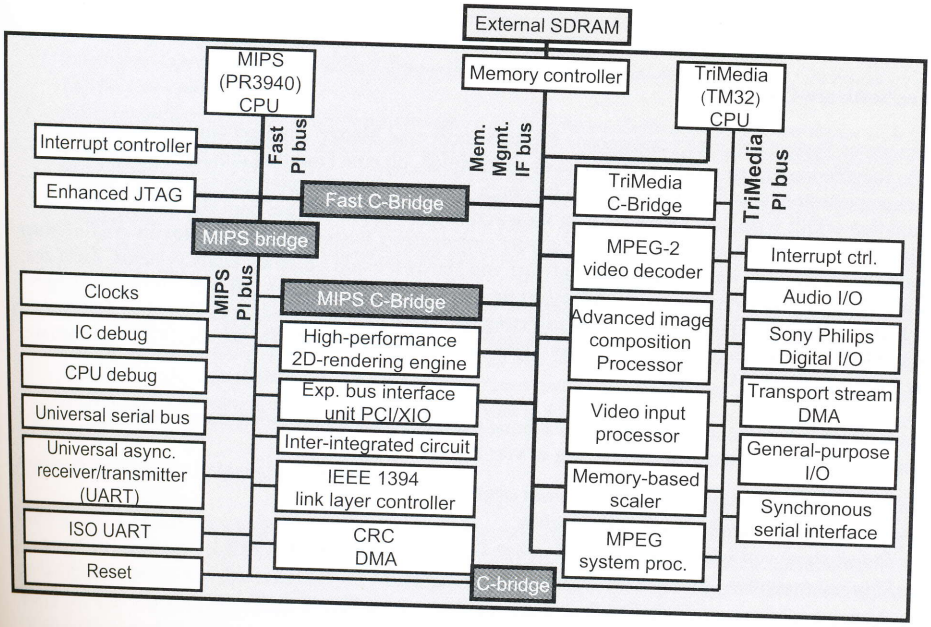

Abb. 1: Multiprocessor System-On-Chip: Philips VIPER [DJR01]

Für diese Entwicklung hin zu komplexen Architekturen gibt es zwei Gründe. Auf der einen Seite steht der hohe Kostendruck und eine Vielzahl von Randbedingungen für eingebettete Systeme, wie Zeitbedingungen, geringe Verlustleistung, erhöhte Sicherheitsanforderungen oder Begrenzung der elektromagnetischen Abstrahlung. Eine besondere Bedeutung kommt dabei einer extrem energieeffizienten Verarbeitung als neuem Optimierungskriterium, vor allem bei mobilen Systemen, zu.

Auf der anderen Seite ermöglicht die vorgegebene Aufgabe eines eingebetteten Systems eine **Spezialisierung** der Hardware. Dabei unterscheiden sich die Aufgaben wesentlich von denen der PC- und Workstation-Welt, bei der in erster Linie höchster Durchsatz von Rechenprozessen gefordert wird. Betrachten wir etwa ein Kraftfahrzeug, so gibt es hier einerseits Steuerungsaufgaben, bei denen das eingebettete System auf Umgebungssignale, etwa einen Knopfdruck, *reagiert* – dies wären **reaktive** Systemfunktionen – und andererseits Berechnungen, die periodisch auszuführen sind, wie ein digitales Filter – diese würde man als **transformierende** Funktionen bezeichnen. Die Spezialisierung ermöglicht in der Praxis Schaltungen mit einem Vielfachen der Verarbeitungsleistung von PC-Prozessoren bei deutlich geringeren Kosten und Verlustleistung (z.B. [DJR01]). Spezialisierung stellt damit ein sehr interessantes Optimierungsproblem dar [Ern99a].

Zwei Entwicklungstrends eingebetteter Systeme zeichnen sich ab. Zum einen werden anhaltende Fortschritte der Halbleitertechnologie weitaus **komplexere integrierte Schaltungen** ermöglichen. Die allgemein als Referenz verwendete Prognose ITRS (International Technology Roadmap for Semiconductors [ITR01]) sagt für das Jahr 2016 kostengünstige integrierte Schaltungen mit 3,2 G Transistorfunktionen voraus.

Der zweite Trend ist der Einsatz eng **kooperierender verteilter Systeme**. Diese Entwicklung findet sich in drahtgebunden vernetzten Systemen, wie im Automobil, wo vor allem die Vernetzung sicherheitskritischer Drive-by-wire-Funktionen hohe Anforderungen an echtzeitige Kommunikation stellen, wie auch in den drahtlosen Anwendungen der Mobilkommunikation des Wearable Computing oder der Ambient Intelligence.

Die beiden Trends sind nicht unabhängig zu verstehen, vielmehr wächst meist, wie etwa in der Automobil- und Verkehrstechnik, sowohl die Komplexität der einzelnen Schaltung als auch der Grad ihrer Vernetzung.

Die zunehmende Komplexität der Anwendungen wie auch der Architekturen erfordert eine drastische Steigerung der Entwurfsproduktivität, die nur über Wiederverwendung immer größerer Systemteile zu erreichen ist [ITR01].

2.2.2 *Forschung*

Hardware/Software-Coentwurf

Ein eingebettetes System soll eine vorgegebene Funktion oder eine Menge vorgegebener Funktionen unter gegebenen Randbedingungen erfüllen. Dabei ist es unerheblich, ob eine Funktion durch programmierte Hardwarekomponenten, also durch Software, oder durch Hardwarekomponenten mit fester oder parametrisierter Funktion erfüllt wird. Im Gegensatz zum klassischen Systementwurf, bei dem Hardware- und Softwareentwurf streng getrennt werden, wird im Entwurf eingebetteter Systeme daher ein **Hardware/Software-Coentwurf** favorisiert, d.h. der gemeinsame Entwurf von Hardware und Software eines Systems. Ziele des Hardware/Software-Coentwurfs sind die **Optimierung des Entwurfsprozesses** mit dem Ziel einer Steigerung der **Entwurfsproduktivität** und die **Optimierung des Entwurfs** selbst mit dem Ziel einer verbesserten **Produktqualität**. Eine tiefergehende Diskussion findet sich etwa in [Ern99b, Ern98, DeG97], eine Sammlung von Papern in [DEW01].

Der in etwa einem Jahrzehnt internationaler Forschung herausgebildete Forschungsansatz beruht auf einer Zerlegung der sehr komplexen Problemstellung in vier Phasen:

- *einer Analyse des Entwurfsprozesses, daraus abgeleitet*
- *eine Zerlegung in handhabbare Teilaufgaben,*
- *der Entwicklung von Lösungsansätzen für diese Teilaufgaben und schließlich*
- *der Untersuchung der Lösungen im Kontext des Entwurfs.*

Bei den Entwurfsprozessen unterscheidet man einen iterativen Hardware/Software-Coentwurf, der von einer gegebenen Systemspezifikation ausgehend über Implementierung und Analyse des Ergebnisses hinsichtlich Randbedingungen und Optimalität iteriert, bis eine tragfähige Lösung gefunden wird, und einen Plattform-Entwurfsprozess, der sich am klassischen quantitativen Rechnerentwurf orientiert [HeP03], und mit einem Benchmark von Beispielprogrammen als Spezifikation arbeitet. Letzterer Ansatz wird gelegentlich auch als „Application-Architecture-Co-Design" bezeichnet.

Als Beispiel soll der Entwurf von Mikrocontrollern und Hardwarebeschleunigern näher betrachtet werden. In den Arbeiten in [GCD92, EHB+96, Mad97, BFS96] wird ein System sequentieller Prozesse auf eine Hardwarearchitektur, bestehend aus einem einzelnen Prozessor, mehreren Coprozessoren, einem Daten- und einem Programmspeicher, automatisch abgebildet. Die Kommunikation zwischen den Komponenten erfolgt über den gemeinsamen Speicher. Diese Hardwarearchitektur ist sowohl typisch für kleine eingebettete Mikrocontroller, die auch heute noch den überwiegenden Teil der eingebetteten Systeme stellen, als auch für Hardwarebeschleuniger, d.h. Architekturen, bei denen ein rechenzeitintensiver Teil einer Anwendung auf einen Coprozessor ausgelagert wird. Aus dieser Abbildung wird eine **Optimierungsaufgabe** abgeleitet. Optimierungsziel ist die Einhaltung von Laufzeitbedingungen unter minimalen Hardwarekosten (Mikrocontroller) oder die Minimierung der Laufzeit unter maximalen Hardwarekosten (Hardwarebeschleuniger). Die Abbildung eines Prozesssystems oder eines einzelnen Programms auf Hardware- und Software-Komponenten wird als **Hardware/Software-Partitionierung** bezeichnet.

Insgesamt ergeben sich die folgenden neuen Teilprobleme:

- *Zerlegung eines sequentiellen Prozesses in kommunizierende Teilprozesse zur Abbildung auf Prozessor bzw. Coprozessor.*
  Dies ist ein Problem, dass sich gut mit Datenflussanalyse erschließen lässt.
- *Analyse/Schätzung der Ergebnisse von manuellem Hardwareentwurf bzw. High-Level-Synthese der Hardware, der Compilierung und des Kommunikationsaufwands.*
  Hier kommen typisch Graphalgorithmen zum Einsatz. Bekannte Lösungen sind Listscheduling und pfadbasierte Schätzung.
- *Optimierende Hardware/Software-Partitionierung.*
  Hier werden heuristische Verfahren eingesetzt, ebenso wie allgemein einsetzbare Verfahren, wie etwa Simulated Annealing, Tabu Search oder Genetische Algorithmen.
- *Kommunikationssynthese.*
  Die Kommunikation zwischen den Prozessen muss auf elementare Kommunikationsprotokolle und Arbitrierungsverfahren umgesetzt werden, wobei minimaler Overhead im Vordergrund steht.

- *Compilierung für Spezialprozessoren und Synthese von Hardware.*
  Die Codegenerierung im Compilerbau hat durch Spezialprozessoren, wie DSP und schwach programmierbare Coprozessoren [Ern99a] eine regelrechte Renaissance erfahren. Eine Übersicht ist [MaG95, LeM97] zu entnehmen.
- *Eine Sammlung zentraler Veröffentlichungen zum Hardware/Software-Coentwurf findet sich in [DEW01]. Auf diesem Gebiet hat es gerade aus Europa sehr einflussreiche Forschungsbeiträge gegeben [Mad97, EHB+96, BFS96]. Anwendungen finden sich heute vor allem im Bereich der anwendungsspezifischen Prozessoren und der rekonfigurierbaren Architekturen. Beispiele für erfolgreiche kommerzielle Tools sind Xtensa [Ten02] oder VAARC [Pro02].*
- *Bei diesen frühen Forschungsansätzen hat man die Automation sehr in den Vordergrund gestellt und sich daher auf einfache Architekturen beschränkt. Wie wir gesehen haben, haben die technische Entwicklung, vor allem aber auch die Anwendungen mit ihren Spezialisierungsmöglichkeiten, sehr komplexe heterogene Hardwarearchitekturen und entsprechend komplexe Softwarearchitekturen hervorgebracht, deren Entwurf sich nicht so ohne weiteres automatisieren lässt. Bei heterogenen Architekturen stehen eher die Integration und die Verifikation im Vordergrund, während die bislang erforschten Coentwurfsmethoden dort vor allem für spezialisierte Einzelkomponenten und Subsysteme zum Einsatz kommen können.*

Entwurf komplexer heterogener Systeme

Heterogenität finden wir zunächst bei den Spezifikationssprachen. Für die Optimierung reaktiver und transformierender Systeme eignen sich grundsätzlich unterschiedliche Modellsemantiken („Models of Computation"). Systeme paralleler (hierarchischer) Automaten eignen sich besonders gut zur Beschreibung von reaktiven Systemen, aber auch zu ihrer Verifikation und Optimierung. Die Optimierung erfolgt durch Minimierung, Verschmelzung und Zerlegung von Zustandsräumen. Für die Modellierung transformierender Systeme haben sich unter anderem Kahn-Graphen bewährt, die eine flexible Optimierung der Ausführungsreihenfolge der Verarbeitungsschritte zur Speicheroptimierung, Parallelisierung oder zu Optimierung der numerischen Eigenschaften eines Systems erlauben. Die Kombination dieser Sprachen ist problematisch, da ihre Berechungsmodelle in der Regel nicht kompatibel sind. In der Praxis verlässt man sich auf die Simulation, wobei die in unterschiedlichen Sprachen beschriebenen Subsysteme zunächst individuell optimiert werden können, bevor sie über eine Simulations-Backplane, die ein Protokoll zwischen Modellen bereit stellt, gemeinsam simuliert werden. Grundlegende Arbeiten zu dieser Simulationstechnik sind in [BHL+94] veröffentlicht. Hat man erst ein solches einfaches Protokoll definiert, kann man wiederum bekannte Techniken der Hardware- und Softwaresynthese anwenden, um Hardwareschnittstellen und Softwaretreiber zu erzeugen [ChO+95, ChH+98, ChB00, VVB+96, CoW02, DFB+97, YNG+02].

Dieser „flachen" Integration fehlt die Möglichkeit zur globalen Analyse und Optimierung. Globale Optimierung und Analyse sind z.B. erforderlich, um globale End-zu-End-Zeitbedingungen berücksichtigen zu können, oder um eine subsystemübergreifende Optimierung des Kommunikationssystems und der Speichergrößen zu ermöglichen. Eine Alternative liegt in kompositionellen Ansätzen, in denen eine übergreifende Semantik definiert wird. Dabei muss es möglich sein, in dieser Semantik effizient zu optimieren. Hier gibt es verschiedene Ansätze, die sich auf hierarchische Semantiken stützen, etwa in PTOLEMY II [GLL99] oder dem kommerziellen Tool CoCentric System Studio [Syn02], oder die mit abstrakten Intervallen arbeiten, etwa FunState oder SPI [ZRE+03]. Eine Übersicht zu diesen Verfahren ist [ErJ00] zu entnehmen.

Heterogene Zielarchitekturen

Die Heterogenität der Hardware/Software-Zielarchitekturen lässt die Integration zu einem zentralen Problem werden. In diesem Kapitel wollen wir uns auf die Analyse der Zielarchitektur als zentrale Problemstellung und als Beispiel für den Einsatz formaler Methoden konzentrieren.

Bei der Verifikation eines eingebetteten Systems kann man grundsätzlich unterscheiden zwischen dem Nachweis der korrekten Umsetzung einer spezifizierten Funktion und dem Nachweis einer ausreichenden Performanz der Zielarchitektur, u.a. einer ausreichenden Prozessor- und Kommunikationsleistung, der Einhaltung von Zeitbedingungen, und einer ausreichenden Speicherkapazität. Bei heterogenen MpSoC und verteilten Systemen ist der Performanznachweis besonders anspruchsvoll, weil unterschiedliche Komponententypen und Schedulingstrategien in einem System gemischt werden. Die hohen Anforderungen an die Optimalität erfordern individuell angepasste Strategien. Signalverarbeitungsalgorithmen mit festen, zeitinvarianten Verarbeitungsschritten werden typisch in einer festen Ausführungsreihenfolge periodisch wiederholt, daneben gibt es Zeitscheibenverfahren mit festen oder variablen Zeitschlitzen für die Entkopplung von Teilaufgaben oder Anwendungen mit Best-Effort-Anforderungen und prioritätsbasiertes Scheduling mit statischer oder dynamischer Prioritätsvergabe. Mehrere dieser Strategien können in einem System kombiniert auftreten. Dabei werden diese Schedulingstrategien in gleicher Weise auf Prozessoren wie auf die Kommunikation zwischen Komponenten angewandt.

Aus der Kombination entstehen in einem System komplexe zeitliche Abhängigkeiten, die nicht in der Funktion der Anwendung begründet sind. Abb. 2 soll dies veranschaulichen. Ein Subsystem S1, bestehend aus einem Standard-RISC-Prozessor, der über einen Speicher mit einem digitalen Signalprozessor (DSP) gekoppelt ist, greift über einen Bus auf einen Coprozessor zu. Parallel dazu kommuniziert in einem zweiten Subsystem ein leistungsfähiger VLIW-Prozessor (Very Long Instruction Word), beispielsweise für die Bildverarbeitung, mit einer Peripheriekomponente über diesen Bus. Abhängig von der Schedulingstrategie dieses Busses werden sich die beiden Subsysteme zeitlich beeinflussen. Paradoxerweise kann das zeitlich günstigste Verhalten des einen Subsystems zur größten Verlangsamung des anderen führen [Ern03]. Derartige Abhängigkeiten sind kritisch für die Systemvalidierung, denn sie können zu transienten Überlastsituationen und/oder Pufferüberläufen führen.

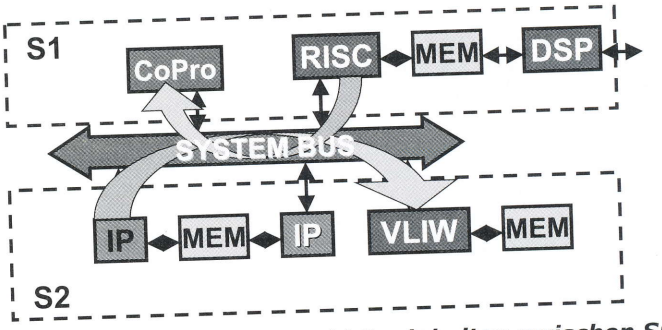

Abb. 2: Komplexe, nichtfunktionale Abhängigkeiten zwischen Subsystemen

In der industriellen Praxis werden die Subsysteme heute typisch getrennt entworfen und nach ihrer Integration mit Hilfe von Simulation bzw. Prototyping gemeinsam verifiziert. Probleme dieses Vorgehens sind die Identifikation von kritischen Fällen, die Behandlung komplexer Ereignisfolgen, die Einbindung teilweise spezifizierter Komponenten, beispielsweise von Legacy-Software und die Behandlung von Produktvarianten, vor allem im Automobilbau. Eine ausreichende Abdeckung durch Simulation ist mit sehr hohen Kosten verbunden.

Alternativ wurden in den letzten Jahren Verfahren für eine subsystemübergreifende formale Analyse vorgeschlagen als Erweiterung von klassischen Verfahren der Echtzeit-Datenverarbeitung, die sich auf einheitliche Schedulingstrategien konzentriert hat. Diese Verfahren bieten, bei Aufbau einer entsprechenden Infrastruktur von formalen Modellen für die Hardware- und Softwarekomponenten, die Chance auf eine formal vollständige Bestimmung des zeitlichen Verhaltens eines eingebetteten Systems bei weit kürzeren Rechenzeiten als im Fall der heute üblichen Simulation. Eine Übersicht findet sich in [Ern03]

2.2.3 *Perspektiven*

Der Entwurf eingebetteter Systeme ist gekennzeichnet durch komplexe Entwurfsflüsse, in denen eine Vielzahl unterschiedlicher Modellierungssprachen und Entwurfswerkzeuge verwendet werden. Heterogene Hardware-/Software-Architekturen als Ergebnis von Systemoptimierung und Wiederverwendung führen zu einer weiteren Erhöhung der Entwurfskomplexität. Ein wichtiger Schritt zur wissenschaftlichen Aufarbeitung ist die Zerlegung des Entwurfs in formal fassbare Teilprobleme und ihre Zurückführung auf bekannte Probleme der Informatik, wie sie im neuen Gebiet des Hardware-/Software-Codesign verfolgt wird. Auf diese Weise kann die Informatik einen zentralen Beitrag zum wirtschaftlich außerordentlich wichtigen Gebiet der eingebetteten Systeme leisten.

Literatur

[BFS96] A. Balboni, W. Fornaciari, D. Sciuto. *Co-synthesis and Co-simulation of control dominated embedded systems.* Design Automation for Embedded Systems, Kluwer, vol. 1, no. 3, pp. 257-289, July 1996.

[BHL+94] J. Buck, S. Ha, E.A. Lee, D.G. Messerschmidt. *Ptolemy: A framework for simulating and prototyping heterogeneous systems.* International Journal of Computer Simulation, Swets&Zeitlinger Publishers, vol. 4, pp. 155 – 182, April 1994.

[ChB00] P. Chou, G. Borriello. *Synthesis and optimization of coordination controllers for distributed embedded systems.* Proceedings Design Automation Conference, pp. 410-415, June 2000.

[ChH+98] P. Chou, K. Hines, K. Partridge, G. Borriello. *Control generation for embedded systems based on composition of modal processes.* Proceedings ICCAD 98, pp. 46 – 53, 1998.

[ChO+95] P. Chou, R. B. Ortega, G. Borriello. *Interface co-synthesis techniques for embedded systems.* Proceedings ICCAD 95, pp. 280 – 287, 1995.

[CoW02] CoWare. *Napkin-to-Chip.* www.coware.com

[DeG97] G. DeMicheli, R.K. Gupta. *Hardware/software co-design.* Proceedings of the IEEE. Vol. 85, No. 3, March 97, pp. 349-365.

[DEW01] G. DeMicheli, R. Ernst, W. Wolf. *Readings in hardware/software co-design.* Morgan Kaufmann, 2001.

[DFB+97] J.M. Daveau, G. Frenades Marchioro, T. Ben-Ismail, A.A. Jerraya. *Protocol selection and interface generation for hardware/software codesign.* IEEE Transactions on VLSI Systems, vol. 5, no. 1, pp. 136 – 144, March 1997.

[DJR01] S. Dutta, R. Jensen, A. Rieckmann. *Viper: A multiprocessor SOC for advanced set-top box and digital TV systems.* IEEE Design&Test of Computers, pp. 21-31, Sep.-Oct. 2001.

[EHB+96] R. Ernst, J. Henkel, Th. Benner, W. Ye, U. Holtmann, D. Herrmann, M. Trawny. *The COSYMA environment for hardware/software cosynthesis of small embedded systems.* Journal of Microprocessors and Microsystems, Butterworth-Heinemann, vol. 20, pp. 159-166, 1996.

[ErJ00] R. Ernst, A.A. Jerraya. *Embedded system design with multiple languages.* Proceedings ASP-DAC 2000, Yokohama, Januar 2000.

[Ern03] R. Ernst. *MPSOC performance modeling and analysis.* To appear in: Jerraya, Wolf, ed., Application-Specific Multi-Processor SoC, Kluwer, 2003.

[Ern98] R. Ernst. *Co-design of embedded systems - Status and trends.* IEEE Design&Test of Computers, pp. 45-54, April 1998.

[Ern99a] R. Ernst. *Embedded system architectures.* In: System Level Synthesis, NATO Series of Applied Sciences, vol. 357, pp. 1-43, 1999.

[Ern99b] R. Ernst. *Automatisierter Entwurf eingebetteter Systeme.* at - Automatisierungstechnik, Oldenbourg, pp. 285–294, Juli 1999.

[GCD92] R. Gupta, C.N. Coelho, G. DeMicheli, *Synthesis and simulation of digital systems containing interacting hardware and software components*. Proceedings Design Automation Conference 92, pp. 225-230, 1992.

[GLL99] A. Girault, B. Lee, E.A. Lee. *Hierarchical finite state machines with multiple concurrency models*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18 (6), pp. 742ff, June 1999.

[HeP03] J. Hennessy, D. Patterson. *Computer architecture – A quantitative approach.* Morgan Kaufman, 3. Aufl., 2003.

[ITR01] J. Adam et al.. *International technology roadmap for semiconductors (ITRS) 2001,* http://public.itrs.net.

[LeM97] R. Leupers, P.Marwedel. *Retargetable Compiler Technology for Embedded Systems*, Kluwer, 1997.

[Mad97]Jan Madsen. *LYCOS: The Lyngby Co-Synthesis System*. Design Automation for Embedded Systems, Kluwer, vol. 2, no. 2, pp. 195-235, March 1997.

[MaG95] P. Marwedel, G. Goossens. *Code generation for embedded processors*. Kluwer, 1995.

[Pro02] Proceler. *VAARC*. www.proceler.com

[Syn02] Synopsys. CoCentric System Studio. www.synopsys.com

[Ten02] Tensilica. *Xtensa*. www.tensilica.com

[VVB+96] D. Verkest, K. van Rompaey, I. Bolsens, H. De Man. *CoWare – A design environment for heterogeneous hardware/software systems*. Design Automation for Embedded Systems, Kluwer, vol. 1, no. 4, pp. 357-386, Oct. 1996.

[YNG+02] S. Yoo, G. Nicolescu, L. Gauthier, A.A. Jerraya. *Automatic generation including fast timed simulation models of operating systems in multiprocessor SoC communication design.* Proceedings DATE 2002, Paris, France, March 2002.

[ZRE+03] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, J. Teich. *SPI – A system model for heterogeneously specified embedded systems.* To appear in: IEEE Transactions on VLSI Systems, 2003.

[ZVE01] Zentralverband Elektrotechnik- und Elektronikindustrie. *Abnehmerstruktur stabilisiert deutschen Markt.* Pressemitteilung Pr - 86/2001, 2001.