Skalierbare Rechensysteme für Echtzeitanwendungen

Stefan Aust und Harald Richter

Institut für Informatik
TU Clausthal, 38678 Clausthal-Zellerfeld
stefan.aust|harald.richter@tu-clausthal.de

Zusammenfassung. Die steigende Komplexität von Echtzeitanwendungen führt zu wachsenden Problemen im Bereich des Task-Schedulings. Eine steigende Nachfrage nach mehr Rechenleistung lässt sich nur durch Mehrprozessorsysteme (i.e. Multicore-CPUs) erfüllen. Das Konzept des Space Sharing bietet die Möglichkeit, Echtzeitaufgaben physikalisch auf ein konfigurierbares Mehrprozessorsystem abzubilden und dort auszuführen. Ein entscheidendes Kriterium für die Echtzeitfähigkeit von Mehrprozessorsystemen ist die Interprozessorkommunikation. Hierfür eignen sich mehrstufige Netze, deren Ursprung im Bereich der Telekommunikation liegt und die auch bei Parallelrechnern eingesetzt werden. Allerdings sind diese bisher nicht echtzeitfähig. In diesem Beitrag wird vorgeschlagen, Space Sharing in Kombination mit blockierungsfreien mehrstufigen Netzen für die Kommunikation zwischen Echtzeit-Tasks einzusetzen, da sie eine vorhersagbare Latenz bei der Interprozessorkommunikation haben.

1 Einleitung

Die Datenverarbeitung in einem Echtzeitsystem zeichnet sich dadurch aus, dass Steuerungs-, Regelungs- und Prozessautomationsaufgaben entweder innerhalb definierter Zeitgrenzen oder zu bestimmten Zeitpunkten ausgeführt werden müssen. Dies wird durch ein mehr oder weniger komplexes Task Scheduling erreicht. In der Vergangenheit ist die Komplexität von Echtzeitanwendungen stetig weiter gewachsen, so dass die zeitliche Korrektheit der Software immer schwieriger zu garantieren ist. Dadurch entstehen zwei Probleme. Das erste Problem besteht in der massiv wachsenden Nachfrage nach mehr Rechenleistung, um die Echtzeitbedingungen zu erfüllen. Nach dem derzeitigen Stand kann diese Nachfrage nicht von einem einzelnen Prozessor, sondern nur durch den Einsatz von Mehrprozessorsystemen gedeckt werden, die als Multicore-CPUs ausgeführt sind. Mehrprozessorsysteme existieren zwar seit vielen Jahren in der Form von großen Parallelrechnern mit zehntausenden Prozessoren, allerdings sind solche Supercomputer auf maximale Rechenleistung ausgelegt. Ein Echtzeitverhalten bei der Datenverarbeitung weisen diese Rechner nicht auf. Das zweite Problem zeigt sich im Multitasking-Betrieb. Je komplexer die Echtzeitanwendung, desto schwieriger wird es, eine geeignete Scheduling-Strategie für die Ausführung nebenläufiger Tasks (Prozesse) zu finden, welche das gewünschte Verhalten des Rechensystems garantiert. In dieser Situation verfolgt der Ansatz des Space Sharings

W.A. Halang (ed.), *Herausforderungen durch Echtzeitbetrieb*, Informatik aktuell,
DOI 10.1007/978-3-642-24658-6_13,

das Ziel, eine skalierbare Rechnerarchitektur mit ausreichend Rechenleistung für die Verarbeitung von Echtzeitdaten zur Verfügung zu stellen. Space Sharing für Echtzeit-Anwendungen wurde erstmals von den Autoren in [1] und [2] eingeführt.

2 Space Sharing

Der heute weit verbreitete Multitasking-Betrieb eines Rechensystems zeichnet sich durch die nebenläufige Ausführung mehrerer Tasks aus. In einem Single- oder Multi-Core-System besteht das Problem bei der nebenläufigen Ausführung darin, dass deutlich mehr Tasks als Prozessoren existieren, wodurch Konkurrenzsituationen zwischen ablaufbereiten Tasks entstehen. Um das Problem des Task-Scheduling zu lösen, schlagen wir Space Sharing für die nebenläufige Ausführung von Tasks vor [1]. Space Sharing basiert auf dem Prinzip des „divide et impera", das auch bei rechenintensiven Aufgaben in Parallelrechnern erfolgreich angewendet wird. Bei Parallelrechnern erfolgt die Steigerung der Rechenleistung durch die Zerlegung einer Rechenaufgabe in möglichst feingranulare Teilaufgaben und der anschließenden simultanen Ausführung auf vielen Prozessoren. Dieses Prinzip funktioniert auch bei der Ausführung nebenläufiger Tasks. Bei Space Sharing wird die Anzahl der Prozessoren so weit erhöht, dass für jede Teilaufgabe ein Prozessor exklusiv zugeordnet werden kann (Abb. 1). Auf diese Weise wird die Konkurrenzsituation nebenläufiger Tasks gelöst. Der Begriff Space Sharing rührt daher, dass Tasks auf die vorhandene Chipfläche verteilt werden.

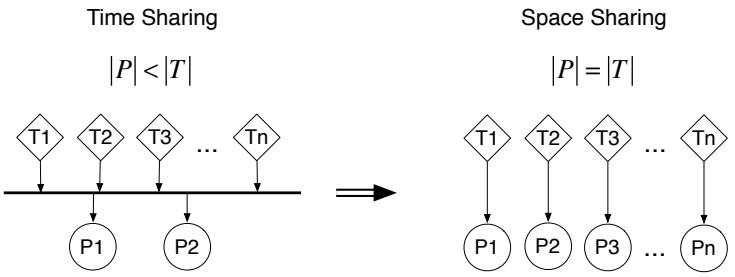

Abb. 1. Time-Sharing vs. Space Sharing [1]

3 Interprozessorkommunikation

Die Eigenschaften eines Multiprozessorsystems werden ganz wesentlich von der Kommunikation zwischen den Prozessoren bestimmt. Um die Echtzeitfähigkeit des Rechensystems sicherzustellen, muss die Interprozessor-Kommunikation in Echtzeit erfolgen. Bei Parallelrechnern beruht die Kommunikationshardware auf

einem skalierbaren Verbindungsnetzwerk zwischen den Prozessoren. Da das Multiprozessorsystem als System-on-Chip (MPSoC) auf einem einzigen Chip implementiert werden soll, wird das Verbindungsnetzwerk als Network-on-Chip (NoC) auf demselben Chip integriert. NoCs lösen mehr und mehr konventionelle Bussysteme ab, die aufgrund ihrer geringen Skalierbarkeit und Bandbreite in Multiprozessorsystemen nicht effizient einsetzbar sind [3, 4]. Dazu wurden in der Vergangenheit diverse Studien an NoCs vorgenommen [5–8]. Diese beschränken sich allerdings auf die Implementierung von Topologien statischer Netze ohne die Berücksichtigung von Echtzeiteigenschaften.

3.1 Mehrstufige Netze

Bei einigen Parallelrechnern werden dynamische Netze als Verbindungsnetzwerk verwendet, die auch als indirekte oder mehrstufige Netze (engl.: Multistage Interconnection Network, MIN) bezeichnet werden. Sie bieten den Vorteil, dass mit ihnen sehr große Netzwerke zu vergleichsweise niedrigen Kosten in Hardware realisiert werden können. Diese Eigenschaft ist für ein NoC aufgrund der limitierten Chipfläche von besonderem Interesse.

Die Architektur mehrstufiger Netze besteht im Wesentlichen aus in Stufen organisierten Kreuzschaltern der Größe 2x2, die wahlweise auf parallelen oder gekreuzten Durchgang gesetzt werden können (Abb. 2).

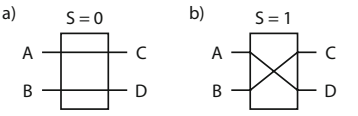

Abb. 2. 2x2 Kreuzschalter: a) gerade, b) gekreuzt

Die einzelnen Schalterstufen sind durch Permutationsfunktionen, den Verdrahtungsstufen, miteinander verbunden. Die Verbindungen zwischen Ein- und Ausgängen des Netzwerks entstehen aus der Verkettung aller Stufen, wobei die Kreuzschalter für eine Vielzahl potentieller Permutationen sorgen. Die Abb. 3 zeigt ein mehrstufiges Netz der Größe 8x8, bei dem die Kreuzschalter durch Unshuffle-Permutationen verdrahtet sind [9].

Im Vergleich zu den bisher bei NoCs eingesetzten Topologien statischer Netze wie Gitter oder Torus, bieten dynamische Netze eine ganze Reihe von Vorteilen. Bei statischen Netzen besitzt jeder Knoten mehrere Nachbarknoten, d.h., dass jeder Knoten mehrere Netzwerkschnittstellen aufweisen muss. Dazu kommt, dass jeder Rechenknoten gleichzeitig auch Routerknoten sein muss. Bei einem mehrstufigen Netzwerk, wie in der Abb. 3 dargestellt, benötigt jeder Rechenknoten unabhängig von der Netzgröße nur eine einzige Schnittstelle zum Netzwerk. Eine Routingaufgabe müssen die Rechenknoten nicht übernehmen. Einen weiteren wichtigen Vorteil bietet die Möglichkeit zur Leitungsvermittlung. Hierdurch

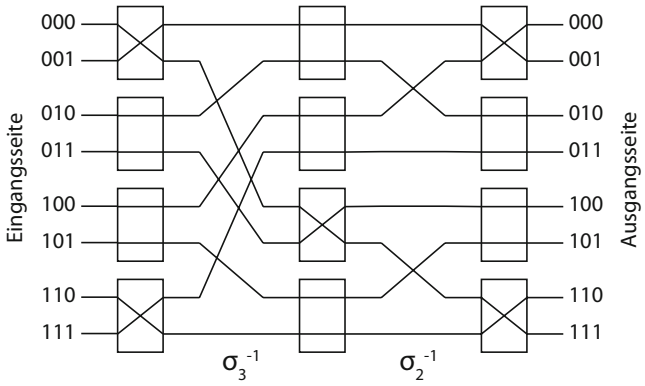

Abb. 3. Permutation im Baseline-Netz der Größe 8x8

kann der Sender die Nachricht über einen exklusiv zur Verfügung stehenden Kommunikationspfad direkt an den Empfänger übermitteln. Dadurch entfällt die Zeit, die ein Transfer über Zwischenknoten auf dem Weg vom Sender zum Empfänger benötigt. Die Entstehung von Datenstau an besonders verkehrsbelasteten Routerknoten im Netz, den Hotspots, wird vermieden, was die Echtzeitfähigkeit deutlich verbessert. Durch ein zentral organisiertes Routing kann das Echtzeitverhalten des Netzes einfacher sichergestellt werden. Nachteilig ist bei einer zentralen Wegewahlinstanz allerdings die höhere Komplexität bei der Leitungsvermittlung. Die Gefahr des Single Point of Failure, d.h. ein Totalausfall der Kommunikation durch einen defekten Netzwerk-Controller, ist aufgrund des SoC-Designs jedoch als sehr gering einzuschätzen.

Mehrstufige Netze sind mit Ausnahme ihrer Echtzeiteigenschaften heute weitgehend erforscht. Aus unserer Sicht lassen sich mehrstufige Netze hinsichtlich ihrer Echtzeiteigenschaften in nicht-blockierungsfreie und blockierungsfreie Netze einteilen. Die Abb. 4 zeigt beide Kategorien mehrstufiger Netze, die in den nachfolgenden Abschnitten näher diskutiert werden.

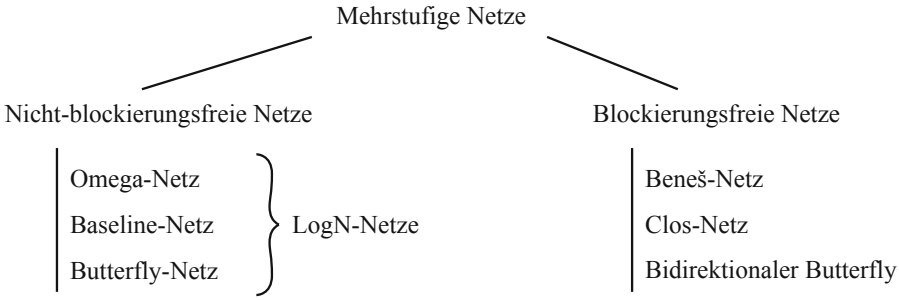

Abb. 4. Echtzeit-Kategorien mehrstufiger Netze

3.2 Nicht-blockierungsfreie Netze

Zu den nicht-blockierungsfreien Netzen zählen alle Varianten der sog. logN-Netze [9]. Deren Bezeichnung ergibt sich aus der Eigenschaft, dass sie aus $N\log_2 N$ Kreuzschaltern in $\log_2 N$ Stufen aufgebaut sind, wobei N die Anzahl der Ein- und Ausgänge im Netzwerk bezeichnet (vgl. Abb. 3). LogN-Netze benötigen die kleinste Anzahl von Stufen, um jeden Eingang mit jedem Ausgang verbinden zu können. Allerdings gilt bei logN-Netzen die Einschränkung, dass nicht alle Permutationen realisiert werden können. D.h., dass zwar jeder Eingang mit jedem Ausgang verbunden werden kann, aber im Allgemeinfall nicht alle Eingänge gleichzeitig. Aus diesem Grund werden logN-Netze auch als nicht-blockierungsfreie Netze bezeichnet. Positiv zu werten ist, dass logN-Netze aufgrund ihrer Eigenschaft der Pfadeindeutigkeit einen äußerst einfachen Routingalgorithmus erlauben. Für Echtzeitaufgaben sind sie nur bedingt geeignet, weil die Latenz der Datenübertragung von den konkreten Verbindungswünschen abhängt. Bei bestimmten ungünstigen Verbindungen muss einer oder mehrere Sender warten, bis die Kollision im Netz aufgelöst ist. Um die Echtzeitfähigkeit von logN-Netzen trotz ihrer Nichtblockierungsfreiheit zu gewährleisten, muss das Vermitteln von Leitungen durch das Netz kontrolliert erfolgen. Dies geschieht mit Hilfe eines Schedulings der zu vermittelnden Verbindungen. Das Scheduling kann entweder statisch während der Konfigurationsphase oder dynamisch zur Laufzeit erfolgen. Der zu beschreitende Pfad und die Dauer der Verbindung kann im statischen Fall in einer Schedulingtabelle festgeschrieben werden. So kann bereits vor der Ausführung einer Software-Anwendung geprüft werden, ob alle Verbindungswünsche fristgerecht erfüllt oder ob Zeitschranken (Deadlines) verletzt werden. Beim dynamischen Scheduling muss ähnlich wie beim Task-Scheduling ein zeiteffizienter Scheduling-Algorithmus existieren. Eine Vorhersage über das tatsächliche Zeitverhalten ist dann nur durch eine Vorabsimulation der Interprozessor-Kommunikation möglich. Letzlich wird damit aber nur das Task-Scheduling auf das Schedulen von Verbindungen verlagert und stellt somit keinen Gewinn dar.

3.3 Blockierungsfreie Netze

Schaltet man zwei nicht-blockierungsfreie logN-Netze hintereinander, so erhält man die zweite Echtzeit-Kategorie mehrstufiger Netze, die sog. blockierungsfreien Netze. Die Abb. 5 zeigt das von V. E. Beneš vorgestellte Beneš-Netz, dessen Topologie aus der Verkettung zweier gespiegelter Baseline-Netze entsteht, wobei die letzte Stufe des ersten Netzes und die erste Stufe des zweiten Netzes miteinander verschmelzen [10]. Durch die auf $2\log_2 N - 1$ vergrößerte Stufenanzahl besitzen blockierungsfreie Netze die Eigenschaft, jeden Eingang zu jedem Zeitpunkt mit jedem freien Ausgang verbinden zu können. Diese Eigenschaft basiert auf der Existenz von Pfad-Alternativen in dem fast doppelt so großen Netz. Die Anzahl möglicher Pfade von jedem Eingang zu jedem Ausgang steigt proportional mit der Anzahl der Ein- und Ausgänge.

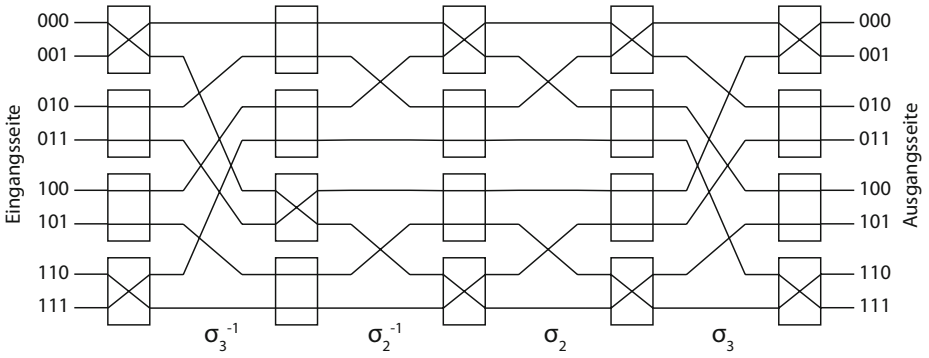

Abb. 5. Permutation im Beneš-Netz der Größe 8x8

Aufgrund der fehlenden Pfadeindeutigkeit können zwar alle gewünschten Verbindungen sofort vermittelt werden, allerdings müssen u. U. bereits bestehende Verbindungen auf alternative Pfade geroutet werden. Da dieser Vorgang bei sequentieller Ausführung eine nichtlineare Zeitkomplexität besitzt, entstehen hier im Vergleich zu den nichtblockierungsfreien logN-Netzen neben den zusätzlichen Stufen weitere Kosten durch das sehr aufwändige Routingverfahren. Dafür bieten diese Netze eine garantiert echtzeitfähige Kommunikationsstruktur ohne die Verwendung eines Schedulings.

Dezentrales Routing im Beneš-Netz Das Routing stellt bei blockierungsfreien Netzen den größten Kostenfaktor dar. Aus diesem Grund gab es Bestrebungen, selbstorganisierende Netze (engl.: self-routing networks) zu entwerfen, um die Routingfunktion ähnlich den logN-Netzen dezentral auf die Ebene der Kreuzschalter zu verlagern [11, 12]. Allerdings ist bis heute kein dezentraler Routingalgorithmus bekannt, der in der Lage ist, alle Verbindungswünsche im Beneš-Netz zu routen. Der Grund für diese Einschränkung liegt in der Topologie des Beneš-Netzes. Aufgrund seines rekursiven Aufbaus und der Subshuffle-Verdrahtung der inneren Stufen zerfällt das Netz zur Mitte in zwei unabhängige Teilnetze. Da durch die Shuffle-Verdrahtungsstufe von jedem Kreuzschalter jeweils ein Ausgang in die obere und ein Ausgang in der untere Hälfte verdrahtet ist, müssen beim Routing die komplementären Ein- bzw. Ausgänge der Kreuzschalter auf beiden Seiten berücksichtigt werden. Aufgrund dieser Abhängigkeiten kann das Routing nicht auf Schalterebene erfolgen.

Zentrales Routing im Beneš-Netz Ein bekanntes Routing-Verfahren, dass alle möglichen Verbindungswünsche im Beneš-Netz erfüllen kann, ist das Looping-Routing [9]. Der große Nachteil dieses Verfahrens liegt darin, dass die Schalterstellungen von einer zentralen Instanz in $\mathcal{O}(\mathrm{N}\log_2\mathrm{N})$ Schritten seriell bestimmt werden. Das heißt, dass der Aufbau einer einzigen Verbindung in einem Netz-

werk mit N=512 Prozessoren in 4096 Einzelschritten erfolgt. Solche Kosten sind in einem Echtzeitsystem nicht tolerierbar.

Die Lösung für das Routing-Problem im Beneš-Netz ergibt sich aus dem sog. Richter-Netz, das von einem der Autoren in [13] vorgestellt wurde. Bei diesem Netz handelt es sich um ein doppeltes Baseline-Netz, welches in eine linke und eine rechte Seite geteilt wird. Das Routing der rechten Seite erfolgt wie in einem nicht-blockierungsfreien logN-Netz, d.h. in $\log_2 N$ Stufen und der bitweisen Auswertung der binären Zieladresse $I = (i_{n-1},...,i_0)$. Damit die rechte Seite blockierungsfrei geroutet werden kann, muss die Eingangspermutation der Zieladressen auf der linken Seite entsprechend vorsortiert werden. Dies geschieht in den ersten $\log_2 N - 1$ Stufen des Netzes durch die stufenweise Separation der Komplementäradressen $I = I' = (i_{n-1},...,i_{s+1})$ in die inneren Teilnetze (Abb. 6).

Neben der funktionalen und topologischen Äquivalenz, die für alle logN-Netze gilt, besitzt das Baseline-Netz zusätzlich die Besonderheit des identischen Routings der normalen und der gespiegelten Variante [9]. Aufgrund dieser besonderen Eigenschaft funktioniert das Routing-Verfahren des doppelten Baseline-Netzes auch im Beneš-Netz, welches aus einem normalen und einem inversen Baseline-Netz aufgebaut ist, so dass das Richter-Netz ohne Einschränkungen in ein Beneš-Netz überführt werden kann (Abb. 6).

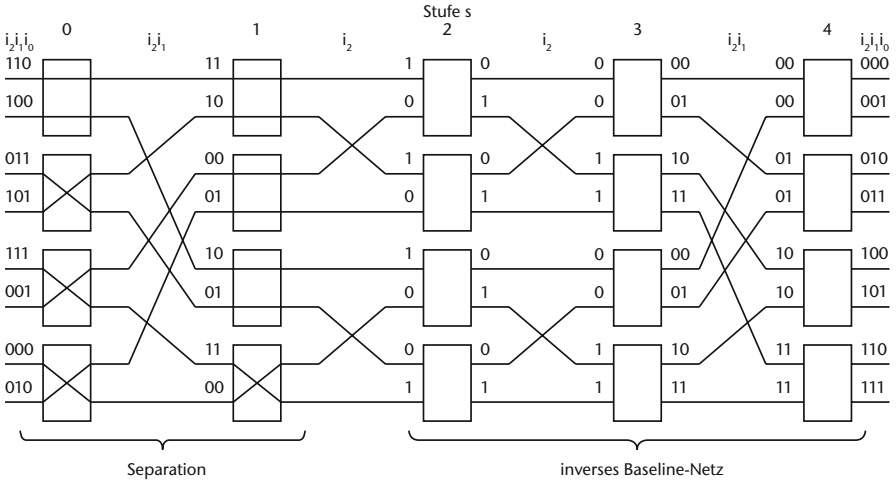

Abb. 6. Echtzeitfähiges Routing im Beneš-Netzes

Der in Abb. 6 dargestellte Routing-Algorithmus lässt sich durch kombinatorische Logik in disjunktiver oder konjunktiver Normalform ausführen. Das entstandene Schaltnetz kann direkt in Hardware implementiert werden (vgl. Abschnitt 4), wodurch das gesamte Netz unabhängig von dessen Größe quasi verzögerungsfrei geroutet werden kann.

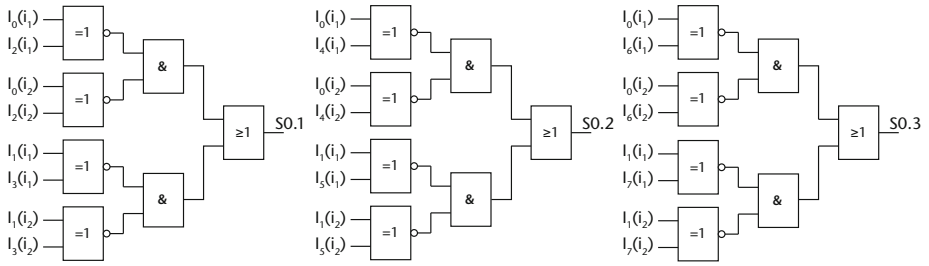

Abb. 7. Schaltnetz zur Bestimmung der Schalterstellungen der ersten Stufe

4 Implementierung im FPGA

Für Space Sharing in Echtzeitsystemen sind zwei grundsätzliche Voraussetzungen nötig. Die erste Voraussetzung besteht darin, dass Art und Umfang der Software vor der Laufzeit bekannt sein müssen und sich zur Laufzeit nicht mehr ändern. Diese Forderung ist in Echtzeitsystemen i.d.R. erfüllbar. Die zweite Voraussetzung betrifft die Implementierung des Multiprozessorsystems in Hardware. Da die benötigte Anzahl an Prozessoren von der Anzahl nebenläufiger Tasks abhängt, muss die Rechenhardware so flexibel anzupassen sein wie Software. Diese Forderung kann mittels programmierbarer Logikbausteine (FPGAs) und der Implementierung als Multiprozessorsystem on-Chip (MPSoC) erfüllt werden. Die Prozessorlogik kann in Form von sog. Softprozessoren flexibel in FPGA-Hardware implementiert werden. Die maximale Taktrate von FPGA-basierten Softprozessoren erreicht mit 50 bis 200 MHz derzeit nur rund ein Zehntel der Taktrate „richtiger“ Prozessoren, da jeder Softprozessor aber nur einen kleinen Teil der Anwender-Software ausführt, ist die Rechenleistung von Softprozessoren im Allgemeinen ausreichend. Auch der zur Verfügung stehende Speicher pro Prozessor ist deutlich kleiner. Viele Aufgaben aus der Steuerungs-, Regelungs- und Prozessautomationstechnik erfordern jedoch keine GHz und GByte. Zudem bieten Softprozessoren die Möglichkeit, dass die Prozessororganisation und -architektur beliebig erweitert oder reduziert werden kann. Softprozessoren erlauben damit eine bedarfsorientierte Implementierung spezifischer Befehlssätze sowie zusätzlicher Recheneinheiten und Koprozessoren, sofern erforderlich. Die maximale Anzahl an Prozessoren in einem FPGA-basierten MPSoC wird durch die Architektur der Prozessoren und durch die Anzahl der verfügbaren Logikzellen auf dem jeweiligen FPGA begrenzt. In den letzten Jahren haben FPGAs aufgrund höherer Integrationsdichten und effizienterer Technologien in jeder Hinsicht einen enormen Entwicklungsschub erfahren (vgl. Abb. 8), so dass mit der derzeit aktuellen Generation Virtex-7 (Stand: Juni 2011) Multiprozessorsysteme mit mehreren hundert Softprozessoren auf einem einzigen Chip, wenn auch bei relativ geringen Taktraten und kleinem Speicher, technisch realisierbar sind.

Die Abb. 9 zeigt den Aufbau und die Implementierung eines MPSoC mit 8 Prozessoren, wie es von uns in einem Virtex-4 FPGA der Firma Xilinx getestet wurde. Die Prozessoren sind proprietäre 32-Bit RISC Softprozessoren vom Typ

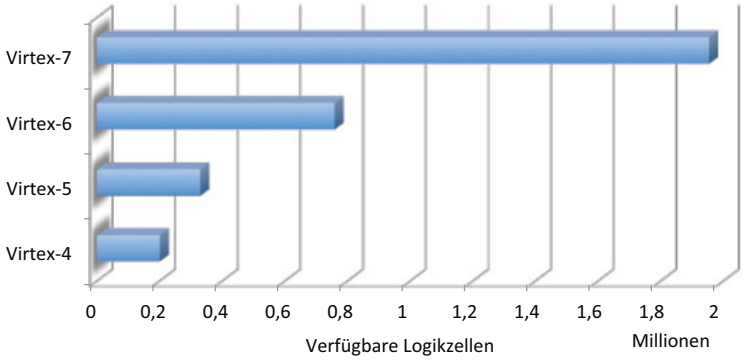

Abb. 8. Evolution von Xilinx FPGAs anhand der verfügbaren Logikzellen pro Chip

MicroBlaze. Jeder Prozessor besitzt lokale Speicher für Daten und Befehle. Die Interprozessor-Kommunikation erfolgt durch das Senden von Nachrichten über 32-Bit breite unidirektionale Kanäle (Fast Simplex Link). Die maximale Bandbreite des Netzwerks steigt aufgrund der simultanen Übertragung auf mehreren Kanälen proportional mit der Anzahl N der Teilnehmer im Netz, was die Skalierbarkeit des MPSoC deutlich verbessert. Das Verbindungsnetzwerk inkl. Routingalgorithmus ist in einem eigenen IP-Core mit entsprechend vielen FSL-Interfaces untergebracht. Zum Senden einer Nachricht wird zunächst die Zieladresse über den Steuerkanal des FSL-Interfaces an das Verbindungsnetzwerk gesendet. Sobald der gewünschte Ausgang nicht belegt ist, wird die Verbindung vom Sender zum Empfänger aufgebaut, so dass bereits im nächsten Takt die Nachricht über den Datenkanal an den Empfänger der Nachricht übermittelt wird. Der Abbau der Verbindung erfolgt durch das Senden der eigenen Adresse über den Steuerkanal an das Verbindungsnetzwerk.

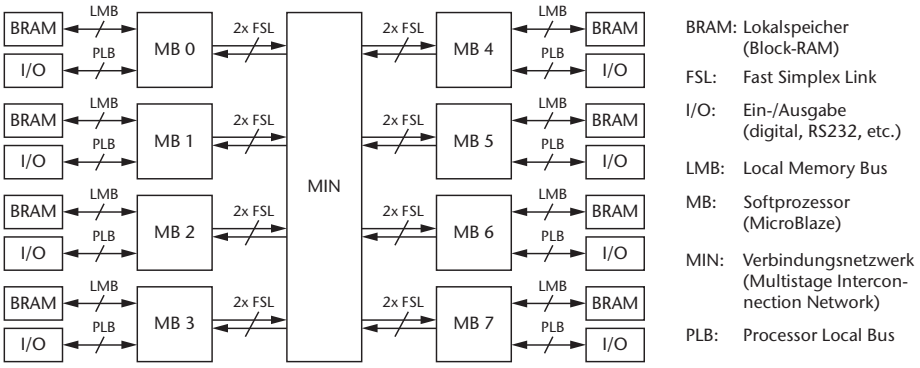

Abb. 9. Implementierung eines MPSoC mit 8 Prozessoren im Xilinx FPGA

Die Kommunikation zwischen Sender und Empfänger ist durch zusätzliche FIFO-Speicher in den FSL-Verbindungen zeitlich voneinander entkoppelt. Dies ermöglicht ein sog. GALS-Design (globally asynchronous, locally synchronous), bei dem die einzelnen Prozessoren unterschiedlich getaktet werden können, um die dynamische Leistungsaufnahme des Rechensystems zu optimieren. Die vom Taktteiler erzeugte Taktrate kann für jeden Prozessor zur Laufzeit vom Programm so verändert werden, dass rechenintensive Programmteile mehr Rechenleistung bekommen als z.B. Warteschleifen. Dazu wird die Taktrate als Kommandozeile im Programm deklariert und bei Ausführung als Wert an den Taktteilerbaustein übergeben. Eine weitere Möglichkeit zur Senkung des Energieverbrauchs besteht bei Space Sharing darin, die Größe der lokalen Befehls- und Datenspeicher im FPGA den Anforderungen des Programms anzupassen.

Literaturverzeichnis

1. Aust, S., Richter, H.: *Space Division of Processing Power for Feed Forward and Feed Back Control in Complex Production and Packaging Machinery.* Proceedings of World Automation Congress, Kobe: S. 1–6, 2010
2. Aust, S., Richter, H.: *Ein Echtzeitparallelrechner zur Rezentralisierung von Steuergeräten im Automobil.* erschienen in Tschöke, H. (Hrsg.), Krahl, J. (Hrsg.), Munack, A. (Hrsg.): *Innovative Automobiltechnik II.* S. 70–88, Expert Verlag 2010
3. Benini, L., und De Micheli, G.: *Networks on chips: a new SoC paradigm.* IEEE Computer Magazine, Vol. 35, Nr. 1, S. 70–78, 2002
4. Lee, H .G., Chang, N., Ogras, U. Y., und Marculescu, R.: *On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches.* ACM Transactions on Design Automation of Electronic Systems, Vol. 12, Nr. 3, Artikel 23, 2007
5. Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Öberg, J., Tiensyrjä, K., und Hemani, A.: *A Network on Chip Architecture and Design Methodology.* Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), S. 105–112, 2002
6. Bjerregaard, T., und Mahadevan, S.: *A Survey of Research and Practices of Network-on-Chip.* ACM Computing Surveys (CSUR), Vol. 38, Ausgabe 1, 2006
7. Zeferino, C. A., und Susin, A. A.: *SoCIN: A Parametric and Scalable Network-on-Chip.* Integrated Circuits and Systems Design (SBCCI), S. 169–174, 2003
8. Benini, L., und De Micheli, G.: *Networks on Chips: Technology and Tools.* San Francisco: Morgan Kaufmann 2006
9. Richter, H.: *Verbindungsnetzwerke für parallele und verteilte Systeme.* Heidelberg: Spektrum Akademischer Verlag 1997
10. Grammatikakis, M. D., Hsu, D. F., und Kraetzl, M.: *Parallel System Interconnections and Communications.* Boca Raton; London; New York; Washington: CRC Press Inc. 2000
11. Nassimi, D., Sahni, S.: *A Self-Routing Beneš Network and Parallel Permutation Algorithms.* IEEE Trans. on Comp., Vol. C-30, Ausgabe 5, S. 332–340, 1981
12. Raghavendra, C. S., Boppana, R. V.: *On Self-Routing in Beneš and Shuffle-Exchange Networks.* IEEE Trans. on Comp., Vol. 40, Nr. 9, S. 1057–1064, 1991
13. Richter, H.: *Multiprozessor mit dynamisch variabler Topologie.* Dissertation, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität München, 1987