DESIGN&ELEKTRONIK
PRODUKTE UND KNOW-HOW FÜR DEN ELEKTRONIK-ENTWICKLER

INDUSTRIECOMPUTER

Embedded Computing
Maßgeschneiderte Prozessoren

Entwicklungssysteme
Co-Design von FPGA und Leiterplatte

Kommunikation
ATCA-Plattformkonzept

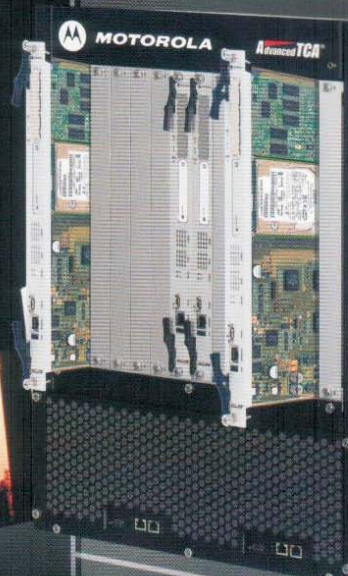
Bahntechnik
Elektronik auf der Schiene

Komplettsysteme
Office in der Produktion

Media-Gateway-Server

IP-Conferencing-Systems

Remote-Network-Controller



Anwendungsspezifische Prozessor-Cores

Prozessoren maßgeschneidert

Anwendungspezifische Prozessoren erfreuen sich wachsender Beliebtheit. Die Realisierung und Implementierung mit konventionellen Entwurfsmethoden ist jedoch eine große Herausforderung. Dieser Artikel erläutert die Probleme und bietet Lösungsansätze für eine neuartige, automatisierte Implementierung.

Oliver Schliebusch, Anupam Chattopadhyay, David Kammler *und* **Martin Witte**
forschen u.a. auf dem Gebiet der Entwurfsautomatisierung unter Leitung der Professoren
Gerd Ascheid, Rainer Leupers *und* **Heinrich Meyr**
am Lehrstuhl für Integrierte Systeme der Signalverarbeitung an der RWTH Aachen

ASIPs, also anwendungsspezifische Prozessoren (Application Specific Instruction Set Processors) vereinen die Leistung optimierter, integrierter Schaltungen mit der Programmierbarkeit (und damit Flexibilität und Wiederverwendbarkeit) von Prozessoren. Sie scheinen damit die ultimative Lösung für die derzeitige Komplexitätskrise in der Chipbranche zu sein. Hinderlich für ihren Siegeszug ist allerdings der enorm hohe Entwurfsaufwand. Software-Tools wie z.B. C-Compiler, Assembler, Linker und Simulator sind jeweils von Hand zu programmieren. Selbstverständlich muss auch die Hardware entworfen und in das Gesamtsystem integriert werden. Der wichtigste Aspekt jedoch ist, dass der Prozessor, und damit alle erwähnten Elemente des Prozessordesigns, hinsichtlich eines gegebenen Problems zu optimieren sind. Fortlaufende Änderungen und Verbesserungen an der Spezifikation sind damit während der so genannten Architekturerkundung (»Architecture Exploration«) unausweichlich.

Über die letzten Jahre hinweg haben sich Architekturbeschreibungssprachen entwickelt, die es erlauben, den Zielprozessor hinsichtlich seiner Funktionalität und seines Befehlssatzes zu beschreiben und automatisch die erforderlichen Softwaretools zu generieren. Aufgrund der sehr kompakten Beschreibung und der Möglichkeit, Änderungen der Prozessorspezifikation schnell umzusetzen, eröffnet sich erst durch diese Sprachen die Möglichkeit einer effizienten Architekturerkundung. Allerdings haben sich diese Beschreibungssprachen von der reinen Hardwarebeschreibung weg entwickelt, so dass erforderliche Informationen über die Hardware, wie z.B. die Taktfrequenz oder die Chipfläche nicht mehr unmittelbar zu Verfügung stehen. Hierdurch ist zusätzlich eine manuelle Implementierung der Hardware in einer Hardwarebeschreibungssprache erforderlich. Das Abstraktionsniveau, welches derzeit für die Hardware-Implementierung in VHDL oder Verilog genutzt wird, bezeichnet man als Register-Transfer-Level (RTL). Allerdings bedeutet diese manuelle Implementierung einen Bruch im Entwurfsfluss von ASIPs und widerspricht der Idee der effizienten Architekturerkundung.

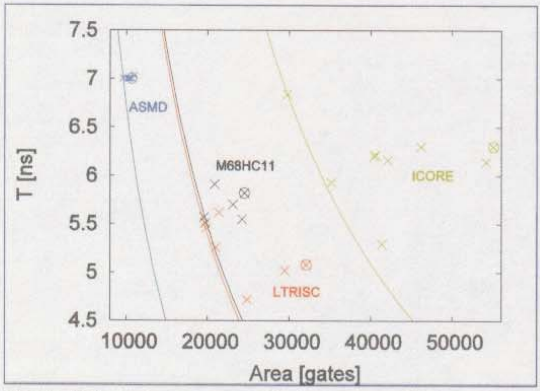


Bild 1 Taktrate in Bezug zur Fläche für verschiedene Prozessorarchitekturen

Zwischen Architektur- und Hardwarebeschreibung

Eine Architekturbeschreibungssprache beschreibt den Prozessor auf einem höheren Abstraktionsniveau als eine Hardwarebeschreibungssprache auf RTL. Einerseits sollen alle Architekturaspekte hinreichend erfasst, andererseits aber auch die Komplexität der Beschreibung in engen Grenzen gehalten werden. Dies lässt sich dadurch erreichen, dass oftmals nur Eigenschaften wie z.B. das Verhalten, Timing oder der Befehlssatz spezifiziert werden, ohne festzulegen, wie die einzelnen Eigenschaften in Hardware zu realisieren sind. So wird z.B. der Befehlssatz eindeutig hinsichtlich der Assembler-Syntax sowie der binären Codierung spezifiziert, allerdings beinhaltet ein Modell in einer Architekturbeschreibungssprache keinerlei Informationen darüber, wie die Befehle zu decodieren sind. Im Kasten »Quellcode 1« ist eine LISA-Operation spezifiziert, die einen MAC-Befehl (Multiply-Accumulate) beschreibt. Das Assembler-Format des Befehls ist in der so genannten SYNTAX-Section (Zeile 6) definiert, die binäre Darstellung in der CODING-Section (Zeile 7). Das Verhalten wird in der BEHAVIOR-Section (Zeilen 8-11) festgehalten und im Falle der Sprache LISA im Wesentlichen durch C-Code spezifiziert. Eine Hardwarebeschreibungssprache erscheint auf den

ersten Blick ähnlich einer Programmiersprache. Es gibt Möglichkeiten der Modularisierung, Konstrukte vergleichbar mit Funktionen oder Prozeduren, usw. Arithmetische Ausdrücke sowie Kontrollstrukturen wie z.B. if oder switch unterscheiden sich nur durch syntaktische Abweichungen. Allerdings existieren darüber hinaus ebenso Elemente zur Beschreibung von typischen Hardwareeigenschaften, wie z.B. Parallelität, reaktives Verhalten oder Timing. Anweisungen werden während der Simulation nicht mehr nur streng sequenziell ausgeführt, sondern repräsentieren u.U. parallele Hardware. Das oben genannte LISA-Codebeispiel ist, soweit dies in diesem Rahmen möglich ist, in eine Hardwarebeschreibung transformiert worden und im Kasten »Quellcode 2« dargestellt. Hierbei wird der Index für die Quell- und Zielregister aus dem Befehlswort extrahiert und den entsprechenden Signalen zugewiesen (Zeilen 4-7). Die nicht gezeigte Registerimplementierung sorgt für das Schreiben der entsprechenden Operanden auf die Signale reg_s1, reg_s2 und reg_s3. Die arithmetischen Operationen werden durchgeführt und das Ergebnis dem Signal zum Schreiben des entsprechenden Registers zugewiesen (Zeile 8). Zusätzlich ist noch ein Kontrollsignal notwendig, um der Registerimplementierung anzuzeigen, dass auch tatsächlich ein neuer Wert in das Register geschrieben werden soll (Zeile 9). Da die mathematische Berechnung nur stattfinden soll, wenn der entsprechende Befehl vorliegt, wird das Befehlswort auf die entsprechende Bitmaske überprüft (Zeile 2). Falls der aktuelle Befehl ein anderer ist, setzt ein »Reset« die verfügbaren Signale zurück (Zeilen 13-18).

Von der Architektur zur Hardware

Schon an diesem einfachen Vergleich wird deutlich, dass sich die Architekturbeschreibung darauf konzentriert, die Funktion des Prozessors zu beschreiben. Die Hardwarebeschreibung aber spezifiziert, wie diese implementiert ist. Dadurch wird erreicht, dass die Architekturbeschreibung wesentlich kompakter ist, als eine Hardwarebeschreibung und sich somit auch wesentlich besser dafür eignet, eine umfassende Architekturerkundung durchzuführen.

Eine automatische Synthese des Hardwaremodells nutzt nun die gegebene Spezifikation, um eine möglichst effiziente Implementierung der Funktion zu erreichen. Der erste Schritt im Syntheseprozess besteht in der Übersetzung der einzelnen Spezifikationen in eine Hardwarebeschreibung. Dabei wird jede Komponente (z.B. Befehlssatz, Register, Speicher, usw.) der Architekturbeschreibung separat in eine Hardwarebeschreibung übersetzt. Da die einzelnen Aspekte in der Architekturbeschreibung unabhängig voneinander spezifiziert sind, wird auch die Übersetzung in die Hardwarebeschreibungssprache einzeln vorgenommen. Die unabhängige Beschreibung in der Architekturbeschreibung bietet wiederum den Vorteil einer schnellen Architekturerkundung, da Aspekte geändert werden können, ohne eine Vielzahl von Abhängigkeiten beachten zu müssen. Eine schrittweise, entkoppelte Übersetzung resultiert allerdings in ineffizienter Hardware, verglichen mit handgeschriebenem Code.

Ein einfaches Beispiel verdeutlicht dieses Problem. Wie im vorangegangenen LISA-Codebeispiel erwähnt, wird das Verhalten eines Befehls durch C-Code spezifi-

ziert. Durch die Spezifikation des Verhaltens ohne das Ziel, die reale Hardware zu beschreiben, ist eine mehrfache Beschreibung gleicher Hardwareoperationen möglich und auch gewollt. Soll nun aus dieser Beschreibung Hardware generiert werden, ist es sicherlich sinnvoll, die verschiedenen Verhaltensbeschreibungen zusammenzufassen. Dabei sind der Typ der Operationen sowie zeitliche Randbedingungen (Exklusivität der Ausführung) zu berücksichtigen. Dieses Problem bzw. die entsprechenden Optimierungen sind unter dem Stichwort »Resource-Sharing« bekannt.

Ohne konkrete Syntheseergebnisse vorwegzunehmen, ergibt sich im Vergleich zu handgeschriebenen Hardwarebeschreibungen eine vergleichbare Taktfrequenz, allerdings eine Fläche, die bis zu einem Faktor zwei größer ist.

Möglichkeiten der Optimierung

Die erreichten Werte für Chipfläche, Taktfrequenz oder Leistungsverbrauch, ohne Optimierungen während der Generierung der Hardwarebeschreibung, sind überraschend. Obwohl Optimierungstechniken wie beispielsweise Resource-Sharing in den etablierten Entwurfswerkzeugen umgesetzt sind, schaffen es die aktuellen Produkte nicht, die generierte Hardwarebeschreibung hinsichtlich des angesprochenen Problems zu optimieren. Bei genauerer Betrachtung erscheint die Größe der generierten Architektur der entscheidende Faktor zu sein. Schon bei kleinen Architekturen muss das Gate-Level-Synthesetool Abhängigkeiten über mehrere Module, komplexen booleschen Funktionen und großen Bitbreiten analysieren. Dadurch steigt der erforderliche Rechenaufwand exponentiell an. Potenziell existierende Optimierungen sind in diesem Fall für existierende Synthesetools, ausschließlich basierend auf einer Hardwarebeschreibungssprache, nicht erreichbar.

Quellcode 1:

```
00 OPERATION
01 {
02   DECLARE
03   {
04     GROUP reg_s1, reg_s2, reg_d = {regi-
ster}
05   }
06   SYNTAX  { »MAC« reg_s1 »,« reg_s2 »,«
reg_d}
07   CODING  { 0b0101 reg_s1 reg_s2 reg_d}
08   BEHAVIOR
09   {
10     reg_d += reg_s1 * reg_s2;
11   }
12 }
```

Quellcode 2:

```
00 Process
01 {
02   if(instruction(15 DOWNTO 12) == »0101«)
03   {
04     addr_s1 <= instruction(11 DOWNTO 8);
05     addr_s2 <= instruction( 7 DOWNTO 4);
06     addr_s3 <= instruction( 3 DOWNTO 0);
07     addr_d  <= instruction( 3 DOWNTO 0);
08     reg_d   <= reg_s3 + (reg_s1 * reg_s2);
09     mac_OK  <= 1;
10   }
11   else
12   {
13     addr_s1 <= 0;
14     addr_s2 <= 0;
15     addr_s3 <= 0;
16     addr_d  <= 0;
17     reg_d   <= 0;
18     mac_OK  <= 0;
19   }
20 }
```

Aus diesem Grund wurden neue Optimierungstechniken untersucht, die Syntheseergebnisse liefern, die teilweise schon besser als handgeschriebener Code sind. Potenzial für neue Optimierungen bietet wieder ein Blick auf die Eigenschaften und Codebeispiele der Architekturbeschreibung sowie der Hardwarebeschreibung. Im Falle der Hardwarebeschreibung ist eindeutig spezifiziert, wie entsprechende logische und arithmetische Operationen zusammenhängen. Allerdings ist nirgends zu erkennen, dass es sich um einen Prozessor handelt, mit den entsprechenden Eigenschaften bzgl. Exklusivität oder auch zeitlichem Verhalten. Diese Informationen müssen aufwendig analysiert und berechnet werden. Andererseits liegen eben diese Informationen direkt in der Architekturbeschreibung vor. Bei einer Vereinigung der Informationen aus der Architekturbeschreibung und der generierten Hardwarebeschreibung sollte also die Komplexität der Optimierungen drastisch sinken, dies kann zu deutlich besseren Syntheseergebnissen führen.

Verschiedene Optimierungen, unter anderem auch Resource-Sharing, wurden umgesetzt. Die erzielten Ergebnisse sind viel versprechend. In Bild 1 sind die Ergebnisse (Takt in ns über Fläche in Gates) für Infineons ASMD und ICORE, Motorolas 68HC11 und LTRisc32 dargestellt. Die beiden Infineon-Architekturen sind ASIPs für die Decodierung digitaler Nachrichtensignale. Während der ASMD allgemein in diesem Bereich einsetzbar ist, ist der ICORE speziell für DVB-T entwickelt worden. Sowohl der M86HC11 als auch der LTRisc32 sind Mikrocontroller für eingebettete Systeme. Die Beispiele verdeutlichen, dass der Aspekt des Resource-Sharings entscheidenden Einfluss auf eine effiziente Implementierung hat. Die bzgl. Funktionalität und Chipfläche kleineren Architekturen bieten bei weitem nicht soviel Optimierungspotenzial wie große Architekturen, die dieselben Hardwareeinheiten durch verschiedene Befehle ansprechen. So konnte z.B. der ICORE in der Fläche um 43% von 54 KGates auf 31 KGates reduziert werden. Der handgeschriebene ICORE benötigte 42 KGates. Die gesamte Prozessorsynthese ist in die LISA-Prozessor-Entwurfsplattform integriert. Diese Plattform basiert auf den Forschungsergebnissen des Instituts für Integrierte Systeme der Signalverarbeitung der RWTH Aachen und wird von der Firma CoWare weiterentwickelt und vertrieben. Sowohl Softwaretools als auch die Hardwarebeschreibung werden automatisch aus einem gegebenen LISA-Modell generiert. Selbstverständlich ist es möglich, die erzeugten Simulatoren in das Gesamtsystem zu integrieren und in einer Co-Simulation zu verifizieren. (mc)

RWTH Aachen
Telefon 02 41/80 27 88 4
Fax 02 41/80 22 19 5