Entwurf eines generischen, applikationsspezifischen, transportgesteuerten Prozessor-Modells in VHDL und Validierung auf einem FPGA

Stefan Hauser

hausers@cs.tu-berlin.de

Hintergrund: Die stetig fortschreitende Entwicklung von Prozessoren hat in den letzten Jahren eine gravierende Änderung erfahren. Bisher galt es stets die Frequenz der einzelnen CPUs zu erhöhen. Mittlerweile geht der Trend jedoch zunehmend in Richtung Mehrkernprozessoren. Neben dieser generellen Tendenz werden auch andere neue Wege im Bereich der Prozessorarchitekturen eingeschlagen. Transportgesteuerte Datenflussarchitekturen bieten in diesem Bereich eine interessante Alternative. Des Weiteren gewinnen auch applikationsspezifische Prozessoren an Bedeutung. Diese Arbeit beschreibt den Entwurf, die Implementierung sowie die Validierung eines applikationsspezifischen und transportgesteuerten Prozessor-Modells.

1 Einleitung

Der grundsätzliche Aufbau einer transportgesteuerten Architektur besteht aus unterschiedlichen Funktionseinheiten, welche über ein Verbindungsnetzwerk Daten untereinander austauschen können (siehe [Cor98]). Der Datenaustausch wird durch Transportoperationen gesteuert, welche den Datenfluss, und damit den Ablauf der Applikation, innerhalb des Prozessors regeln. Corporaal beschreibt dabei ausschließlich sehr einfache Funktionseinheiten, die über eine einzige Operation verfügen. Im Entwurf von Menge (siehe [Men03]) werden komplexere Funktionseinheiten verwendet, denen zusätzliche Steuerinformationen für die Auswahl der richtigen Operation zur Verfügung gestellt werden müssen. Auch der vorliegende Entwurf sieht vielseitige Funktionseinheiten vor. Darüber hinaus kann die Art und Anzahl der Einheiten in Abhängigkeit der eigenen Applikation festgelegt werden. Das Design lässt sich ohne größeren Aufwand um eigene Funktionseinheiten erweitern, sofern diese die spezifizierte Schnittstelle einhalten. Des Weiteren können nicht nur die Funktionseinheiten an die eigene Applikation angepasst, sondern auch das Verbindungsnetzwerk kann für die eigene Anwendung adaptiert werden. Die Auswahl einer geeigneten Konfiguration erfolgt idealerweise mit Hilfe von Analyseverfahren, wie sie beispielsweise in [Gre08] vorgestellt werden.

2 Realisierung

Entscheidend für den erfolgreichen Einsatz einer derartigen Architektur ist das Verbindungsnetzwerk. Eine Datenübertragung mit geringster Latenzzeit stellt dabei das wesentlichste Entwurfskriterium dar. Daher sind schalterbasierte Netzwerke, wie sie für Mehrkernprozessoren verwendet werden, ungeeignet. Im vorliegenden Design wurde aus diesem Grund auf eine klassische busbasierte Verbindungsstruktur gesetzt. Neben der Anzahl der Busse kann auch festgelegt werden, welche Funktionseinheiten mit welchen Bussen verbunden werden. Das bietet mehr Freiraum für die Adaption an die eigene Applikation, da eine Verknüpfung aller Einheiten mit allen verfügbaren Bussen zwar maximale Flexibilität bietet, jedoch steigt damit auch signifikant der Hardwareaufwand und die Verzögerungszeit. Eine Spezifikation des Verbindungsnetzwerkes kann im festgelegten XML-Format angegeben werden und wird von den implementierten Werkzeugen weiterverarbeitet.
Bei der Anbindung der Funktionseinheiten zeigt sich eine wesentliche Herausforderung der Architektur. Bei fast allen Instruktionen handelt es sich um dyadische Operationen. Demzufolge konsumieren die meisten Funktionseinheiten je Operation zwei Werte, liefern aber jeweils nur ein Resultat. Ein effizienter Betrieb ist nur möglich, wenn nahezu alle Funktionseinheiten ausgelastet sind. Um das zu ermöglichen, wurden verschiedene Lösungsansätze entwickelt. Als erstes wurde eine erweiterte Speicherschnittstelle entworfen, welche dem Netzwerk mehrere zusammenliegende Datenworte anbietet und somit den parallelen Zugriff auf Speicherbereiche ermöglicht. Ferner können Werte über einen Bus mehreren Konsumenten zur Verfügung gestellt werden. Diese Form des Busbetriebes wird als Multicast bezeichnet.
Wie in [MHG09] beschrieben wurde, zeigen einfache Analysen von Programmen, dass Werte direkt von weiteren Operationen konsumiert werden. Dabei lässt sich oft eine spezielle Abfolge (z.B. Multiplikation und Addition) finden, die sehr häufig auftritt. Zur Unterstützung derartiger Befehlsfolgen können Funktionseinheiten direkt miteinander verbunden werden. Dadurch wird unter Umständen eine weitere Ausdünnung der Netzvermaschung möglich. Weiterhin lässt sich anhand von Applikationen erkennen, dass Werte häufig mehrfach von einer Einheit benötigt werden. Diese können, angestoßen durch spezielle Operationen, in den Eingangsregistern der Funktionseinheiten vorgehalten werden. Bei den Programmuntersuchungen fällt ebenfalls auf, dass Resultate oft von der sie erzeugenden Funktionseinheit weiterverarbeitet werden. Dieses Prinzip entspricht im Grunde einem 1-Addressrechner, bei dem das Ergebnis über ein Akkumulatorregister rückgekoppelt wird. Eine derartige Arbeitsweise der Funktionseinheiten wird von der Architektur ebenfalls unterstützt und verringert die Kommunikation über das Netz. Alle vorgestellten Mechanismen zur Reduktion der Netzauslastung lassen sich über spezielle Operationen aktivieren und werden durch eine spezielle Schnittstelle, welche die eigentlich Funktionseinheit mit dem Netzwerk verbindet, realisiert.
Da nicht alle erzeugten Werte im Folgetakt weiterverarbeitet werden können, ist eine Pufferung in prozessorinternen Registern erforderlich. Ein gemeinsamer Registerspeicher, der den Einheiten über das Verbindungsnetzwerk zur Verfügung steht, erschien nicht optimal. Im vorliegenden Design wurde auf einen verteilten Registerspeicher gesetzt, welcher im Prinzip aus den den Funktionseinheiten nachgeschalteten Ergebnisregisterspeichern be-

steht. Dadurch kann ein gegebenenfalls unnötiges Sichern von Werten in einem entfernten Register vermieden werden. Konsequenzen aus dieser Form des Registersatzes ergeben sich für das Programmiermodell, welches erheblich an Komplexität zunimmt. Für die Evaluation des Designs wurde ein Werkzeug konzipiert, welches einen Assembler für konkrete Prozessorinstanzen generieren kann.
Die erzeugten Binäroperationen steuern neben dem Verbindungsnetzwerk auch die einzelnen Funktionseinheiten. Daher kann zwischen der eigentlichen Transportoperation und der Kontrolloperation unterschieden werden. Interessant ist, wie die Kontrollinformation zur konkreten Funktionseinheit gelangt. Dazu wurden zwei unterschiedliche Wege untersucht: Zum einen die Bündelung von Kontroll- und Transportoperationen, welche sich für das Programmiermodell als übersichtlicher erwies, und zum anderen die unkonventionell wirkende Trennung beider Operationen. Klassische TTA Architekturen setzen auf die erste Variante, da die Funktionseinheiten sehr einfach gehalten sind und die Unterscheidung zwischen verschiedenen Operationen über virtuelle Adressen erfolgt.

3 Ergebnisse

Eine derartige Trennung erwies sich als äußerst vorteilhaft, da die Steuerinformationen gezielt für jede Einheit abgelegt werden können. Sind diese Instruktionen an die Transportoperation gekoppelt, so muss das Netz nicht nur die Daten zur richtigen Zieleinheit transportieren, sondern zusätzlich auch die Steuerinformationen richtig weiterleiten. Einsparungen zeigen sich sehr deutlich in der Synthese der unterschiedlicher Konfigurationen. Eine Abschätzung der notwendigen Hardwareressourcen kann bereits vor der Synthese anhand von empirisch ermittelten Formeln erfolgen.
Synthetisiert wurden unterschiedlichste Konfigurationen für FPGAs der Firma Xilinx. Dabei zeigte sich nicht nur eine deutliche Verringerung der benötigten Hardware bei der Verwendung von ausgedünnten Verbindungsnetzwerken, sondern auch eine spürbare Verringerung des kritischen Pfades. Daher ist ein spezialisiertes Netz mit minimaler Verbindungsstruktur stets zu bevorzugen. Allerdings erfordert dieser Entwurf deutlich höhere Entwicklungszeiten. Die Arbeit zeigt den manuellen Aufbau des Verbindungsnetzwerkes anhand von Datenflussgraphen, welche für die Zielapplikation erzeugt wurden.
Um die Funktionalität des Prozessors unter Beweis zu stellen, wurden einfache Applikationen ausgewählt, zu denen eine passende Konfiguration erstellt wurde. Programmierung und Registerallokation erfolgten per Hand, dabei erwies sich die Trennung von Transport- und Kontrolloperationen weniger hinderlich als anfangs angenommen. Es konnte sogar eine Verringerung des Speicherbedarfs um 27 % festgestellt werden. Ursache dafür ist im Wesentlichen eine Verkürzung der Transportoperationen durch den Wegfall der Kontrollinformation. Die zusätzlich hinzugekommenen Steuerinformationen je Funktionseinheit wiegen das aber bei weitem nicht auf.
Die Entwicklung einer eigenen Assemblersprache mit dazugehörigen Werkzeugen verkürzte den Validierungszyklus. So konnte beispielsweise die Berechnung der Fibonaccizahlen erfolgreich für eine speziell angepasste Prozessorinstanz parallelisiert werden.

Literatur

[Cor98] Henk Corporaal. *Microprocessor architectures from VLIW to TTA*. Wiley, Chichester [u.a.], 1998.

[Gre08] C. Gremzow. Quantitative global dataflow analysis on virtual instruction set simulators for hardware/software co-design. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, Seiten 377–383, 2008.

[Men03] Matthias Menge. *Zen-1, Prozessor mit kontrollflussgesteuertem Datenfluss / M. Menge. Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik. [Hrsg.: Die Professoren der Fakultät IV - Elektrotechnik und Informatik der Technischen Universität Berlin].* Leiter der Fachbibliothek Informatik, Sekretariat FR 5-4, Berlin, 2003.

[MHG09] Nico Moser, Stefan Hauser und Carsten Gremzow. Reduzierung der Kommunikation in TTA-Verbindungsnetzen mittels Laufzeitanalyse. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Seiten 107–116, 2009.