In: Kropf, Th., Kumar, R. and Schmid, D.
GI/ITG-Workshop Formale Methoden zum Entwurf korrekter Systeme
Bad-Herrenalb, Germany, March 1993, pp. 35-40

Ein hierarchisches Modell zur formalen Verifikation von RISC-Prozessoren

Sofiène Tahar, Ramayya Kumar
Institut für Rechnerentwurf und Fehlertoleranz (Prof. Dr. D. Schmid)
Universität Karlsruhe, Postfach 6980, W-7500 Karlsruhe

1. Einleitung

Die Hardware-Verifikation mit formalen Systemen gewinnt heutzutage immer mehr Interesse aus dem Umfeld der Hardware-Designer. Hauptziel vieler Forschungsarbeiten ist die Erstellung von automatischen Verifikationswerkzeugen, die im Rahmen eines CAD-Frameworks direkt von Hardware-Designern zur Verifikation realistischer (d.h. kommerzieller) Hardware-Entwürfen benutzt werden können. Ein allgemeines Werkzeug für beliebige realistische Schaltungsarten ist mit dem heutigen Stand der Technik nicht erreichbar. Dies liegt daran, daß man meist eine große Lücke zwischen einer Spezifikation auf hoher Abstraktionsebene und einer Hardware-Implementierung auf niedriger Ebene hat. Ein Ausweg dagegen wäre die systematische Verwendung von *schaltungsspezifischen* Theorien, welche die formale Spezifikation und Verifikation durch eine Vielzahl von angepaßten Spezifikationsformalismen, Proof Management und Heuristiken weitgehend unterstützen sollen. Hierzu kann man sich den natürlichen Entwurfsvorgang einer speziellen Schaltungsklasse zunutze machen, um die Verifikation so zu strukturieren, daß sie handhabbar wird (z.B. durch Einfügung von Hierarchien). Einige dieser Schaltungsklassen wären z.B.: Finite-State-Machines, Multiplizierwerke, Floating-Point-Prozessoren, Mikroprozessoren, Signalprozessoren (Systolic Arrays), usw.

2. Stand der Technik

Mikroprozessoren sind vielfältig anwendbare komplexe Schaltungen, die eine besonders gut abgegrenzte Klasse von realistischen Entwürfen bilden. Seit den Anfängen der Forschung im Bereich der Hardware-Verifikation dienten Mikroprozessoren als Maßstab für die Leistung und Güte von Verifikationssystemen [Gord83], [Hunt87], [SrBi89]. Dies liegt vor allem daran, daß die vielseitige Struktur und das Verhalten eines typischen Mikroprozessors einen großen Bereich der Hardware-Verifikationsproblematik abdecken. Es wurden daher eine Vielzahl von prototyp-artigen Entwürfe mit den verschiedensten Verifikationssystemen spezifiziert und auf Korrektheit formal geprüft [Gord83], [Hunt87], [Cohn89a], [Cohn89b], [Joyc89], [Wind90]. Aus technischen als auch aus methodischen Gründen wurden drastisch vereinfachte oder zu abstrakte Mikroprozessor-Modelle (sog. "toy-machines") verifiziert. Diese enthalten höchstens einen Bruchteil der Fähigkeiten moderner realistischer Schaltungen, die heutzutage kommerziell vorhanden sind. Außerdem wurde

mit Ausnahme der Arbeit von P. Windley [Wind90] nur ein spezieller Prozessor betrachtet und verifiziert.

Ziel der vorliegenden Arbeit ist die Entwicklung eines allgemeinen (generischen) Konzepts zur Verifikation von *realistischen RISC-Prozessoren*, das in dem Rahmen eines umfassenden Verifikationswerkzeuges integriert wird. Eine Studie realer CISC-Prozessoren hat gezeigt, daß diese einem zu komplexen und somit sehr unstrukturierten Entwurf, sog. "dirty design", unterliegen. Sie sind daher für eine formale Spezifikation und Verifikation nicht handhabbar. Dies ist einer der Hauptgründe warum man bis jetzt keinen kommerziellen Mikroprozessor formal verifiziert hat.
Die RISC-Philosophie besagt, daß die Hardware-Komplexität auf die Software verlagert wird, um so einen *einfachen* Entwurf zu haben und einen hohen Durchsatz zu erzielen. Wegen dieser verringerten Hardware-Komplexität stellen RISCs, gegenüber CISCs, eine vorteilhafte Ausgangsbasis zur Anwendung von Methoden der formalen Verifikation dar [Bode90]. Die Forderung eines hohen Durchsatzes, meist durch paralleles Pipelining, bringt hingegen andere Probleme mit sich, die den Verifikationsprozeß erschweren.
In diesem Beitrag folgt zunächst eine informelle Beschreibung eines hierarchischen Modells zur Spezifikation und Verifikation von RISC-Prozessoren, dann wird eine mögliche Formalisierung dieses Konzepts vorgestellt und anschließend eine Zusammenfassung und ein Ausblick auf weitere Arbeiten gegeben.

3. Ein RISC-Verifikationsmodell

Bisherige Mikroprozessor-Verifikationen haben fast ausschließlich mikroprogrammierte Prozessoren, also CISCs, behandelt [Hunt85], [Joyc89], [Wind90]. Der Entwurf dieser Art von Prozessoren wird durch eine Hierarchie von sogenannten Interpretern aufgebaut [Ance86]. Dieses Interpreter-Modell (siehe Bild 1) wurde bei manchen Verifikationsarbeiten direkt ausgenutzt [Joyc89], [Wind90], [Aror90] um den Verifikationsprozeß hierarchisch zu strukturieren.

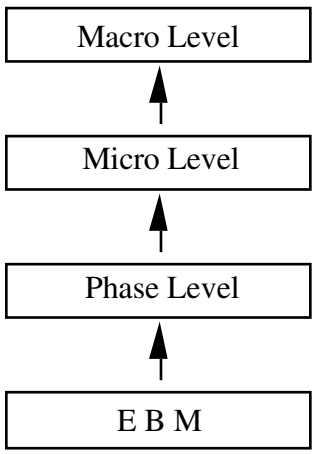

Bild 1: CISC-Interpreter-Modell

Aus früheren Verifikationsarbeiten ist es ersichtlich, daß der größte Verifikationsaufwand beim Übergang vom EBM (Electronic Block Model) zur nächst höheren Abstraktionsebene betrieben wird [Hunt87], [Cohn89b], [Wind90]. Mit dem Interpreter-Modell ist der Abstraktionssprung zwischen

dem EBM und dem Phase-Level nicht mehr so groß. Außerdem braucht man nur noch eine geringe Anzahl von Phasen (4 bis 6), die von der EBM aus auf Korrektheit zu beweisen sind.
RISC-Prozessoren unterliegen einer anderen Entwurfsphilosophie. Man hat z.B. eine festverdrahtete Steuerung, die eine hierarchischen Strukturierung der Verifikation mit diesem Interpreter-Modell nicht mehr anwendbar macht. Daher ist man bestrebt andere Modelle zu entwickeln.
Bei einem RISC-Prozessor ([Kate85], [Kane88], [Furb89], [HePa90], [Bode90]) wird jeder Befehl in einer Anzahl von physikalischen Schritten, sogenannten Pipeline-Stufen (z.B. IF, ID, EX, WB) ausgeführt. Jede Pipeline-Stufenoperation entspricht einen Taktzyklus. Während eines Taktes wird jede Stufenoperation auf mehreren Takt-Phasenoperationen aufgeteilt. Vergleicht man CISC-Mikrobefehle mit RISC-Pipelinestufen und CISC-Phasen mit RISC-Phasen, so findet man also hier wieder eine Art hierarchischer Struktur der Befehlsausführung. Im Gegensatz zu den CISCs, wo man die gleichen physikalischen Phasenoperationen für alle Mikrobefehle hat, sind hier die Phasenoperationen von einer Stufenoperation zur anderen verschieden. Diese sind wiederum von einem Befehl zum anderen unterschiedlich. Das führt dazu, daß man theoretisch eine sehr große Anzahl $N_S = N_b * n_S$ bzw. $N_p = N_b * n_S * n_p$ von möglichen Stufen- bzw. Phasenoperationen hat (N_b steht für die Anzahl der Architekturbefehle, n_S für die Anzahl der Pipeline-Stufen und n_p für die Anzahl der Taktphasen), die von der EBM impliziert werden müssen .
Die RISC-Architektur besitzt zusätzlich viele reguläre Befehle, welche die gleiche Anzahl von Pipelinestufen mit ähnlichen Stufenoperationen benötigen. Diese Befehle werden zu einer Klasse zusammengefaßt. Eine Befehlsklasse entspricht auch die intuitive Zuordnung der Befehle nach ihrer Semantik, z.B.: ALU, FLP, LOAD, STORE und CONTROL. Die große Anzahl der möglichen Stufenoperationen reduziert sich somit zunächst auf ($N_k * n_S$). Zusätzlich verwenden manche Befehlsklassen die gleichen Stufenoperationen gemeinsam, und es wird außerdem nicht immer die volle Pipelinetiefe benötigt. Somit hat man in der Realität eine kleinere Anzahl von Stufenoperationen ($N_S « (N_k * n_S)$).
Die Anzahl der Phasenoperationen ($N_p = N_S * n_p$) entspricht nicht direkt den Aufwand der Spezifikation und Verifikation der Phasenoperationen. In der Tat, gibt es Stufenoperationen, die für einige Register-Transfers den vollen Taktzyklus oder nur ein paar Phasen benötigen. Die verbleibenden (leeren) Phasenoperationen werden durch eine Art *Id*-Funktion spezifiziert, die zu keinen Zustandsänderungen führen sollen.

Zusammenfassend wird der gesamte Verifikationsvorgang (siehe Bild 2) hierarchisch in folgender Schritte aufgeteilt :

EBM → Phase Level (N_p)	(1):	das EBM impliziert alle Phasenoperationen
Phase Level (N_p) → Stage Level (N_S)	(2) :	die Phasenoperationen implizireren alle Stufenoperationen
Stage Level (N_S) → Class Level (N_k)	(3) :	die Stufenoperationen implizieren alle Befehlsklassen

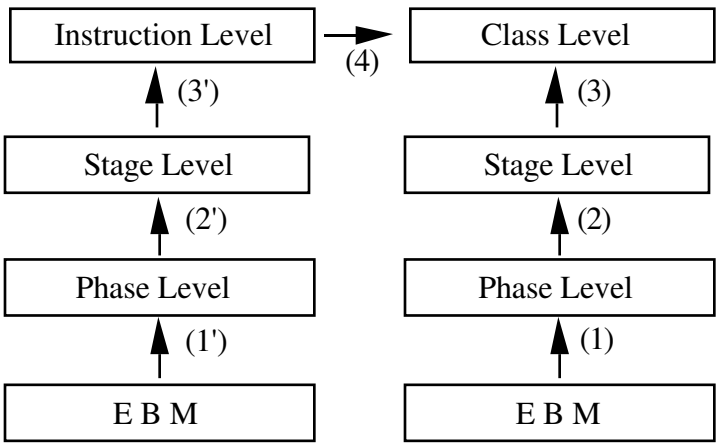

Bild 2: RISC-Interpreter-Modell

Das Hauptverifikationsziel der Korrektheit der Architekturbefehle, wird anschließend durch Instanziierung der Schritte 1 bis 3 für die einzelnen Befehlen erreicht (siehe Schritte 1' bis 3' in Bild 2). Zur Vollständigkeit wird noch bewiesen, daß die Befehlsklassen alle Architekturbefehle abstrahieren: Instruction Level (N_b) → Class Level (N_k) (4)

4. Formale Spezifikation und Verifikation

Die Instruction-, Class-, Stage- und Phase-Level stellen abstrakte Interpreter dar. Allgemein ist ein Interpreter ein Zustandsübergangssystem (state transition system), das hauptsächlich aus einer Menge von Zuständen und Zustandsübergangsfunktionen besteht. Der *Zustand* des Interpreters wird mit einem n-Tupel von Zustandskomponenten dargestellt, welche die in der betrachteten Abstraktionsebene sichtbaren Speicherelemente bilden (z.B. [RF, PC, I-MEM, D-MEM] für den Instruction-Level). Die *Zustandsübergangsfunktionen* geben die Semantik der Befehle bzw. der Operationen wieder. Als Beispiel seien auf dem Instruction-Level die Spezifikation des ADD-Befehls und auf dem Stage-Level die Spezifikation der EX-Stufenoperation der ALU-Befehlsklasse betrachtet:

$\forall t.\ ADD(\ldots, PC, RF[rd], \ldots)(t+1) = (\ldots, PC+4, RF[rs1] + RF[rs2], \ldots)(t)$

$\forall t.\ EX_A(\ldots, ALUout, \ldots)(t+1) = (\ldots, A\ op\ B, \ldots)(t)$

Das EBM ist kein Interpreter mehr und stellt eine Struktur dar. Diese wird durch eine Hierarchie von Konjunktionen von Prädikaten beschrieben, welche die verschiedenen Hardware-Komponenten und deren Verbindungen zueinander wiedergeben. Dies wird wie folgt formal spezifiziert:

EBM (…) =
∃ . . .
(DataPath …) ∧
(ControlUnit …).

DataPath (. . .) =
∃ . . .
(ALU . . . .) ∧ (RegFile . . .) ∧ (Shifter . . .) ∧ . . .

In einer Pipeline-Verarbeitung hat man zu jedem Zeitpunkt eine bestimmte Anzahl von Befehlen in paralleler Bearbeitung, die sich gegenseitig stören können. Um dieses Problem der Parallelität zu handhaben, wird die Verifikation in zwei Schritten aufgeteilt:

1. $SW_Ass, EBM, Phase\ Level, Stage\ Level, Class\ Level \vdash \neg\ Conflict_Pred$

2. $\neg\ Conflict_Pred, EBM, Phase\ Level, Stage\ Level, Class\ Level \vdash EBM \Rightarrow Instruction\ Level$

Im ersten Teil wird gezeigt, daß, ausgehend von der Spezifikation des Interpreter-Modells und von gegebenen Softwareannahmen, wie z.B. die Anzahl der "delay slots", die parallele Verarbeitung von Befehlen zu keinen Konflikten (Hazards) führt. Es gibt drei Arten von Hazards ([Bode90], [HePa90], [Fubr89], [Milu89], [MüSc91]): Struktur-Hazards (Resourcen-Konflikte), Data-Hazards (Befehlsabhängigkeiten) und Control-Hazards (Unterbrechung des linearen Befehlsflußes). Diese unterschiedlichen Konfliktarten werden explizit durch Prädikate beschrieben, die dann auf Nichterfüllung bewiesen werden müssen. Als Beispiel eines solchen Prädikates sei der für Struktur-Hazards:

$$Struct_Pred\ (r: Resource) := \exists S_i, S_j: Stage.\ (\ S_type(S_i) \neq S_type(S_j)\ ) \wedge S_used_Pred(S_i, r) \wedge S_used_Pred(S_j, r)$$

Der zweite Teil der Verifikation besteht darin zu beweisen, daß unter Annahme eines "störungsfreien" Ablaufes der Befehlsbearbeitung und ausgehend von der Spezifikation des Interpreter-Modells die Semantik der Architekturbefehle korrekt implementiert ist. Als Beispiel dazu wird der Verifikationsschritt $Stage\ Level \Rightarrow Class\ Level$ wie folgt formalisiert:

$$\forall K_i\ .\ (\ IF_i \wedge\ ID_i \wedge EX_i\ \wedge WB_i\ ) \rightarrow Sem(K_i)$$

5. Zusammenfassung und Ausblick

Ziel dieser Arbeit ist es ein generisches Modell zur formalen Verifikation von realistischen Mikroprozessoren zu entwickeln. Bisherige Verifikationsarbeiten haben nur toy-machines verifiziert. Dies liegt hauptsächlich daran, daß reale Prozessoren zu komplex sind und einem dirty design unterliegen. In Hinsicht auf eine formale Verifikation weisen RISC-Prozessoren mit ihrem einfacheren Entwurf viele Vorteile gegenüber CISCs. Wegen der unterschiedlichen Entwurfsphilosophie bei RISCs sind existierende Verifikationsverfahren, wie das Interpreter-Modell, nicht mehr anwendbar. Als Lösungsansatz wurde ein hierarchisches Verifikationsmodell vorgestellt, das den natürlichen Eigenschaften eines allgemeinen RISC-Prozessors entspricht. Es wurde eine mögliche Formalisierung der Spezifikation und Verifikation gegeben. Die Verifikation wird in zwei Teilen durchgeführt. Zuerst wird die Konfliktfreiheit der parallelen Pipeline-Verarbeitung gezeigt, dann wird die semantische Korrektheit der Architekturbefehle unter Annahme einer störungsfreien Bearbeitung bewiesen.

In weiteren Arbeiten soll nun das entwickelte Modell verallgemeinert und als generische Theorie in Logik höherer Ordnung (HOL) implementiert werden. Weiter wird als Anwendungsbeispiel ein in CADENCE implementierter, realer RISC-Prozessors anhand dieses Modells verifiziert.

6. Literatur

Ance86	Anceau, F. : The Architecture of Microprocessors; Addison-Wesley Publishing Company, 1986.
Aror90	Arora, T. : The Formal Verification of the VIPER Microprocessor: EBM to Phase, Phase to Microcode Level; Master's thesis, Univrsity of California, Davis, 1990.
Bode90	Bode, A. : RISC-Architekturen; BI-Wiss.-Verlag; Mannheim; 1990.
Cohn89a	Cohn, A. : A Proof of the Viper Microprocessor: The First Level; In: VLSI Specification Verification and Synthesis, Eds. G.Birtwistle and P.A. Subrahmanyam, Kluver, 1989.
Cohn89b	Cohn, A. : Correctness Properties of the Viper Block Model: The Second Level; In: Current Trends in Hardware Verification and Automated Theorem Proving, Eds. G. Birtwistle and P.A. Subrahmanyam, Springer, 1989.
Furb89	Furber, S. : VLSI RISC Architecture and Organization; Electrical engeneering and electronics, Dekker, New York, 1989.
Gord83	Gordon, M. : Proving a Computer Correct; Technical Report No. 42, Computer Laboratory, University of Cambridge, 1983.
HePa90	Hennessy, J., Patterson, D. : Computer Architecture A Quantitative Approach; Morgan Kaufmann Publishers, Inc. San Mateo, California, 1990.
Hunt87	Hunt, W. : The Mechanical Verification of a Microprocessor Design; In: From HDL Description to Guaranteed Correct Circuit Designs, Ed. D. Borrione, North-Holland, 1987.
Joyc89	Joyce, J. : Multi-Level Verification of Microprocessor-Based Systems; PhD thesis, Cambridge University, December 1989.
Kane88	Kane, G.: MIPS RISC Architecture; Englewood Cliffs, NJ, Prentice Hall, 1988.
Kate84	Katevenis, M. : Reduced Instruction Set Computer Architectures for VLSI; MIT Press, Cambridge, Massachusetts, 1984.
Milu89	Milutinovic, V. : High Level Language Computer Architecture; Computer Science Press, Inc., 1989.
MüSc91	Müller-Schloer, C. , Schnitter, E. : RISC-Workstation Architekturen; Springer Verlag, 1991.
SrBi89	Srivas, M., Bickford, M. : Verification of a Pipelined Microprocessor Using Clio; In: Hardware Specification, Verification and Synthesis: Mathmatical Aspects, Eds. M. Leeser and G. Brown, Springer, 1990.
Wind90	Windley, P.: The Formal Verification of Generic Interpreters; PhD thesis, University of California, Davis, Division of Computer Science, July 1990. Research Report CSE-90-22.