Formale Verifikation des Befehlssatzes eines in SystemC modellierten Mikroprozessors

Daniel Große Ulrich Kühne Rolf Drechsler
Institut für Informatik, Universität Bremen, 28359 Bremen
{grosse, ulrichk, drechsle}@informatik.uni-bremen.de

Abstract: In dieser Arbeit wird ein in SystemC modellierter Mikroprozessor inklusive seines Befehlssatzes vollständig formal verifiziert. Dadurch kann die korrekte Abarbeitung von Assemblerprogrammen auf dem Mikroprozessor sichergestellt werden. Durch die Verwendung eines SystemC Modells wird der kombinierte Hardware/Software Entwurfsprozess drastisch vereinfacht.

1 Einleitung

Im industriellen Umfeld erfolgt der Entwurf einer Schaltung sehr oft mittels spezieller Hardwarebeschreibungssprachen, wie z.B. VHDL oder Verilog. Auf Grund der Anforderungen heutiger System-on-Chip (SoC) Entwürfe, bei denen der Anteil an Systemfunktionen, die in Software implementiert werden, stetig zunimmt, werden in der Praxis für Systemmodelle C-artige Sprachen eingesetzt. Um die entstehende Lücke zu VHDL/Verilog zu schließen, wurden verschiedene Erweiterungen von C/C++ vorgeschlagen, die es ermöglichen, Hardware zu modellieren. In diesem Zusammenhang wurde die Systembeschreibungssprache SystemC [SC2] eingeführt. SystemC ist eine C++-Klassenbibliothek, frei verfügbar, unterstützt die Modellierung auf unterschiedlichsten Abstraktionsebenen und stellt einen zyklengenauen Simulator zur Verfügung.

Verschiedene Prozessoren wurden bereits in SystemC modelliert (siehe z.B. [BGGR05]). Im Folgenden wird der Mikroprozessor `Zykluno` betrachtet, der in [BDM05] beschrieben wurde. Ein SystemC Modell für diesen Prozessor wurde in [GKG+05] eingeführt. In der vorliegenden Arbeit wird der komplette Befehlssatz dieses SystemC Modells formal verifiziert. Dadurch steht in der Systembeschreibungssprache SystemC an der Hardware/-Software Schnittstelle ein beweisbar korrekt arbeitender Mikroprozessor zur Verfügung. Als Verifikationsmethode kommt Bounded Model Checking zum Einsatz.

2 Der Mikroprozessor `Zykluno`

Der Gesamtaufbau von `Zykluno` ist in Abbildung 1 skizziert. Bei `Zykluno` handelt es sich um einen Vertreter der Harvard-Architektur. Programm- und Datenspeicher haben jeweils eine Wortbreite von 16 Bit und eine Größe von 4 bzw. 128 KByte. Ein Maschinenbefehl von `Zykluno` umfasst 16 Bit. Aus Platzgründen wird hier lediglich ein Überblick über die 5 Befehlsklassen von `Zykluno` gegeben ohne näher auf Details einzugehen:

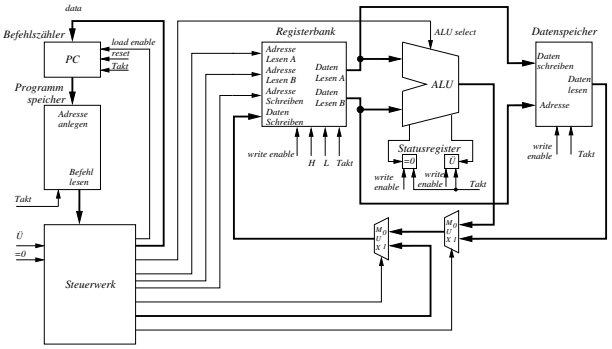

Abbildung 1: Blockschaltbild des Rechners `Zykluno`

- 6 Lade- und Speicherbefehle (Datentransfer zwischen Registerbank und Datenspeicher oder Ein-/Ausgabegerät, Laden des High- bzw. Low-Byte eines Registers mit einer Konstanten)
- 8 arithmetische Befehle (Addition/Subtraktion mit und ohne *carry*, Rotation und Shift links/rechts)
- 8 logische Befehle (Bitweise Negation, Bitweise Exklusiv-Oder-Verknüpfung, Konjunktion, Disjunktion zweier Operanden sowie Maskierung, Invertierung, Löschen oder Setzen eines einzelnen Bits eines Operanden)
- 5 Sprungbefehle (unbedingter Sprung, bedingter Sprung bei gesetztem/gelöschtem *carry*- oder *zero*-Bit)
- 5 sonstige Befehle (Stapelspeicher-Operationen Push und Pop, Programmhalt, Unterprogrammsprung, Rücksprung von Unterprogramm)

Für weitere Details zum Aufbau von `Zykluno` bzw. bezüglich des SystemC Modells wird auf [BDM05] und [GKG+05] verwiesen.

3 Formale Verifikation des SystemC Modells von `Zykluno`

3.1 Bounded Model Checking für SystemC

Für die formale Verifikation von SystemC Beschreibungen wurde in [GD05] der Eigenschaftsprüfer *CheckSyC* vorgestellt. Das Werkzeug ermöglicht es für SystemC Beschreibungen auf der Register-Transfer-Ebene temporale Eigenschaften nachzuweisen bzw. zu widerlegen. Die Eigenschaftsprüfung selbst beruht auf dem Prinzip des *Bounded Model Checking* (BMC) wie es in [WTSF04] beschrieben wurde. Beim BMC werden temporale Eigenschaften für eine Schaltkreisbeschreibung über einem endlichen Zeitfenster nachgewiesen. Eine Eigenschaft besteht dabei aus zwei Teilen: dem Annahmeteil (*assume part*) und dem Beweisteil (*prove part*). Wenn die Aussagen des Annahmeteils wahr

Tabelle 1: Laufzeiten für die Eigenschaftsprüfung der einzelnen Komponenten

Komponente	Anzahl Eigenschaften	Summe der CPU-Zeiten (Sek)
Registerbank	4	1,03
Befehlszähler	3	0,08
Steuerwerk	11	0,23
Datenspeicher	2	0,49
Programmspeicher	2	0,48
ALU	17	4,41

werden, dann sollen auch die Aussagen des Beweisteils gelten. Für den Beweis einer solchen Eigenschaft werden zunächst die Grenzen des betrachteten Zeitintervalls bestimmt. Im nächsten Schritt wird der Schaltkreis entsprechend der Breite des Zeitintervalls abgerollt, d.h. die Zustandsvariablen des aktuellen Zeitpunktes werden durch die zugehörigen Übergangsfunktionen des vorherigen Zeitpunktes ersetzt. Dadurch wird aus der initial sequentiellen Fragestellung eine kombinatorische Fragestellung erzeugt. Anschließend wird der abgerollte Schaltkreis zusammen mit der Eigenschaft in ein Erfüllbarkeitsproblem (SAT) überführt. Ist dieses nicht erfüllbar, so gilt die Eigenschaft.

3.2 Verifikation des Befehlssatzes von `Zykluno`

In diesem Abschnitt wird die formale Verifikation des Befehlssatzes für das SystemC Modell von `Zykluno` beschrieben. Zunächst wurden Eigenschaften für jede einzelne Komponente von `Zykluno` (siehe auch Abbildung 1) entwickelt. Beispielsweise wurde für das Steuerwerk bewiesen welche Steuerleitungen unter welchen Befehlen zu setzen sind. Insgesamt konnte dadurch die Korrektheit der einzelnen Blöcke nachgewiesen werden. Die im Folgenden vorgestellten Experimente wurden auf einem Athlon XP 2800 mit 1 GByte Hauptspeicher durchgeführt. Tabelle 1 fasst die Ergebnisse für die Blockverifikation zusammen. Um ein handhabbares formales Modell von `Zykluno` erzeugen zu können, wurden die Speicher in der Größe reduziert. Die betrachteten Komponenten sind in der ersten Spalte aufgeführt. Die zweite Spalte zeigt die Anzahl der Eigenschaften, die für die jeweilige Komponente spezifiziert worden sind. Die letzte Spalte gibt die CPU-Zeit in Sekunden an, die für den Beweis der jeweils für diese Komponente spezifizierten Eigenschaften insgesamt benötigt wird. Man erkennt, dass die Eigenschaften mit Hilfe von BMC sehr schnell bewiesen werden konnten. Zeitlich am aufwendigsten sind die Eigenschaftsbeweise für die ALU.

Aufbauend auf der Blockverifikation wurden im nächsten Schritt die einzelnen Befehle von `Zykluno` und deren Auswirkungen für die gesamte CPU betrachtet. Aus Platzgründen kann die Verifikation des kompletten Befehlssatzes nicht diskutiert werden. Exemplarisch wird die Verifikation des Additionsbefehls ohne Übertrag (ADD) beschrieben. Der ADD-Befehl hat die in Tabelle 2 dargestellten Eigenschaften.

Eine an PSL angelehnte Formulierung der Eigenschaft für den ADD-Befehl ist in Abbildung 2 dargestellt. Zunächst werden aus dem betrachteten Befehlswort der Opcode

Tabelle 2: Eigenschaften des ADD-Befehls

Assemblernotation: `ADD` $R[i]$, $R[j]$, $R[k]$

Funktion: addiert die Registerinhalte $R[j]$ und $R[k]$ und schreibt das Ergebnis in Register i

Befehlsformat:

15 ... 11	10 9 8	7 6	5 4 3	2 1 0
0 0 1 1 1	*bin(i)*	- -	*bin(j)*	*bin(k)*

```
1  OPCODE := instr[15:11];
2  Ri_A := instr[10:8];
3  Rj_A := instr[5:3];
4  Rk_A := instr[2:0];
5  Rj := reg.reg[Rj_A];
6  Rk := reg.reg[Rk_A];
7
8  property ADD
9  always
10  // assume part
11  ( reset = 0 && OPCODE = "00111" &&
12    Ri_A > 1 && Rj_A > 1 && Rk_A > 1 )
13  ->
14  // prove part
15   next(
16    (reg.reg[prev(Ri_A)] + (65536 * stat.C)
17      = prev(Rj) + prev(Rk))
18     && ((reg.reg[prev(Ri_A)] = 0) <-> (stat.Z = 1))
19
20     // no side effects
21     && ((prev(Ri_A) != 2) -> reg.reg[2] = prev(reg.reg[2]) )
22     && ((prev(Ri_A) != 3) -> reg.reg[3] = prev(reg.reg[3]) )
23     && ((prev(Ri_A) != 4) -> reg.reg[4] = prev(reg.reg[4]) )
24     && ((prev(Ri_A) != 5) -> reg.reg[5] = prev(reg.reg[5]) )
25     && ((prev(Ri_A) != 6) -> reg.reg[6] = prev(reg.reg[6]) )
26     && ((prev(Ri_A) != 7) -> reg.reg[7] = prev(reg.reg[7]) )
27   );
```

Abbildung 2: Eigenschaft `ADD` für den Befehl Addition ohne Übertrag

und die drei vom ADD-Befehl benötigten Register Variablen mit aussagekräftigen Namen zugewiesen, um die Lesbarkeit der Eigenschaft zu erhöhen (Zeile 1-6). Dann wird die Eigenschaft ADD definiert (Zeile 8). In den Zeilen 11 und 12 werden die Annahmen beschrieben unter denen der Beweisteil (Zeile 15-27) gelten soll: Unter den Bedingungen, dass kein Reset vorliegt (Zeile 11), es sich um den ADD-Befehl handelt (Zeile 11) und nicht die Register $R[0]$ bzw. $R[1]$ (dabei handelt es sich um Spezialregister) adressiert werden (Zeile 12), sollen einen Takt später die folgenden Aussagen gelten: Register $R[i]$

Tabelle 3: Laufzeiten für die Eigenschaftsprüfung der verschiedenen Befehlsklassen

Befehlsklasse	Anzahl Eigenschaften	Summe der CPU-Zeiten (Sek)
Arithmetische Befehle	8	186,30
Logische Befehle	8	32,71
Lade- und Speicherbefehle	6	15,16
Sprungbefehle	5	6,68
Sonstige Befehle	5	7,14

(=reg . reg[**prev**(Ri_A)]) enthält die Summe von $R[j]$ und $R[k]$ (Zeile 17), das Carry (stat .C) wird korrekt berechnet (Zeile 16) und das Zero Bit (stat .Z) gesetzt, genau dann wenn die Summe null ist (Zeile 18). Ferner sollen alle Register unverändert bleiben, in die nicht geschrieben wird (Zeile 21-26). Insgesamt wird durch den Beweis dieser Eigenschaft für das SystemC Modell gezeigt, dass der ADD-Befehl für beliebige Wahl der Parameter korrekt arbeitet, d.h. insbesondere das korrekte Ergebnis berechnet und dieses nur in das Zielregister schreibt, ohne dabei den Inhalt der anderen Register zu modifizieren.

Für den kompletten Befehlssatz von `Zykluno` wurden Eigenschaften analog formuliert. Tabelle 3 zeigt die erzielten Resultate. Dabei ist in jeder Zeile die benötigte CPU-Zeit für den Beweis aller Eigenschaften der gerade betrachten Befehlsklasse angegeben. Die Tabelle zeigt, dass der komplette Befehlssatz von `Zykluno` in weniger als 5 CPU-Minuten verifiziert werden konnte.

Literatur

[BDM05] B. Becker, R. Drechsler und P. Molitor. *Technische Informatik — Eine Einführung*. Pearson Education Deutschland, Februar 2005.

[BGGR05] A. Braun, T. Grosser, J. Gerlach und W. Rosenstiel. Entwicklung einer SystemC-basierten Simulationsumgebung für einen 8-Bit-RISC-Mikrokontroller. In *GI/ITG/GMM-Workshop, Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2005.

[GD05] D. Große und R. Drechsler. *CheckSyC*: An Efficient Property Checker for RTL SystemC Designs. In *IEEE International Symposium on Circuits and Systems*, Seiten 4167–4170, 2005.

[GKG+05] D. Große, U. Kühne, C. Genz, F. Schmiedle, B. Becker, R. Drechsler und P. Molitor. Modellierung eines Mikroprozessors in SystemC. In *GI/ITG/GMM-Workshop, Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2005.

[SC2] Synopsys Inc., CoWare Inc., and Frontier Design Inc., http://www.systemc.org. *Functional Specification for SystemC 2.0*.

[WTSF04] K. Winkelmann, H.-J. Trylus, D. Stoffel und G. Fey. A Cost-efficient Block Verification for a UMTS Up-link Chip-rate Coprocessor. In *Design, Automation and Test in Europe*, Jgg. 1, Seiten 162–167, 2004.