Ein Softwaresimulator auf Basis von COM/OLE für kleine eingebettete Systeme

Tino Rachui, Egmont Woitzel*, Hagen Ploog
Universität Rostock
Fachbereich Elektrotechnik und Informationstechnik
Institut für Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Straße 31; 18119 Rostock-Warnemünde
Email: tino.rachui@etechnik.uni-rostock.de

* FORTecH Software Entwicklungsbüro Dr.-Ing. E. Woitzel
Joachim-Jungius-Straße 9 · 18059 Rostock
Email: EWoi@fortech.de

Kleine eingebettete Systeme finden in zunehmenden Maße Verbreitung. Den Kern dieser Systeme bilden in vielen Fällen spezielle 4 bit oder 8 bit Mikroprozessoren, die auf Basis einer Hardwarebeschreibungsprache wie VHDL entwickelt werden. Im vorliegenden Beitrag wird eine Simulations- und Testumgebung für die Softwareentwicklung solcher Mikroprozessoren beschrieben, die dem Standard moderner Entwicklungstools entspricht. Ein wesentlicher Vorteil dieser Entwicklungsumgebung liegt in der schnellen Anpassungsfähigkeit an verschiedene Mikroprozessorarchitekturen. Realisiert wird dies u. a. durch den Einsatz moderner Komponentensoftwaretechnologien wie COM/OLE.

Motivation

Kleine eingebettete Systeme finden im Umfeld von intelligenten Sensoren und Aktoren sowie Smart Cards immer größere Einsatzgebiete. Die Charakteristik solcher Single-Chip-Systeme, liegt darin, daß sie zumeist aus einem reduzierten oder minimalen Mikroprozessorkern und nur wenigen peripheren Komponenten bestehen. In vielen Fällen beschränken sich die dabei zum Einsatz kommenden Prozessorarchitekturen auf eine Verarbeitungsbreite von 4 oder 8 Bit. Erst die Möglichkeit, solch kleine Systeme programmieren zu können, sichert die Anwendbarkeit desselben Chips für eine größere Anzahl von Einsatzfällen und damit die für einen ökonomischen Einsatz sinnvollen Stückzahlen.

Die Entwicklung derartiger Systeme erfolgt typisch mit Hilfe einer Hardwarebeschreibungssprache wie z. B. VHDL. Dieser Ansatz bringt zwei wesentliche Vorteile mit sich. Zum einen verringert sich die benötigte Entwicklungszeit, um eine neue Mikroprozessorarchitektur zur entwerfen und zum anderen ist es erheblich einfacher, eine in VHDL-Beschreibung vorliegende Mikroprozessorarchitektur an verschiedene Anwendungsfälle zu adaptieren.

Der Entwicklungsprozeß eingebetteter Systeme auf Basis minimaler Mikroprozessorarchitekturen

Der Entwicklungsprozeß kleiner eingebetteter Systeme läßt sich aus zwei Blickwinkeln heraus betrachten. Einerseits aus der Sicht des Chipentwicklers und andererseits aus der Sicht des Applikationsentwicklers, der das konkrete eingebettete System implementiert. Der letztgenannte soll im weiteren als Benutzer bezeichnet werden. In den seltensten Fällen wird

der Benutzer selbst auch der Chipentwickler sein. Mittelständische Unternehmen werden mit dieser Aufgabe einen externen Dienstleister beauftragen.

Beide Gruppen sind üblicherweise an der Entwicklung eines konkreten eingebetteten Systems beteiligt. Der Ausgangspunkt für die Entwicklung eines geeigneten Systems ist die Spezifikation des gewünschten Einsatzbereiches durch den Benutzer. Anhand dieser Zielvorgabe definiert der Entwickler dann mit der Zielvorgabe eines optimalen Verhältnisses zwischen Leistungsfähigkeit und Ressourcenbedarf einen aus seiner Sicht geeigneten Chip. Dieser wird typisch durch die Parametrisierung eines generischen Prozessorkerns und dessen Ergänzung um applikationsspezifische Peripheriekomponenten erzeugt. Die Anzahl der zur Verfügung stehenden Register hat z. B. unmittelbare Auswirkungen auf die benötigte Chipfläche, aber gleichzeitig auch auf die erreichbare Rechengeschwindigkeit. Der Entwickler hat während dieser Phase durch eine Vielzahl von VHDL-Entwicklungstools die Möglichkeit, seinen Entwurf hinsichtlich der Zielkriterien zu optimieren.

Üblicherweise verfügt der Benutzer nicht über eine VHDL-Entwicklungsumgebung. Die Möglichkeiten dieser Systeme sind ihm daher nicht zugänglich. Der Chipentwickler muß deshalb das System um eine Entwicklungsumgebung ergänzen. Diese sollte bereits vor der Verfügbarkeit der realen Hardware dem Benutzer die Möglichkeit zur Entwicklung der Software für die Zielapplikation bieten. Nur so lassen sich Entwurfs- und Leistungsprobleme der gewählten Konfiguration rechtzeitig erkennen. Die Simulation des Zielsystems kann allerdings den Test auf der realen Hardware nicht ersetzen. Die Entwicklungsumgebung muß daher sowohl eine Softwaresimulation als auch ein In-Circuit-Debugging unterstützen.

Architektur der Entwicklungsumgebung

Betrachtet man den beschriebenen Gesamtentwicklungsprozeß, so ist es aus Benutzersicht notwendig, daß die Entwicklungsumgebung die Zielarchitektur simulieren kann, komfortable Werkzeuge zum Editieren und Testen von Programmquellcode und eine integrierte Projektverwaltung mitbringt. Aufgrund der sehr beschränkten Ressourcen der Zielsysteme wird die Softwareentwicklung typisch auf Basis einer Assemblersprache erfolgen. Dennoch ist z. B. ein Einzelschritt-Betrieb auf Quelltextniveau sinnvoll und Stand der Technik.

Demgegenüber steht das Interesse des Chipentwicklers, möglichst wenig Aufwand in die Anpassung der Entwicklungsumgebung an eine spezielle Architektur zu investieren. Dieser bestimmt unmittelbar die Kosten für seine Dienstleistung, da sich die Kosten für den Entwicklungsaufwand schon mit einem Projekt amortisieren müssen.

Daraus ergibt sich, daß eine derartige Entwicklungsumgebung unter dem Aspekt maximaler Wiederverwendbarkeit entworfen werden muß. Grundlage dafür ist die Annahme eines abstrakten Rechnersystems, das von der Entwicklungsumgebung kontrolliert wird. Das Rechnersystem kann softwaretechnisch durch Objekte abgebildet werden, die CPU, Speicher und Peripherie repräsentieren. Die Anzahl und Größe bzw. Wortbreite der Speicherräume können durch das CPU-Objekt spezifiziert werden, so daß nahezu beliebige Prozessor-architekturen abgebildet werden können. Peripheriebaugruppen können in erster Näherung als spezielle Speicherräume aufgefaßt werden.

Ein solcher objektorientierter Ansatz erlaubt die Implementierung der Betriebsweisen Simulation bzw. In-Circuit-Debugging durch den Austausch der entsprechenden Objekt-implementierungen. Gerade unter diesem Aspekt bietet sich die Realisierung der Objekte

durch Komponenten an, da so der Austausch der Implementierung ohne Neuübersetzung des gesamten Systems möglich wird. Die im Rahmen dieses Beitrags vorgestellte Entwicklungsumgebung FLEXS (Flexibles Mikroprozessorentwicklungs- und Simulationssystem) folgt diesem Entwurf.

Als Zielplattform für FLEXS wurde Windows NT gewählt. Die Hauptgründe für diese Entscheidung waren unter anderem die zunehmende Akzeptanz von Windows NT im industriellen Umfeld, das leistungsfähige I/O-System von Windows NT und die Verfügbarkeit einer leistungsfähigen Infrastruktur zur Entwicklung moderner Komponentensoftware in Form von COM/OLE [PAP96] [KAT98] [MICH98/99].

Die FLEXS Shell

Alle von der konkreten Prozessorarchitektur und der Betriebsart unabhängigen Funktionen wurden in der FLEXS-Shell implementiert, die unter Verwendung der COM/OLE-Technologie die Funktionalität aller anderen Komponenten importiert und integeriert. Das Benutzerinterface wurde in Aussehen und Handhabung typischen PC-basierten Softwareentwicklungsumgebungen angenähert, so daß nur eine kurze Einarbeitungszeit für den Umgang mit dem System notwendig ist. Abbildung 1 zeigt die Benutzeroberfläche.

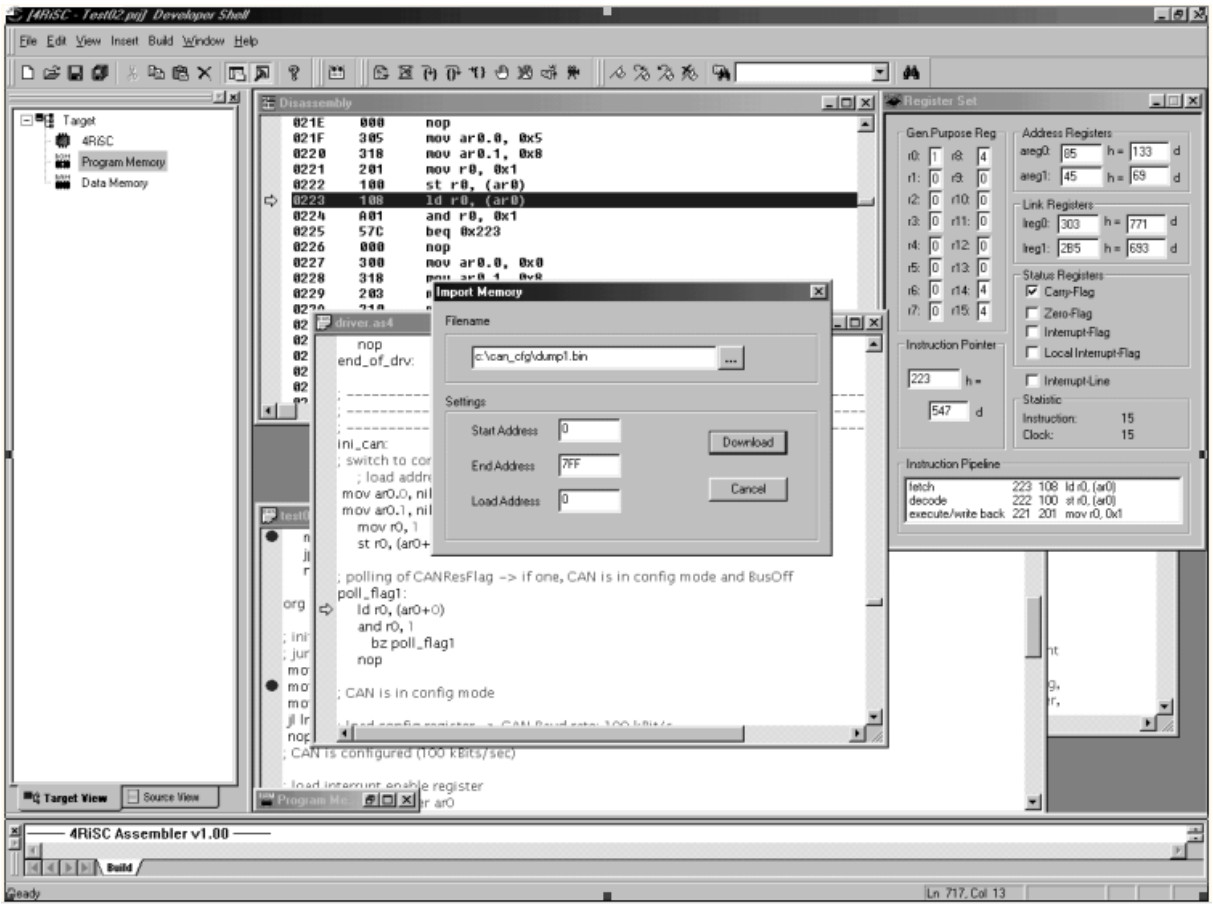

Abbildung 1: FLEXS-Bedienoberfläche

Im linken Teil des Fensters ist das Projekt zu sehen, hier die Einstellungen zum Zielsystem. Rechts ist ein Fenster mit der Visualisierung des Zustands des Cores zu finden. Hier können z. B. auch Registerinhalte verändert werden. In der Mitte des Fensters ist eine Dialogbox für den Import eines Speicherimages zu sehen, darunter verschiedene Fenster mit Quelltext

(Editor und Source-Level-Debugging) bzw. rückübersetztem Assemblercode. Am unteren Rand liegt der Ausgabebereich für Meldungen der externen Tools wie dem Assembler. Nicht abgebildet ist die Anzeige eines Speicherbereichs.

Die Shell implementiert unter Zuhilfenahme der Dienstleistungen der zielsystemabhängigen Komponenten folgende Kernfunktionalität:

- Einfache Projektverwaltung für Zielsystem und Quelltext
- Quelltexteditor mit syntaxabhängiger Colorierung
- Automatische Neuübersetzung und Download des Projekts nach Quelltextänderungen
- Source-Level-Debugging (Einzelschrittbetrieb, Animation, Unterbrechungspunkte)
- Anzeige und Manipulation des CPU-Zustands
- Anzeige und Manipulation von Speicherinhalten
- Rückübersetzung des Programmspeicherinhalts
- Laden und Sichern von Speicherinhalten und Systemzustand

Diese Funktionalität wird dem Benutzer unter derselben Oberfläche im Simulationsbetrieb und während des In-Circuit-Debuggings angeboten.

Komponenten für den Simulatorbetrieb

Im Simulatorbetrieb wird das Zielsystem innerhalb des Entwicklungsrechners in Software nachgebildet. Abbildung 2 zeigt eine vereinfachte Darstellung der beteiligten Komponenten.

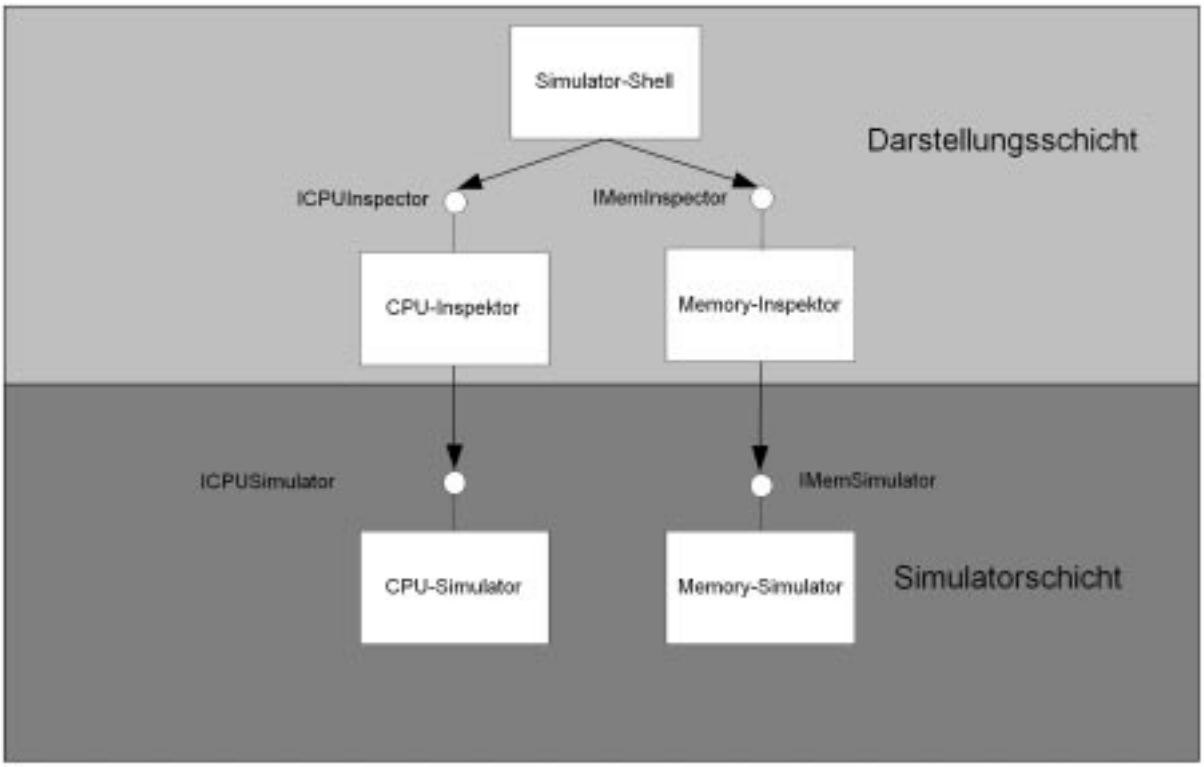

Abbildung 2: FLEXS-Simulatorbetrieb

Das System ist in zwei Schichten strukturiert. Die Darstellungsschicht beinhaltet die FLEXS-Shell und die sogenannten Inspektor-Komponenten. Die Komponenten dieser Schicht übernehmen ausschließlich visualisierende Funktionen und werden im Simulator- und In-Circuit-betrieb verwendet. Der CPU-Inspektor übernimmt dabei sowohl die Anzeige des

CPU-Zustands als auch die Rückübersetzung des Maschinencodes sowie die Verbindung zu den externen Tools. Die Komponente Memory-Inspektor erfüllt dieselbe Aufgabe für die Visualisierung eines Speicherbereichs. Je nach Architektur des Zielsystems können mehrere unabhängige Speicherräume existieren. Die Shell kann daher mit einer beliebigen Anzahl von Memory-Inspektoren gleichzeitig umgehen.

Die eigentliche Simulation des Systems erfolgt in der darunterliegenden Schicht. Die Komponente CPU-Simulator besitzt hier eine Schlüsselposition. Sie bildet taktgenau das Verhalten der Hardware-CPU nach. Dazu kann sie die Memory-Simulator-Komponenten auch direkt aufrufen.

Komponenten für den In-Circuit-Betrieb

Soll statt der Simulation mit der realen Hardware kommuniziert werden, wird die Simulatorschicht vollständig durch die Adapterschicht ersetzt, wie Abbildung 3 zeigt.

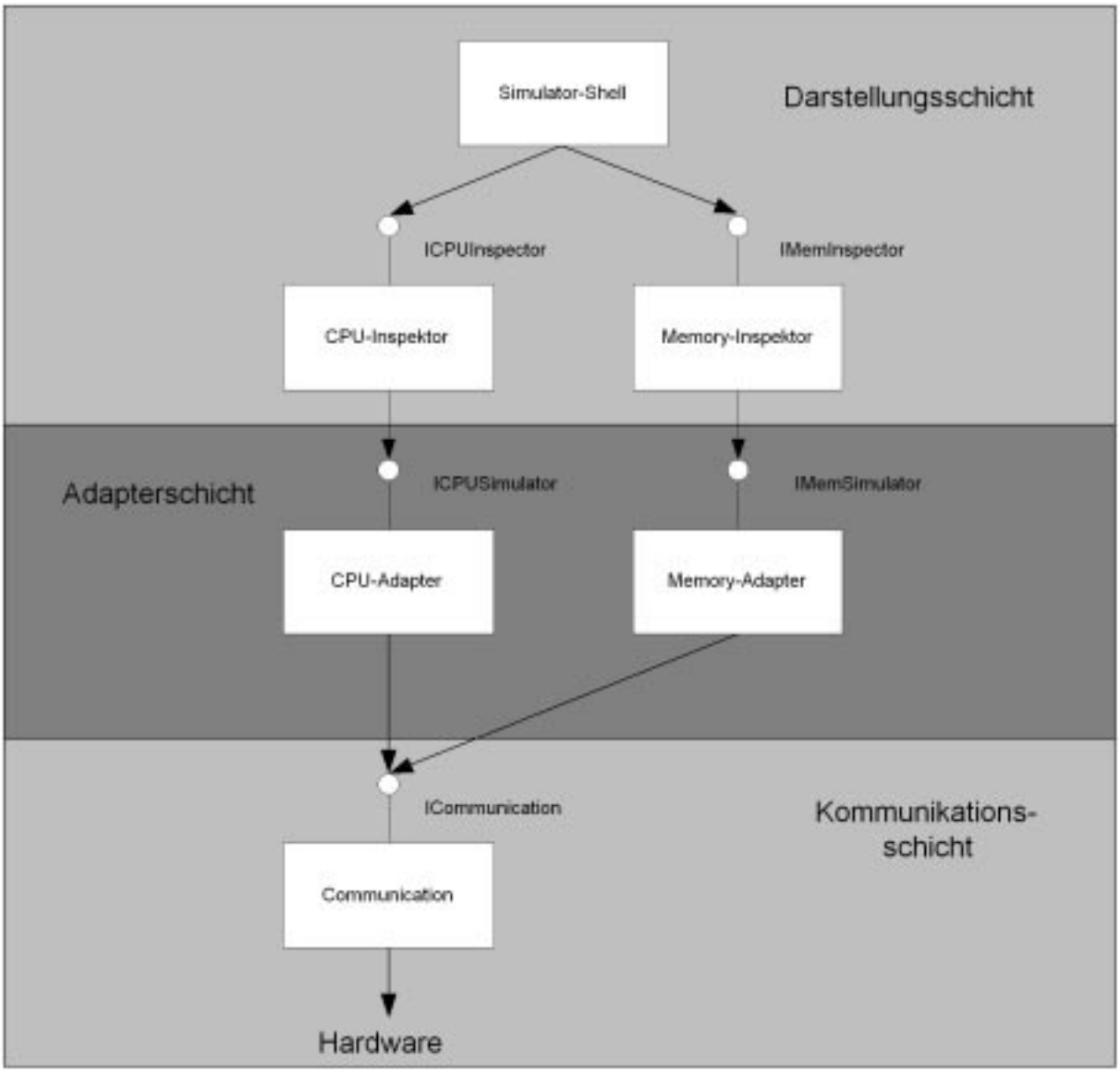

Abbildung 3: FLEXS im In-Circuit-Betrieb

Die nebenwirkungsfreie Austauschbarkeit von Simulator- und Adapterschicht wird durch die Implementierung derselben Schnittstellen möglich. Die Komponenten der Adapterschicht setzen die Anfragen der Darstellungsschicht im wesentlichen auf Anfragen an die Testhardware um, die über die Kommunikationsschicht weitergeletet werden. Die Kommunikationskomponente erfüllt zwei Aufgaben. Einerseits werden die Forderungen der höheren Schichten sequentialisiert und andererseits wird das verwendete Kommunikations-protokoll gekapselt und dadurch für die Adapterkomponenten transparent.

Adaptierung an unterschiedliche Zielhardware

Aufgrund seiner modularen Struktur läßt sich FLEXS mit relativ geringem Aufwand an unterschiedliche Zielhardware anpassen. Der für die betrachtete Zielgruppe häufig auftauchenden Forderung nach Skalierbarkeit der der CPU-Architektur z. B. in Bezug auf die Registeranzahl oder die Größe des adressierbaren Speicherraums wird bereits im Interface zwischen Shell und CPU-Inspektor Rechnung getragen. Dieses erlaubt den Aufruf eines im CPU-Inspektor implementierten optionalen Konfigurationsdialogs. Die so gewonnenen Informationen müssen je nach Art der Änderung eventuell auch in den Komponenten CPU-Simulator und CPU-Adapter berücksichtigt werden.

Die Adaptierung an eine gänzlich neue CPU-Familie ist mit den größten Aufwendungen verbunden, da in diesem Fall mindestens drei Komponenten neu implementiert werden müssen, nämlich CPU-Inspektor, CPU-Simulator und CPU-Adapter. Die Komponenten für die Visualisierung, Simulation und Adaptierung von Speicherbereichen können aufgrund ihrer generischen Implementierung unverändert wiederverwendet werden. Gleiches trifft auf die Kommunikationskomponente zu. Der tatsächlich erforderliche Aufwand hängt sehr stark von der Komplexität der CPU ab. Im CPU-Inspektor wird der Aufwand wesentlich durch die zu implementierende Disassemblerkomponente bestimmt, die für das Single-Stepping und die zugehörigen in der Shell implementierten Ansichten benötigt werden.

Ein zweiter aufwandsbestimmender Faktor ist die Art der Einbindung eines externen Assemblers in den CPU-Inspektor. Einerseits besteht die Möglichkeit zur Umlenkung der Fortschritts- und Fehlermeldungen des Assemblers in ein Fenster der Shell. Dies setzt voraus, daß der Assembler seine Ausgaben über den Standardausgabekanal realisiert. Etwas aufwendiger ist die Auswertung der Symbol- und Debuginformationen zur Synchronisation des Quelltextes mit dem CPU-Zustands in der entsprechenden Ansicht der Shell, der gegebenenfalls aus dem Objektcode rekonstruiert werden muß. Wird der Assembler gemeinsam mit dem CPU-Core etwa auf Basis eines tabellengesteuerten generischen Assemblers oder unter Zuhilfenahme eines Compiler-Compilers entwickelt, bietet sich die Einbindung in Form einer DLL an, die dem Inspektor Zugriff auf die internen Informationen des Assemblers gewährt.

Ähnliches trifft auf den CPU-Simulator zu. Die Dekodierung der Einzelbefehle kann nach denselben Algorithmen erfolgen, die im Inspektor zur Disassemblierung verwendet werden. Zusätzlicher Aufwand entsteht jedoch durch die Mitführung der für die Ausführung benötigten Taktzyklen sowie die Simulation CPU-interner Zustände, die z. B. in Architekturen mit einer Instruktions-Pipeline auftreten. Der Aufwand für die Implementierung eines CPU-Adapters kann demgegenüber fast vernachlässigt werden, da hier im wesentlichen nur eine Weiterleitung der Inspektoranfragen an die Kommunikations-komponente erfolgt.

Für einen typischen 4-Bit-Mikrocontrollerkern kann man von einem Aufwand von etwa 6...12 Mannwochen für die Implementierung der oben genannten Komponenten ausgehen. Dies ist nur ein Bruchteil des Implementierungsaufwands für die Shell. Alle dort implementierten Leistungsmerkmale werden ohne Neuübersetzung des Systems für die neue CPU-Architektur verfügbar.

Projektstatus und Weiterentwicklung

Das FLEXS-System unterstützt zur Zeit zwei unterschiedliche 4-Bit-Cores [HAR98] [PLO97]. Das In-Circuit-Debugging wurde für den Core RUN4 implementiert. Dieser Core wurde im Auftrag eines mittelständischen Unternehmens an der Rostocker Universität entwickelt. Als Debug-Interface wird JTAG verwendet, wobei ein ebenfalls an der Rostocker Universität entwickelter JTAG-EPP-Umsetzer für die Kommunikation mit dem unter Windows-NT arbeitenden Entwicklungssystem verwendet wird [HIL97]. Der Entwurf der Kommunikationsarchitektur wurde alternativ auf Basis einer seriellen RS232-Verbindung verifiziert, wobei als zu testendes System ein Softwaresimulator verwendet wurde, der auf Basis der vorhandenen RUN4-Simulationskomponenten implementiert wurde.

Der Schwerpunkt der Weiterentwicklung des FLEXS-System liegt auf der Ergänzung um Komponenten zur takgenauen I/O-Simulation. I/O-Signale werden dabei aus Sicht der CPU als spezielle Speicherräume betrachtet und folgerichtig über das IMemory-Interface angekoppelt. Bei der Implementierung der I/O-Simulatoren werden zwei unterschiedlich komplexe Ansätze verfolgt. Einerseits sollen externe Signale mit Hilfe eines speziellen Editors als Szenarien abgespeichert werden können, die bei einem Simulationslauf als takabhängige Datenwerte im I/O-Raum verwendet werden. Dadurch kann z. B. die Antwortzeit des Systems in worst-case-Situationen ermittelt werden. Der zweite Ansatz geht darüber hinaus, indem anstelle fester I/O-Szenarien von einer frei programmierbaren Prozeßsimulation dynamisch berechnete I/O-Daten verwendet werden sollen.

Weitere Arbeiten betreffen die Anreicherung der in der Shell implementierten Funktionalität. Dies sind insbesondere die Einführung eines Beobachtungsfensters für Variablen und Speicherzellen sowie die Implementierung komplexer Break- and Tracepoint-Konditionen.

Referenzen:

[HAR98] Harnack, Matthias: „4-Bit-RISC Mikrocontroller“, Diplomarbeit an der Technischen Universität Braunschweig, 1998

[HIL97] Hildebrandt, Jens: „Aufbau eines Parallelport-JTAG-Interface für IBM-kompatible Personalcomputer“, Diplomarbeit am Fachbereich ET/IT, Institut MD der Universität Rostock, 1997

[KAT98] Katschke, Bert: „Entwicklung eines Softwaresimulators für einen 4 Bit RISC Prozessor auf Basis von COM/OLE", Studienarbeit am FB ET/IT, Institut MD der Universität Rostock, 1998

[MICH98] Michelsen, Robert: „Entwurf und Implementierung eines Quelltexteditors mit unterstützter Syntaxcolorierung auf Basis der COM/OLE-Technologie“, Kleiner Beleg am Fachbereich ET/IT, Institut MD der Universität Rostock, 1998

[MICH99] Michelsen, Robert: „Entwicklung einer Simulations- und Entwicklungs-umgebung für Mikroprozessoren auf Basis von COM/OLE“, Studienarbeit am Fachbereich ET/IT, Institut MD der Universität Rostock, 1998

[PAP96] Papenfuß, Frank: „Evaluierung minimaler Prozessorarchitekturen“, Diplomarbeit am Fachbereich ET/IT, Institut MD der Universität Rostock, 1996

[PLO97] Ploog, Hagen: „RUN4-Mikrocontroller“, Interner Bericht, Fachbereich ET/IT, Institut MD der Universität Rostock, 1997