HW/SW-Codesign ressourcenminimaler Systeme, Teil 1
Eine Simulations- und Online Debugumgebung

Tino Rachui, Hagen Ploog, Jens Hildebrandt

Universität Rostock
Institut für Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Str. 31, 18119 Rostock, Tel.: (0381) 498 35 36
email: {tr, hp, hilde}@e-technik.uni-rostock.de

Abstract: Im vorliegenden Beitrag wird eine Simulations- und Testumgebung für die Softwareentwicklung speziell kleiner 4 bis 8 Bit Mikroprozessoren beschrieben, die in Bezug auf die Leistungsmerkmale dem Standard moderner Entwicklungstools entspricht. Ein leistungsstarkes Merkmal der vorgestellten Entwicklungsumgebung ist die Fähigkeit, sowohl im Simulations- als auch im Online Debugmodus zu laufen. Die Architektur und die verwendeten Technologien zur Realisierung dieser Entwicklungsumgebung ermöglichen außerdem eine schnelle Anpassung an verschiedene Mikroprozessorarchitekturen.

1 Einleitung

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.
Um von der Flexibilität solcher Mikroprozessoren profitieren zu können benötigt der Anwender entsprechende Entwicklungs- und Testumgebungen. Die zur Verfügung gestellte Toolkette gliedert sich dabei häufig in mehrere Einzeltools auf. Üblicherweise unterscheidet man zwischen:

- Integrierten Softwareentwicklungsumgebungen (IDE)
- Simulationsumgebungen für das Zielsystem
- Incircuit-Emulatoren zum Test der realen Hardware

Die in diesem Beitrag vorgestellte Entwicklungsumgebung vereint die Leistungsmerkmale der oben beschriebenen Tools. Die Architektur und die Designmerkmale dieses Entwicklungssystems, im weiteren Verlauf als FLEXS (Flexibles Mikroprozessorentwicklungs- und Simulationssystem) bezeichnet, wurden bereits in [RAC98] beschrieben. FLEXS wurde unter Windows NT auf Basis der COM/OLE Technologie entwickelt.
Dieser Beitrag konzentriert sich schwerpunktmäßig auf das Online-Debug-Interface der Entwicklungsumgebung.

2 Architektur der Entwicklungsumgebung

Ein wesentliches Designkriterium bei der Entwicklung von FLEXS war die möglichst maximale Wiederverwendbarkeit der einzelnen Teilkomponente. 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. 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 Objektimplementierungen. Gerade unter diesem Aspekt bietet sich die Realisierung der Objekte durch Software-Komponenten an, da so der Austausch der Implementierung ohne Neuübersetzung oder Neustart des gesamten Systems möglich wird. Für den Benutzer

besteht vielmehr die Möglichkeit, beim Anlegen eines neuen Entwicklungsprojektes zwischen Simulations- und In-Circuit-Betrieb auszuwählen (Bild 1).

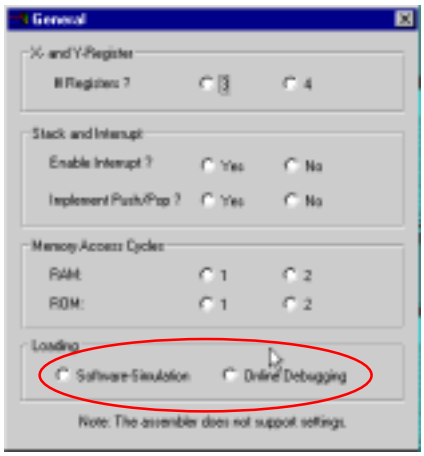

Bild 1: Konfigurationsdialog zur Auswahl zwischen Simulator- und In-Circuit-Betrieb

3 Komponenten für den Simulatorbetrieb

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

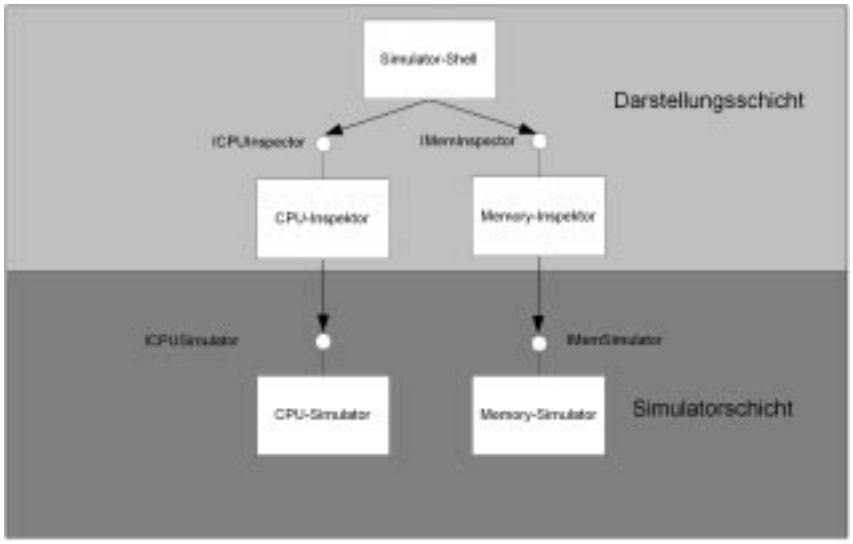

Bild 2: FLEXS-Simulatorbetrieb

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.

4 Komponenten für den In-Circuit-Betrieb

Der eigentliche Vorteil der oben beschriebenen Schichtenarchitektur tritt zutage, wenn statt der Simulation mit der realen Hardware kommuniziert werden soll. In diesem Fall wird die Simulatorschicht vollständig durch eine Adapterschicht ersetzt, wie Bild 3 zeigt. Die nebenwirkungsfreie Austauschbarkeit der Simulator-Komponenten durch die Adapter-Komponenten wird durch die Implementierung derselben Schnittstellen möglich (ICPUSimulator und IMemSimulator). Weder die Simulator-Shell noch die Inspektor-Komponenten bedürfen einer Neuübersetzung, vielmehr ist es möglich, diesen Austausch zur Laufzeit von FLEXS zu vollziehen.
Die Komponenten der Adapterschicht übersetzen die Anfragen der Darstellungsschicht in Anfragen an die Testhardware. Um diese Anfragen an die Zielhardware weiterzuleiten, wurde der Adapter-Schicht noch eine Kommunikationsschicht hinzugefügt. Die in dieser Schicht enthaltene Kommunikationskomponente erfüllt grundsätzlich zwei Aufgaben. Einerseits werden die Forderungen der höheren Schichten sequentialisiert und andererseits wird das verwendete Kommunikationsprotokoll gekapselt und dadurch für die Adapter-Komponenten transparent.

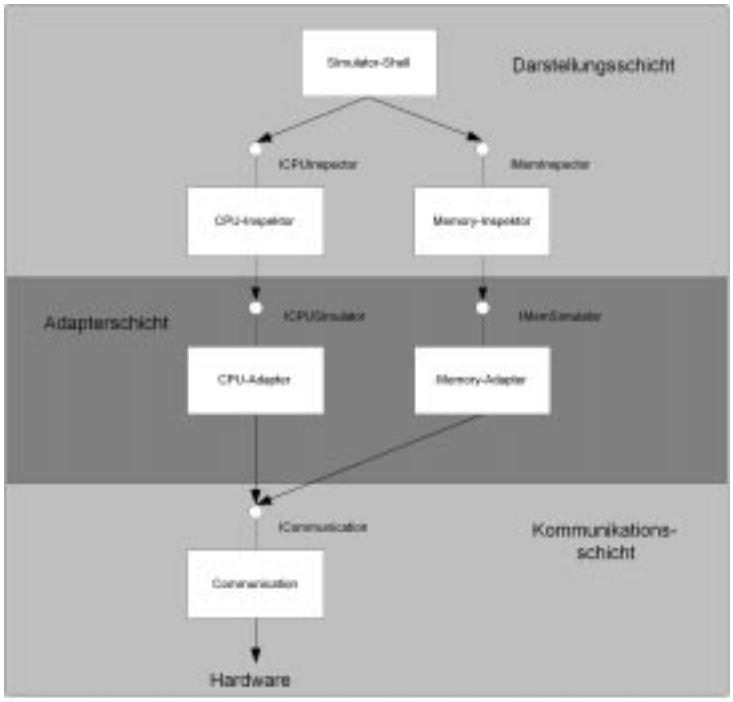

Bild 3 FLEXS im In-Circuit-Betrieb

Die Kommunikationskomponente selbst wurde ebenfalls noch in unterschiedliche Schichten gegliedert. Eine weitere Unterteilung hier schien notwendig, da hier im wesentlichen von drei Freiheiten ausgegangen werden kann:

1. Der Freiheit des verwendeten Mikrocontrollers.
2. Der Freiheit des verwendeten Debug-Interface.
3. Der Freiheit der verwendeten PC-Schnittstelle.

Da in vielen Fällen das verwendete Debug-Interface im direkten Zusammenhang mit dem verwendeten Mikrocontroller steht, ist eine Zusammenfassung von Punkt eins und zwei sinnvoll. Durch das verwendete Schichtenmodell ist eine Trennung jedoch jederzeit realisierbar. Die Kommunikationsschicht wurde schließlich in zwei Unterschichten geteilt. Die obere Schicht (Serializer-Schicht) ist abhängig vom verwendeten Mikrocontroller und dem Debug-Interface. Die untere Schicht (User-Mode-DLL) hingegen ist von der verwendeten PC-Schnittstelle abhängig. Bild 4 verdeutlicht die Modularisierung der Kommunikationsschicht.
Ändert sich einer der oben genannten Freiheitsgrade, dann müssen diese Änderungen nur in der betreffenden Schicht widergespiegelt werden.

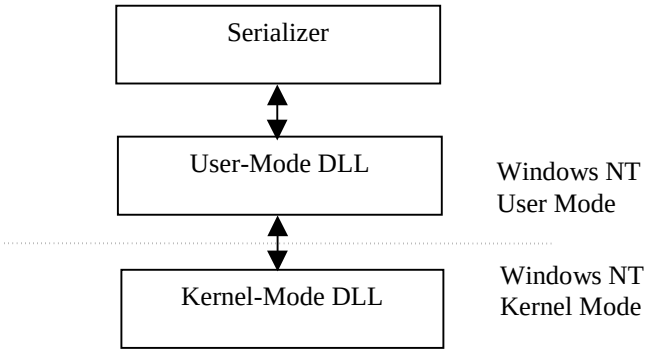

Bild 4: Modularisierung der Kommunikationsschicht

5 Zusammenfassung

Das FLEXS-System unterstützt zur Zeit zwei unterschiedliche 4-Bit-Cores [HAR98] [PLO97]. Das In-Circuit-Debugging wurde für den Core RUN4 implementiert. 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 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.

6 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

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

[RAC98] Rachui, Tino; Woitzel Egmont; Ploog Hagen: „Ein Softwaresimulator auf Basis von COM/OLE für kleine eingebettete Systeme“, Embedded Intelligence Nürnberg, 1998