Ein miniaturisierter Java-Prozessor für mobile Systeme mit eingeschränkten Ressourcen

Frank Golatowski, Nico Bannow, Dirk Timmermann*

Universität Rostock
Fachbereich Elektrotechnik u. Informationstechnik
18119 Rostock, Richard-Wagner-Str.31
Email: gol@e-technik.uni-rostock.de

*Infineon Technologies, CMD DSP
Postfach 80 09 49
81609 München
Email: Nico.Bannow@infineon.com

Abstract. In diesem Beitrag wird der am Institut für Angewandte Mikroelektronik und Datentechnik entwickelten 32-Bit-Java-Prozessor JSM beschrieben. Bei diesem Prozessor handelt es sich um die derzeit kleinste uns bekannte Implementierung eines Javasystems für den mobilen Bereich. Der Prozessor wurde für den Einsatz als SmartCard-Prozessor (JavaCard) bzw. in eingebetteten Systemen entwickelt. Die realisierte Funktionalität entspricht der SUN-JavaCard-Spezifikation 2.1.1. Die in diesem Standard definierten Java-Bytecode-Befehle sind vollständig implementiert. Der Prozessor-Core ist vollständig synthetisierbar. Er wurde in einem FPGA der Virtex1000E-Reihe implementiert und seine Funktionsweise mit dem Rapid-Prototyping-System Aptix MP3 nachgewiesen.

1. Einführung

Der Bedarf an Prozessoren, die direkt JAVA-Code ausführen, wird in den nächsten Jahren stark anwachsen. Die Mobilfunkanbieter beginnen, Java-basierte Mobildienste anzubieten. Diese erfordern, dass die eingesetzten mobilen Systeme Java-Befehle ausführen. Es gibt zwei Möglichkeiten Java direkt auf einem solchen eingebetteten System auszuführen: Die spezielle Softwareimplementierung einer virtuellen Javamaschine (JVM) [Gol99, Kra99] oder eine Implementierung des Java-Standards direkt in Hardware. In diesem Beitrag wird eine Hardwareimplementierung vorgestellt.
Mit der hier vorgestellten Implementierung haben wir das Ziel verfolgt, ein sicheres aber erweiterbares Smartcard-System zu entwickeln. Dieses System genügt dem JavaCard-Standard und ist somit als JavaCard einsetzbar. Darüber hinaus ist das System auch für den Einsatz als eingebettetes System geeignet.
Ein solches Chipkartensystem stellt einfach zu nutzende mobile Rechenkapazität bereit. Es ergeben sich eine Fülle von Anwendungsgebieten, z.B. in Mobiltelefonen, in mobilen Informationssystemen und als Zahlungssystem.

1.1. Stand der Technik

Von SUN werden verschiedene Java-Prozessorklassen hergestellt. Pico-Java ist z.B. ein konfigurierbarer Java-CPU-Kern. Die Prozessoren sind optimiert für die Ausführung von Java-Code, unterstützen aber auch die Ausführung von C und C++ Code. Das Ziel dieser Unterstützung besteht in der Migration dieser Programmiersprachen [Sun98, OT97, HO99].
Die Firma ARM bietet mit dem Prozessor ARM926EJ-S einen vollständig synthetisierbaren 32-Bit-Core an [ARM01]. Die Architektur dieses Prozessors verfügt neben den ARM-Befehlssätzen (32-Bit-ARM-Befehlssatz, 16-Bit-Thumb®-Befehlssatz) über einen dritten Befehlssatz- dem Java-Bytecode. Bei der Ausführung von Bytecode wechselt der Prozessor vom ARM-Modus in den Java-Modus. Ein Teil der definierten Java-Bytecodes wird in Hardware realisiert (140 Befehle). Die restlichen 94 Befehle werden durch eine spezielle Software-Emulation ausgeführt. Undefinierte Bytecodes führen dazu, dass der Prozessor den Java-Modus verlässt und

Exception-Handler ausgeführt werden, die ARM-Code enthalten. Durch diese Architektur soll ein Optimum bezüglich der Kosten des Prozessors, die u.a. durch die Anzahl der Gates bestimmt werden, und der zur Verfügung gestellten Leistung hergestellt werden.
Neben diesen echten Java-Prozessoren gibt es Prozessoren, die durch ihre Architektur geeignet sind, virtuelle Javamaschinen (JVM) effizient zu implementieren und dadurch das Ausführen von Java-Code beschleunigen. Dazu gehört der PSC1000 von Patriot Pacific [Pat98]. Der PSC1000 ist ein stackbasierter 32-Bit-RISC-Prozessor. Durch die stackbasierte Architektur ist der Prozessor geeignet, um JVM und Just-In-Time-Compiler zu implementieren. Er verzichtet auf leistungssteigernde Architekturmerkmale, wie z.B. Pipeling, und enthält keine Programm- oder Daten-Caches. Weiterhin wird ein operandenloser Befehlssatz verwendet und auf breite Befehlsstrukturen mit Mehrfachoperanden verzichtet. Ein Befehl ist 8 Bit breit. Operanden werden im Operandenstack erwartet. Der Prozessor enthält einen separaten I/O-Prozessor für die Entkopplung zeitkritischer Vorgänge von reinen Rechenaufgaben.
MIPS Technologies hat mit MIPS32™4Ksc einen synthetisierbaren Embedded-32-Bit-Core vorgestellt [MIPS01], der für den Einsatz in Smartcard-Anwendungen mit niedriger Verlustleistung vorgesehen ist. Die SmartMIPS-Architektur basiert auf der RISC-Architektur MIPS32. Sie definiert einen leistungsstarken Befehlssatz zur Beschleunigung von Software-Kryptographie-Algorithmen und bietet spezielle Befehle, die die Implementierung virtueller Maschinen verbessert.

1.2. Java

Java ist eine objektorientierte Programmiersprache. Im Gegensatz zu anderen Programmiersprachen liegt das vom Compiler übersetzte Programm nicht in einer speziellen Maschinensprache vor, sondern als Classfile im sogenannten Java-Bytecode. Dieser Java-Bytecode wird von einem Bytecode-Interpreter ausgeführt. Ein Bytecode-Interpreter ist in allen aktuellen Internetbrowsern integriert.
Diese virtuellen Maschine auf denen die Java-Classfiles laufen, können als Software- oder Hardwarelösung ausgeführt sein. Derzeitige JVM (Java Virtual Machines) sind meistens als Softwarelösung realisiert.
Das liegt vor allem an der weiten Verbreitung der Desktop- und Server-Systeme, auf denen diese Softwarelösungen laufen. Spezielle Hardwareimplementierungen für den PC-Markt haben sich als nicht sinnvoll erwiesen.
Anders ist die Lage auf dem Gebiet von eingebetteten Systemen, wie z.B. von Smartcards. Diese Karten werden ständig neu ausgegeben. Durch den geringen Stückpreis von Smartcards werden diese nach Ablauf einer bestimmten Zeit – aufgrund nicht mehr erfüllbarer Ansprüche – nicht etwa neu programmiert, sondern nach nur wenigen Jahren oder Monaten Betriebszeit weggeworfen.
Die meisten derzeitig existierenden Karten, die den Java Standard implementieren, laufen mittels einer Softwareemulation. Eine direkte Unterstützung von Java-Bytecode mittels eines Java Prozessors hat mehrere Vorteile. Darunter fallen die höhere Ausführungsgeschwindigkeit und die Einsparung von Chipfläche durch das Wegfallen der gesamten Emulationssoftware. Ein weiterer Vorteil ist die Möglichkeit einer konsequenten Umsetzung von Sicherheitsrichtlinien bereits im Prozessor. Im Gegensatz zu einem normalen Prozessor kann ein Test auf viele verschiedene Sicherheitsaspekte größtenteils parallel zur normalen Programmabarbeitung erfolgen.

1.3. Java auf Smartcards: JavaCard

Der Java Bytecode für Smartcards enthält einen eingeschränkten Befehlssatz. Das ist bedingt durch die begrenzten Ressourcen einer solchen Karte. Die Chipfläche darf 25 mm² nicht übersteigen. Die herkömmliche Technologie bei der Fertigung dieser Chips für Smartcards liegt bei 0.7 µm im Gegensatz zu 0.13 µm bei der neusten Generation von Halbleiterbausteinen. Auf diesem sehr begrenzten Platz müssen der Prozessor selbst, der gesamte Speicher (ROM, RAM, FLASH) und die Kommunikationsschnittstellen untergebracht werden. Eine komplette Implementierung des Java Standards ist unter den genannten Gesichtspunkten nicht oder aber nur schwer mit anderweitigen Einschränkungen realisierbar (z.B. weniger Speicher). Aus diesem Grund wurde von der Firma SUN eigens für Smartcards eine Spezifikation ausgearbeitet [Sun].

2. Aufbau des Java-Prozessors JSM (Java Silicon Machine)

Ein Prozessor, der objektorientierte Eigenschaften durch seine Hardware unterstützt, unterscheidet sich wesentlich von einem Universalprozessor.
Die *Java Silicon Machine* ist durch die äußerst komplexen Befehle als CISC Prozessor einzuordnen. Allerdings setzt er den Bytecode, wie die meisten modernen Prozessoren auch, intern in RISC Code um. Diese Vorgehensweise ist jedoch nicht zu verwechseln mit dem bei JVMs oftmals üblichen Umsetzen von Java Bytecode in native, dem Prozessor angepasste Befehle. Vielmehr kann der hier vorgestellte Prozessor den Bytecode direkt abarbeiten. Der Prozessor wird in mehrere lose gekoppelte Module (FSMs) aufgeteilt, um ein strukturiertes, leicht zu änderndes und einfach zu testendes Modell zu erhalten.

Grundsätzlich lassen sich mit der gewählten Struktur alle in modernen Prozessorarchitekturen vorkommende Möglichkeiten wie z.B. Pipelining, paralleler Einsatz mehrerer gleichartiger Module, Sprungvorhersage usw. realisieren.
Der Java-Prozessor JSM (Abbildung 1) ist durch ein modulares Design charakterisiert. Für den Prozessor galt es die hohen Sicherheitsanforderungen der Klasse von Anwendungen zu berücksichtigen, für die der Prozessor entwickelt wurde (high Security Design). Intern werden die Bytecodebefehle durch Microcodebefehle realisiert, die in zwei kaskadierten Microcodetabellen abgelegt sind. Die erste gibt zu jedem anliegenden Bytecode die Einsprungadresse in den eigentlichen Microcode wieder, der in der zweiten Tabelle gespeichert ist. Bei der Ausführung von Microcodesequenzen können beliebige Sprünge und Unterprogrammaufrufe durchgeführt werden. Es ist möglich auf der Basis dieser Mikrocodebefehle beliebige andere Befehle zu implementieren- auch die Implementierung eigener Bytecodes ist möglich.
Der Prozessor ist um weitere Komponenten erweiterbar. In der vorliegenden Version enthält der Prozessor beispielsweise eine einen Triple-DES- Kryptographieprozessor.

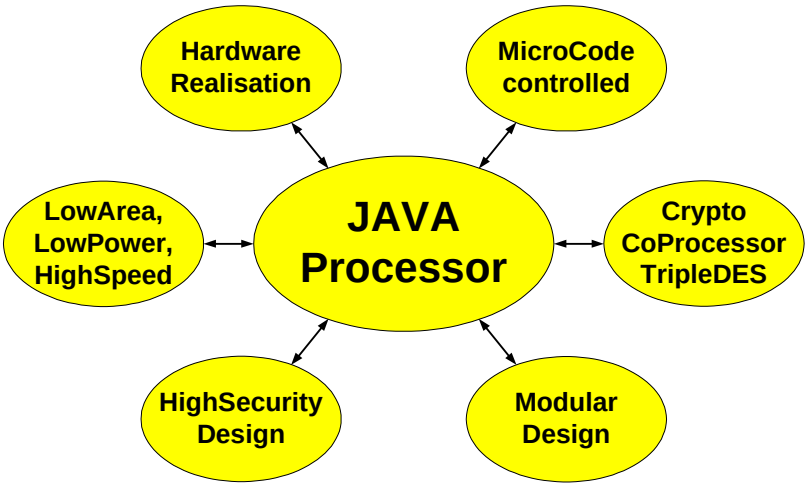

Abbildung 1: Eigenschaften des Java-Prozessors

Der Prozessor besteht aus den Modulen

- Control-Unit (CU),
- Arithmetic Logical Unit (ALU),
- Memory Managment Unit (MMU),
- Stack und
- Input/Output-Unit (I/O-Unit) (s. Abbildung 2)

2.2. Control-Unit

Die Control-Unit hat die Aufgabe, alle untergeordneten Einheiten zu steuern. Die Abarbeitung von Bytecode ist grundsätzlich mikrocodebasiert. Dadurch muss die Programmausführung nicht mittels einer aufwendig zu implementierenden festverdrahteten Schaltung erfolgen. Durch die mikrocodegesteuerte Strategie sind sowohl eine flexible, schnelle Implementierung und Anpassung an Vorgaben als auch ein einfaches Debugverhalten zur Fehlerbehebung möglich. Die Aufgabe der Control-Unit ist es, den Mikrocode ordnungsgemäß zu laden und auszuführen. Die Control-Unit arbeitet die Mikrocodebefehle schrittweise ab. Dazu werden aus einer Tabelle die Befehle für die zu steuernden Module ausgelesen und an diese für die Dauer eines Taktes weitergeleitet. Danach werden solange die Befehle NOP (no operation) angelegt, bis die Module die Befehlsausführung beendet haben. Anschließend beginnt der Zyklus von neuem: ein neuer Mikrocodebefehl wird geladen und ausgeführt.
Für jeden Java-Bytecodebefehl muss mindestens ein Mikrocodebefehl abgearbeitet werden. Dies schließt den Befehl NOP mit ein. Die Anzahl der notwendigen Mikrocodebefehle ergibt sich aus der Komplexität des jeweiligen Javabytecodes.

2.3. ALU

Das Rechenwerk (ALU) dient dem Ausführen von Grundrechenoperationen. Die Operationen umfassen die Addition, Subtraktion, Negation, Multiplikation, Division, bilden des Rests, Schiebeoperationen, logische Operationen, Testoperationen sowie Vergleichsoperationen. Die ALU enthält zwei Register A und B, die die Eingangsoperatoren speichern. Dazu müssen beide Register nacheinander oder in einem Zyklus geladen werden. Erst dann ist eine (sinnvolle) Ausführung der arithmetischen oder logischen Funktion möglich. Die ALU erhält die Operanden als vorzeichenbehaftete 32 Bit-Werte.

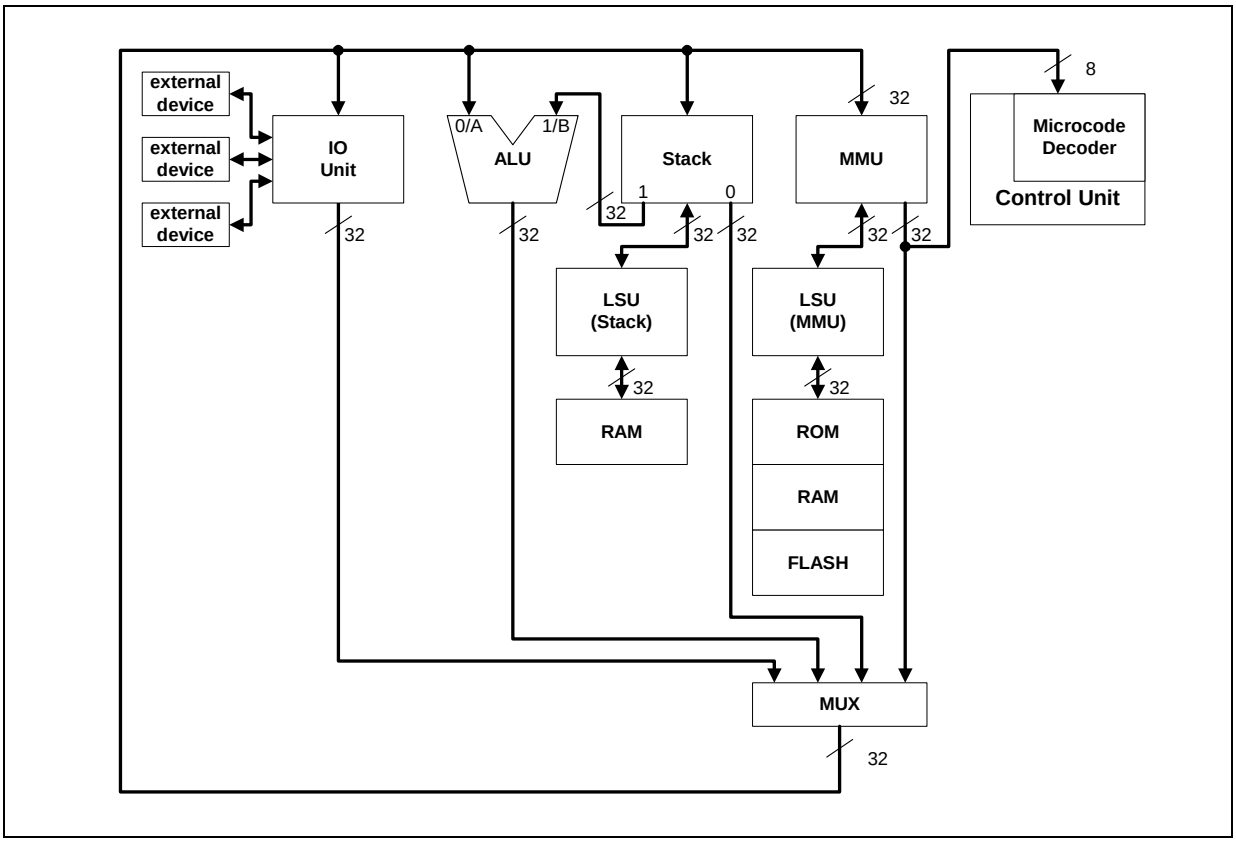

Abbildung 2 Aufbau des JSM-Prozessors

2.4. MMU

Die MMU dient der gesamten Speicherverwaltung der JSM mit Ausnahme der Stackframes, die der Stack eigenständig verwaltet. Die MMU kann verschiedene Speichertypen ansprechen, die je nach Typ der Speicherung von Betriebssystem, Applets, Klassenvariablen und Objekten dienen. Die MMU kann theoretisch, d.h. in Abhängigkeit von der Größe des vorhandenen Speichers, dem Java Standard entsprechend 2^{32} Bit adressieren.

2.5. Stack

Der Stack dient der Zwischenspeicherung von Variablen einer Methode bzw. der Übergabe von Daten an eine Methode. Des weiteren werden bei Methodenaufrufen prozessorinterne Daten abgespeichert, wie der Programmzähler (PC), die aufrufende Methode und die Klasse der aufrufenden Methode.
Die derzeitige Implementierung sieht eine Begrenzung des Stacks auf 2^{10} = 1024 Einträge vor.

2.6. IO- Unit

Die IO- Unit dient der externen Kommunikation des Prozessors über entsprechende Eingabe- und Ausgabegeräte (IO- Geräte), die nicht im Standard von Java definiert bzw. nur über die jeweilig entsprechende API ansprechbar sind. Die IO- Unit stellt die Schnittstelle des Prozessors zu den jeweiligen IO- Geräten dar. Diese sind zwingend notwendig, da ein Ablauf der Programme auf der Karte ohne externe Kommunikation keinen Sinn macht. Auf die externen Geräte darf nur vom Betriebssystem bzw. von dessen API aus zugegriffen werden.
Eine weitere Verwendungsmöglichkeit der IO- Unit ist die Ansteuerung zusätzlicher Hardware, die der Beschleunigung interner Berechnungen dient (kryptographischen Koprozessor).

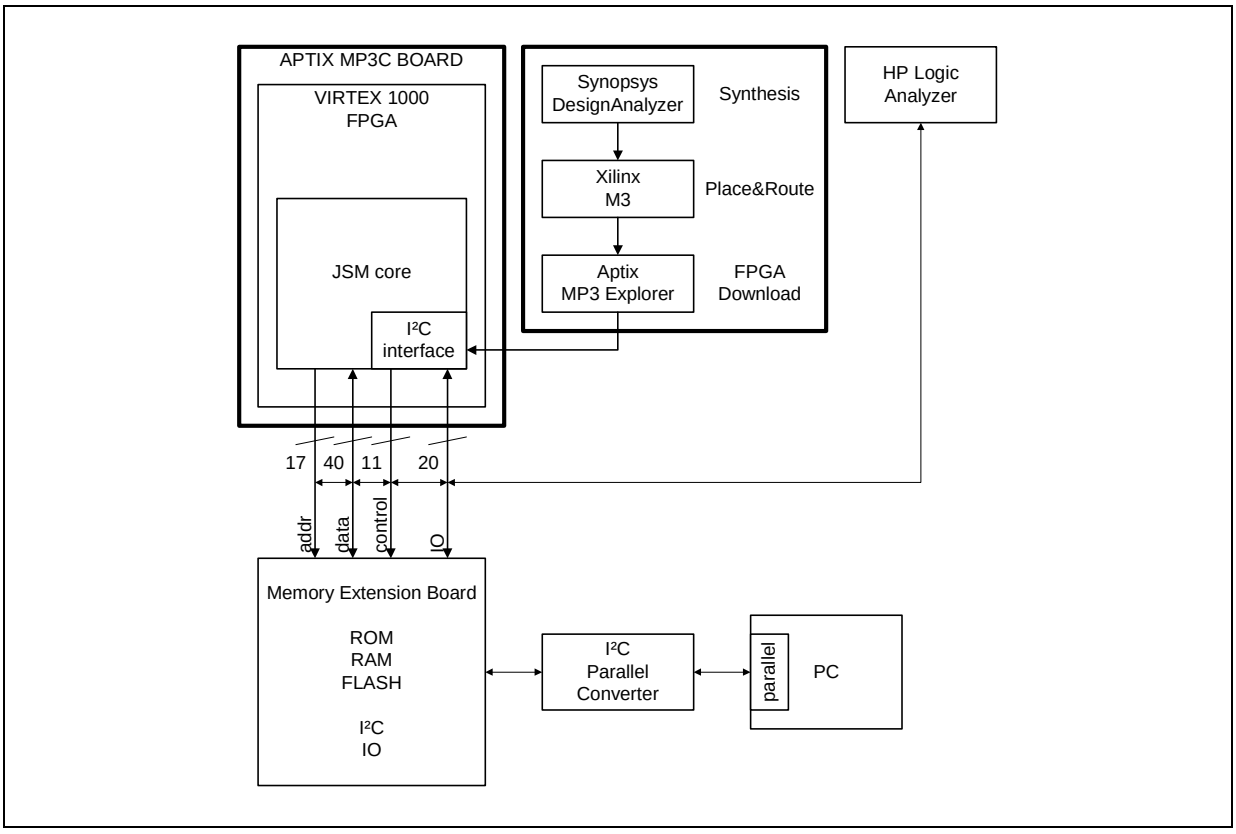

Abbildung 3: Rapid-Prototyping

3. Rapid-Prototyping und Entwicklungsumgebung

Der Funktionsnachweis wurde in der Abbildung 3 dargestellten Rapid-Prototyping-Umgebung erbracht. Ausgehend von der VHDL-Beschreibung wurde das Design unter Synopsys synthetisiert und mit den XILINX-M3-Tools auf das FPGA abgebildet. Das Design wird nachfolgend mittels Aptix-System-Explorer auf das Rapid Prototyping System MP3C in das Virtex1000-FPGA geladen. Damit befindet sich der JavaCore im ausführungsbereiten Zustand. Die Java-Applets werden als Bytecode im externen Speicher (Memory Extension Board) abgelegt. Die Applets können sowohl im ROM als auch im Flash abgelegt werden. Ebenso besteht die Möglichkeit, dass die Java-Applets in den Speicher über ein I^2C-Interface von einem Standard-PC (Terminal) geladen werden. Das Terminal dient weiterhin als Human Interface. In einer Beispielanwendung befindet sich das Applet bereits im ROM. Der erforderliche Bytecode wurde fest in den EPROM abgelegt.

4. Zusammenfassung

In diesem Artikel wurde der Javaprozessor JSM vorgestellt, der den Javabytecode direkt ausführt. Der Prozessor entspricht dem JavaCard-Standard 2.1.1. [Sun00]. Der Funktionalitätsnachweis wurde auf dem Aptix Rapid-Prototyping-System MP3 erbracht. Der Einsatz von Java erlaubt eine neue Qualität im Erstellen komplexer und doch überschaubarer Programme. Problematisch ist u.U. der hohe Speicherbedarf, die durch die objektorientierte Programmiersprache erforderlich ist. Allerdings wird die Chipfläche mit modernen Technologien immer besser ausgelastet, so dass mehr Fläche für Speicher und Logik bereitsteht. Der vorliegende Java-Prozessor-Core benötigt in einem 0.18µ Prozess eine Fläche von weniger als 0.2 mm² zuzüglich der Mikrocodetabellen.

5. Literatur

[ARM01] *JazelleTM – ARM® Architecture Extensions for Java Applications, White Paper, www.arm.com*

[Ban00] *N. Bannow, Java-Prozessors für Smartcards und kleine eingebettete Systeme. Diplomarbeit, Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock, Dez. 2000*

[HO99] *Hangal, S., O'Connor, M.: Performance Analysis and Validation of the picoJava Processor. IEEE Micro, 19 (3), S. 66-72, 1999*

[ICCD99] *H. Ploog, R. Kraudelt, N. Bannow, T. Rachui, F. Golatowski, D. Timmermann: A Two Step Approach in the Development of a Java Silicon Machine (JSM), Workshop on Hardware Support for Objects And Microarchitectures for Java. Austin, Texas, Oktober 1999*

[Gol99] *F. Golatowski, H. Ploog, R. Kraudelt, T. Rachui, O. Hagendorf: Java Virtual Machines für ressourcenkritische eingebettete Systeme und SmartCards, Java Informationstage JIT 99, ITG/GI-Fachtagung, Düsseldorf, September 1999*

[Kra99] *R. Kraudelt, Entwicklung und Implementierung einer JAVA virtuellen Maschine (JVM) für den Einsatz in besonders ressourcenkritischen Systemen (Smartcards). Diplomarbeit, Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock, 1999*

[MIPS01] *MIPS Technologies, MIPS Technologies bringt neuen Prozessor- Core für Smartcard-Anwendungen mit besonders niedriger Leistungsaufnahme auf den Markt, Pressemitteilung, www.mips.com/pressrelease/022001H.html.Feb. 2001*

[OT97] *O'Connor, J., Tremblay, M.: picoJava-I: The Java Virtual Machine in Hardware. IEEE Micro, 17 (2), S.45-53, 1997*

[Pat98] *Patriot Scientific: Java Processor PSC1000. elektronik industrie, H. 2, S. 51.f, 1998*

[Sun98] *Sun Microsystems: picoJava-I Microprocessor Core Architecture. Datenblatt, http://www.sun.com/microelectronics/picoJava/, 1998*

[Sun] *Sun Microsystems, Inc., Java Card Technology Home Page http://java.sun.com/products/javacard.,*

[Sun00] *Sun Microsystems Inc., JavaCardTM 2.1.1 Virtual Machine Specification, 2000*