JSM - Ein Java Prozessor für eingebettete Systeme: Aufbau, Implementierung und Rapid-Protoptyping

Frank Golatowski, Nico Bannow, Jens Hildebrandt, Hagen Ploog, Dirk Timmermann

1 Einleitung

Ein Prozessor, der eine objektorientierte Programmiersprache durch seine Hardware unterstützt, unterscheidet sich wesentlich von einem Universalprozessor. In diesem Beitrag wird der Aufbau, die Implementierung und das Rapid-Prototyping des am Institut MD entwickelten Java-Prozessor JSM beschrieben. Bei diesem Prozessor handelt es sich um die derzeit kleinste uns bekannte Implementierung eines JavaCard-Prozessors. Der Prozessor wurde für den Einsatz als SmartCard-Prozessor bzw. in eingebetteten Systemen entwickelt. Er wurde in einem FPGA der Virtex1000E-Reihe implementiert und seine Funktionsweise mit dem Rapid-Prototyping-System Aptix MP3 nachgewiesen.
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 Handys Java-Befehle ausführen. Es gibt zwei Möglichkeiten Java direkt auf einem eingebetteten System auszuführen: Die spezielle Softwareimplementierung einer virtuellen Javamaschine (JVM) für Mikrocontroller und eine Implementierung des Java-Standards direkt in Hardware.

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

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 vorkommenden Designprinzipien wie Pipelining, paralleler Einsatz mehrerer gleichartiger Module, Sprungvorhersage usw. realisieren.
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 1)

2.1 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

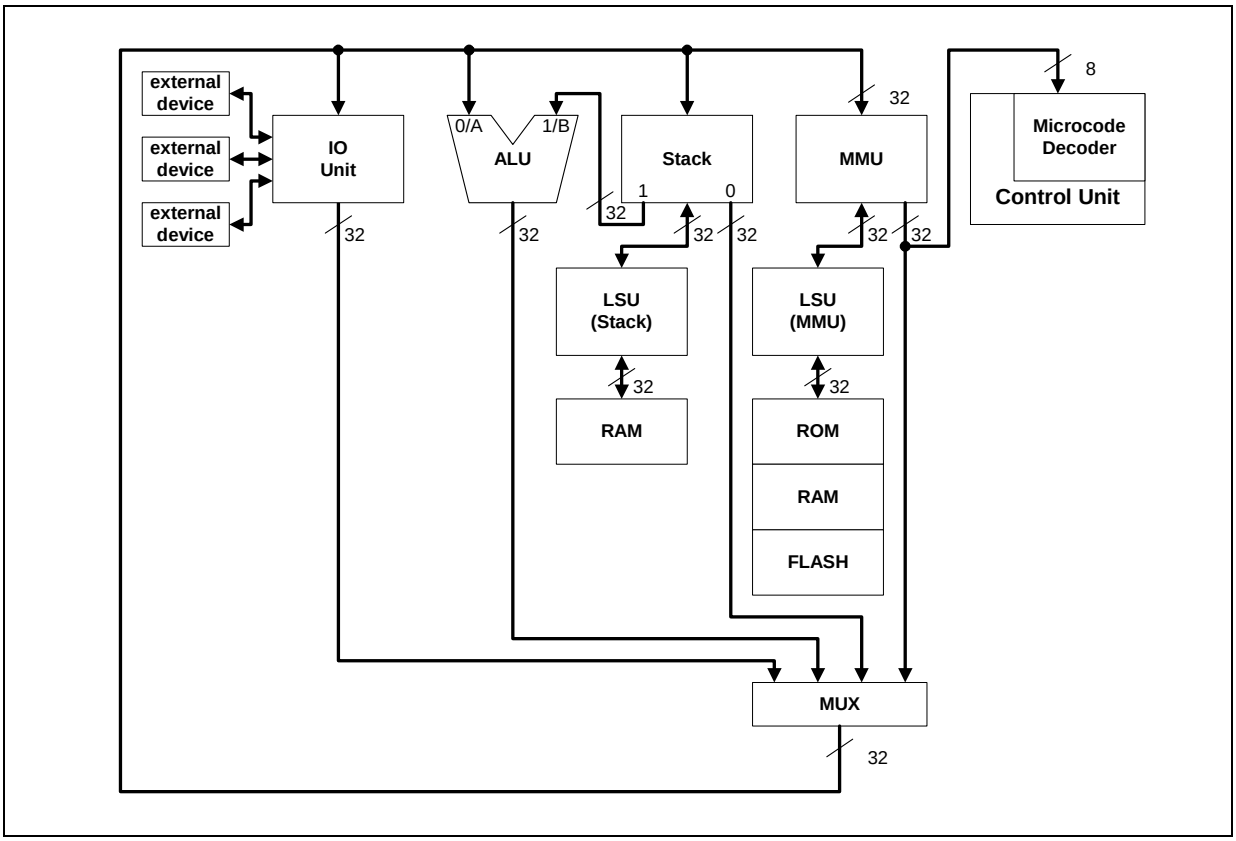

Abbildung 1 Aufbau des JSM-Prozessors

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 so lange 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.

2.2 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.

2.3 MMU

Die Memory Management Unit (MMU) dient der gesamten Speicherverwaltung der JSM. Die MMU kann verschiedene Speichertypen ansprechen, die je nach Typ der Speicherung von Betriebssystem, Applets, Klassenvariablen und Objekten dienen. Die MMU kann in

Abhängigkeit von der Größe des vorhandenen Speichers, dem Java Standard entsprechend 2^{32} Bit adressieren.

2.4 Stack

Der Stack dient der Zwischenspeicherung von Variablen einer Methode bzw. der Übergabe von Daten an eine Methode. Bei Methodenaufrufen werden prozessorinterne Daten abgespeichert wie z.B. der Programmzähler (PC), die aufrufende Methode und die Klasse der aufrufenden Methode. Die Stackframes werden nicht von der MMU, sondern eigenständig vom Stack verwaltet. Die derzeitige Implementierung begrenzt den Stack auf 2^{10} = 1024 Einträge.

2.5 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 der Anschluss zusätzlicher Hardware, die der Beschleunigung von internen Berechnungen dient (z. B. eines kryptographischen Koprozessor)

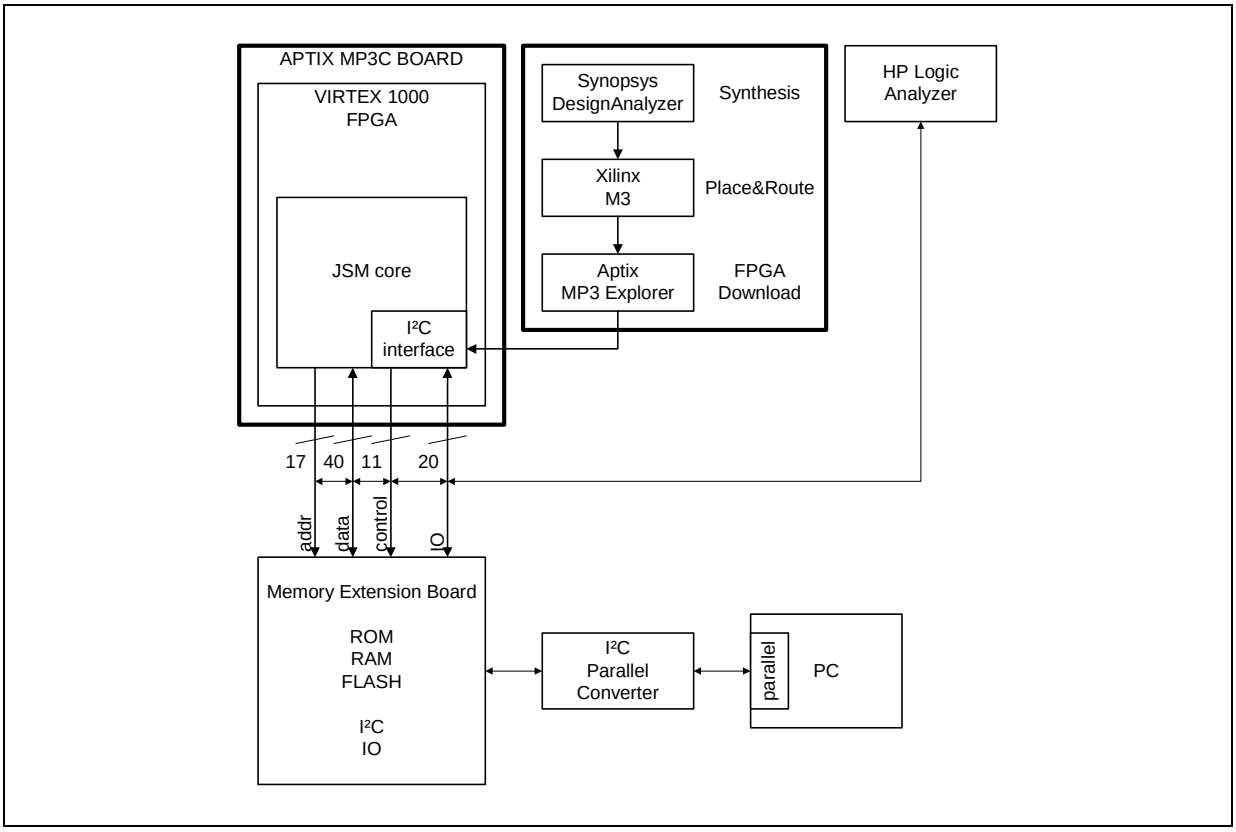

Abbildung 2: Rapid-Prototyping

3 Rapid-Prototyping und Entwicklungsumgebung

Der Funktionsnachweis wurde in der Abbildung 2 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 der 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, der 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

[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 Micro architectures for Java. Austin, Texas, Oktober 1999

[JIT99] F. Golatowski, H. Ploog, R. Kraudelt, T. Rachui, O. Hagendorf: Java Virtual Machines für ressourcenkritische eingebettete Systeme und Smart-Cards, Java Informationstage JIT 99, ITG/GI-Fachtagung, Düsseldorf, September 1999

[Sun00] Sun Microsystems Inc., 2000, JavaCard™ 2.1.1 Virtual Machine Specification

Verfasser:

Dr.-Ing. Frank Golatowski
Universität Rostock
Fachbereich Elektrotechnik u. Informationstechnik
Institut für Angewandte Mikroelektronik und Datentechnik
18119 Rostock, Richard-Wagner-Str.31
EMail: gol@e-technik.uni-rostock.de

Dipl.-Ing. Nico Bannow

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