TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für technische Informatik, Professur für VLSI Entwurfssysteme, Diagnostik und Architektur

Implementierung eines hardwaregestützten Schedulers für den Bytecode-Prozessor SHAP

Vortrag zum Beleg

Marco Kaufmann
marco.kaufmann@inf.tu-dresden.de

Dresden, 08.04.2009

Gliederung

1. Aufgabenstellung und Motivation
2. Einleitung
3. Das Schedulermodul
4. Ergebnisse

Aufgabenstellung

- Auslagerung eines Großteils der Scheduleraufgaben in eine dedizierte Hardwarekomponente.
- Literaturstudium zu gebräuchlichen Schedulingansätzen für JVM-Implementierungen im Embedded-Bereich.
- Berücksichtigung von Monitorverwaltung und blockierender Ein- und Ausgabe.
- Auswahl, Entwurf und Implementierung eines Ansatzes.
- Zusammenfassung und Dokumentation des Entwurfs und der erzielten Ergebnisse.

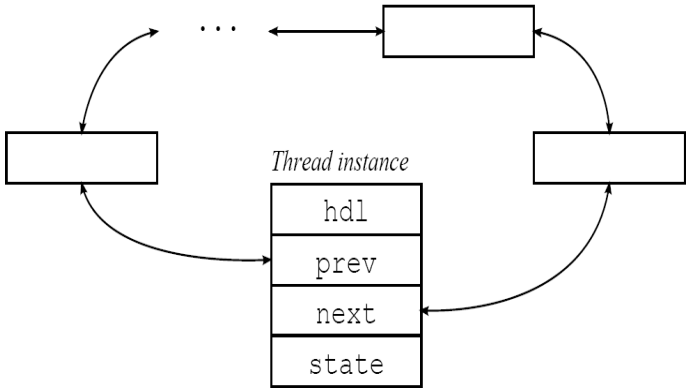
Motivation

- Entlastung der SHAP-CPU, mehr Rechenkapazität für Applikationen.
- Verbesserung bezüglich der Garantie von Rechenkapazitäten und somit der Vorhersagbarkeit von Ausführungszeiten, da Scheduleroverhead nicht eingeplant werden muss.
- Komplexere Threadverwaltung und differenzierteres Threadscheduling erwünscht.

Threadscheduling in eingebetteten und Echtzeitsystemen

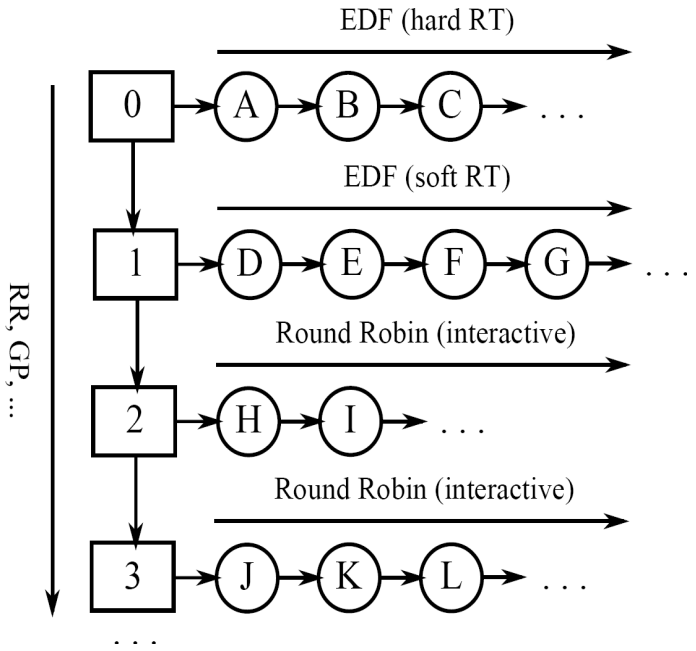
- Echtzeitfähigkeit:
  - Vorhersagbarkeit von Ausführungszeiten wichtiger als optimale Auslastung oder hoher Durchsatz.
- Verbreitete Schedulingverfahren in ES:
  - RMS, EDF, LLF (prioritätengesteuerte Verfahren).
  - Round-Robin, Weighted-Round-Robin.
  - RMS, RR, WRR nur, falls Parameter a priori bekannt.
- Garantie von Rechenkapazitäten in einem Zeitintervall hinreichend für die Vorhersagbarkeit von Ausführungszeiten.
  - Isolation der Threads untereinander bezüglich der Rechenkapazität, die sie erhalten, ermöglicht dynamisches Erzeugen von Threads.

Vorherige Implementierung in SHAP



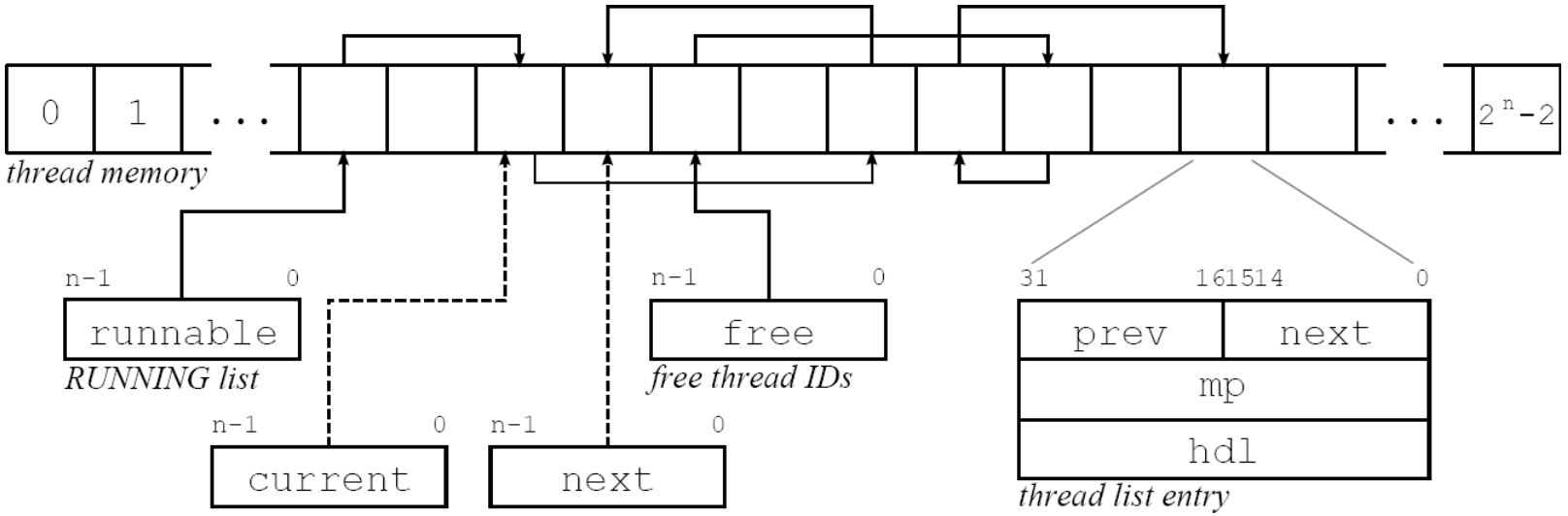
- Datenstrukturen zur Thread-verwaltung im Object-Heap von SHAP.
- *Runnable* als ringförmige Verkettung von Thread-Objekten.
- keine Threadprioritäten.
- Keine Warteschlangen.
- Round Robin Scheduling.

Threadscheduling im Schedulermodul



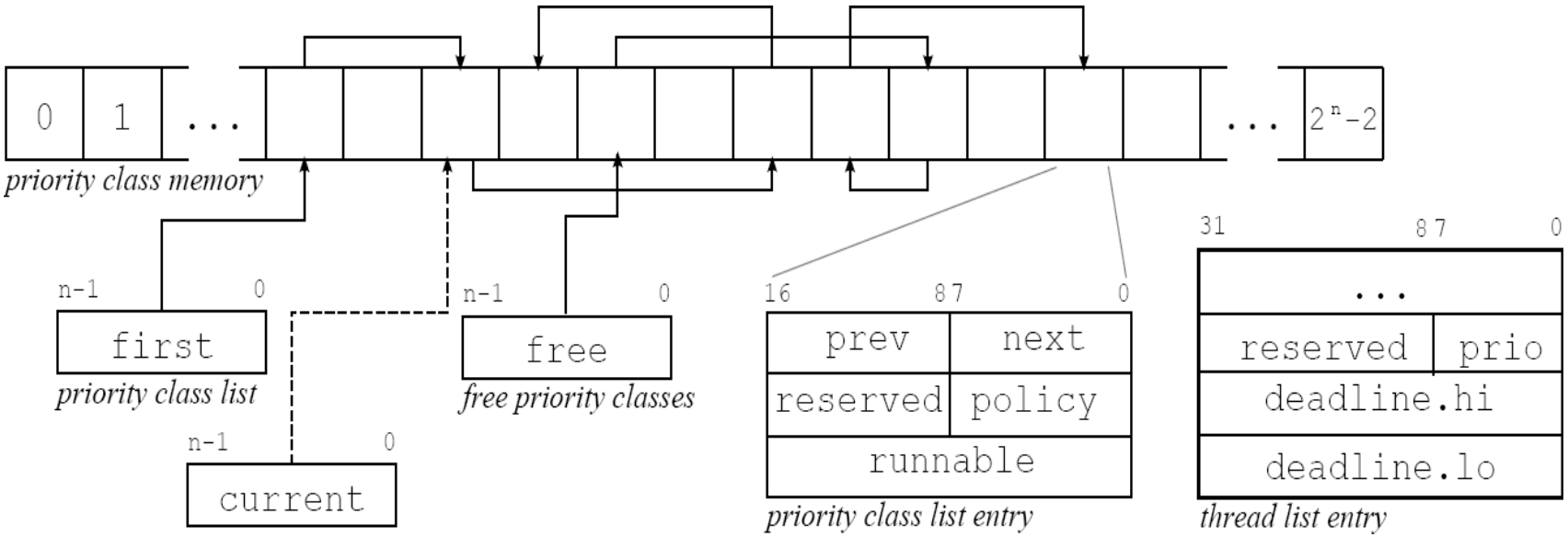
- Zweistufiges Threadscheduling
  - *Runnable* in Prioritätenklassen unterteilt.
  - Anwendung unterschiedlicher Schedulingstrategien auf Thread– und Prioritätenklassenebene zugleich möglich.
- Derzeit unterstützt: RR und FPP.
- Ermöglicht die Isolation von Threads verschiedener Prioritätenklassen gegeneinander.

Threadverwaltung im Schedulermodul (I)

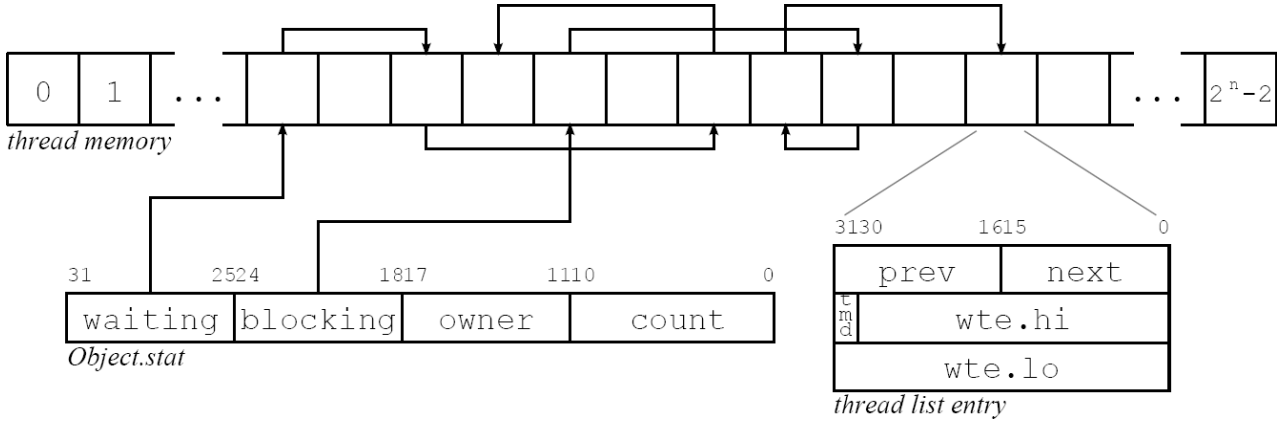


- Datenstrukturen zur Threadverwaltung in eigenem Speicher des Schedulers.
- Thread IDs anstatt Thread Objekte.

Threadverwaltung im Schedulermodul (II)

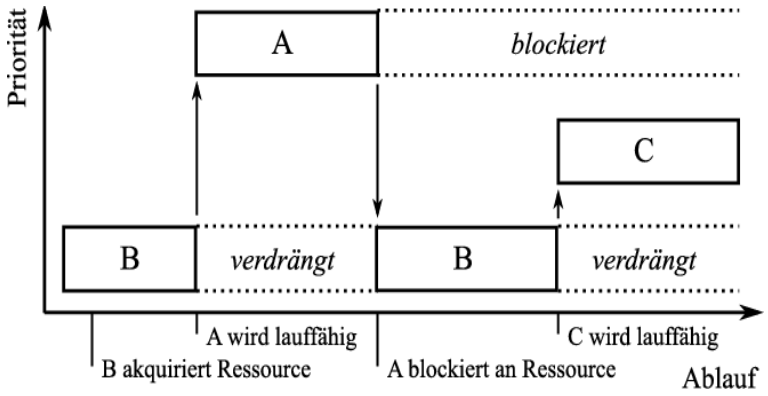


Monitorverwaltung



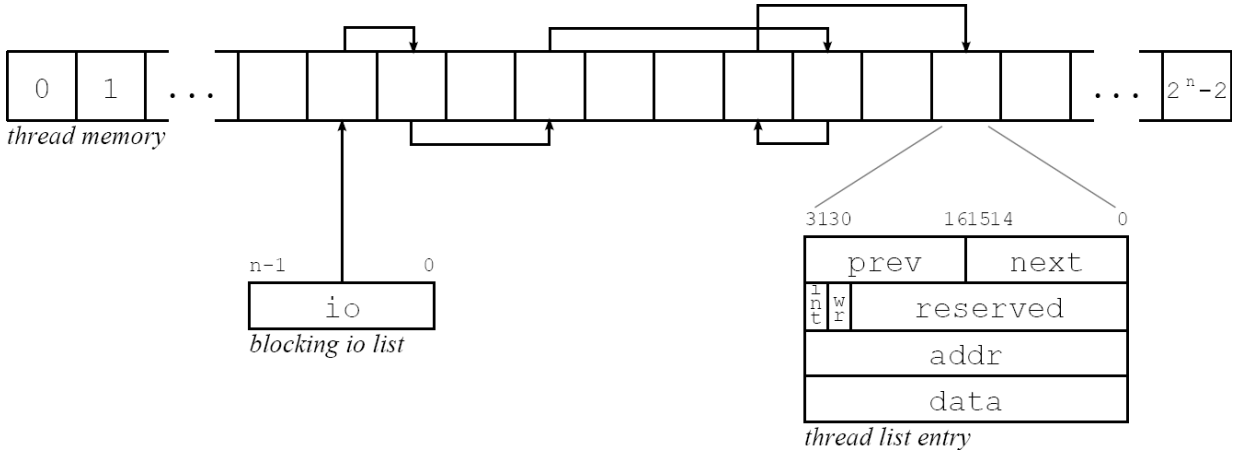
- eigene Warteschlangen für blockierende und wartende Threads für jeden Monitor.
- Zusätzliche Felder für Timed-Waiting im Datensatz eines Threads.

Vermeidung Priority Inversion



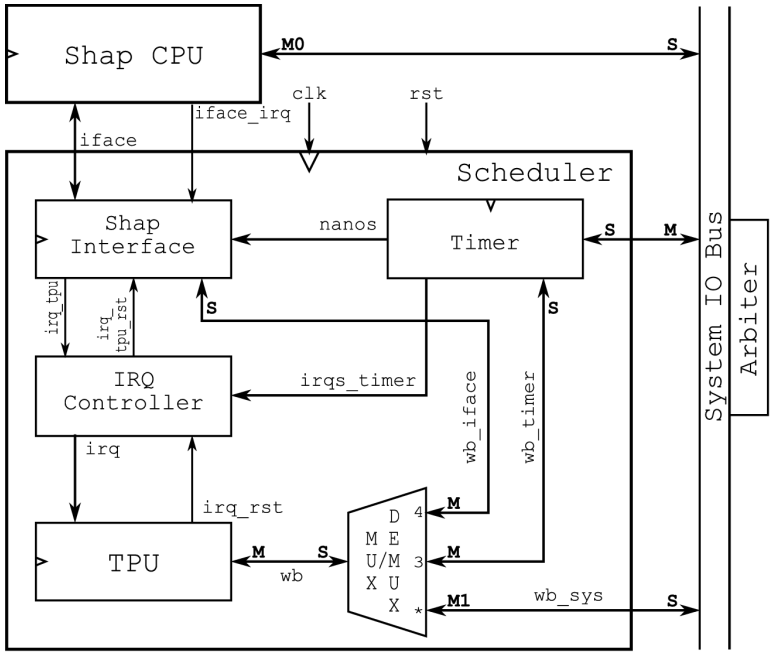
- Gebräuchlich: Priority Inheritance und Priority Ceiling.
- anderer Ansatz: blockierender Thread gibt Rechenzeit direkt an Monitorbesitzer ab.

Blockierende Ein- und Ausgabe



- Polling wird durch Scheduler übernommen.
- Blockierende Threads werden erst aufgeweckt, wenn IO-Operation erfolgreich war.
- Programmierbare Pollingzykluslänge.

Implementierung



- SHAP-CPU: Anpassung der Mikrocodesubroutinen, Einführung neuer Microcodebefehle.
- SHAP-Interface: Schnittstelle zwischen SHAP-CPU und Schedulerkomponente. Beantwortet einfache Anfragen an den Scheduler selbst, z.B. Zeitabfrage, Monitorbeschleunigung.
- TPU: Prozessorkern für Thread- und Monitorverwaltung und zur beantwortung komplexerer Anfragen an den Scheduler.

Syntheseergebnisse

Slices	*FlipFlops*	*LUTs*	*Block-RAMs*	*Multiplier*
3870 (50%)	3337 (21%)	7209 (46%)	12 (50%)	5 (20%)

SHAP mit Schedulermodul auf Spartan-3 FPGA

Komponente	*Slices*	*FlipFlops*	*LUTs*	*Block-RAMs*	*Multiplier*
Scheduler Gesamt	798 (10%)	694 (4%)	1551 (10%)	2 (8%)	2 (8%)
TPU	411 (5%)	285 (1%)	760 (4%)	2 (8%)	2 (8%)
Timer	176 (2%)	219 (1%)	319 (2%)	0 (0%)	0 (0%)
SHAP-Interface	192 (2%)	74 (0%)	331 (2%)	0 (0%)	0 (0%)

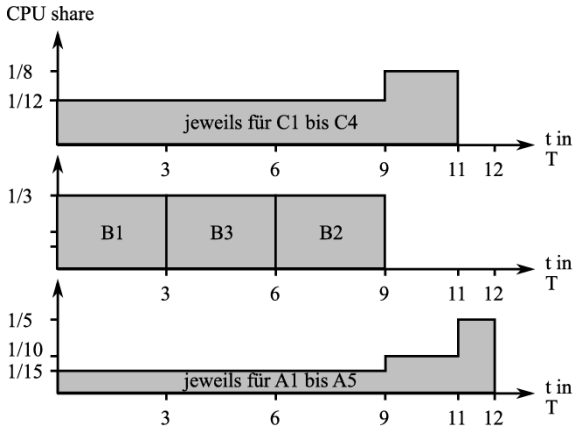
Scheduler und Teilmodule auf Spartan-3 FPGA

Stand der Implementierung / Probleme

- Vollständig implementiert (Hardware, SHAP-Mikrocode, TPU-Software):
  - Timerfunktionen (Milli- und Nanosekundentimer).
  - Thread- und Prioritätenklassenverwaltung.
  - Kontextwechsel-Interrupt, zweistufiges, preemptives Scheduling mit Schedulingverfahren FPP und RR.
- Hardwareseitige Vorraussetzungen für blockierende Ein- und Ausgabe durch Schedulermodul bereits vorhanden, Implementierung in TPU Software und Anpassung des SHAP-Mikrocodes aber noch ausstehend.
- Fehlerhafte Implementierung der Monitorverwaltung, nicht funktionsfähig.

Testapplikation – Threadscheduling

Prioritätenklasse	*Threads*
A (RR)	A1, A2, A3, A4, A5
B (FPP)	B1 (30), B2 (10), B3 (20)
C (RR)	C1, C2, C3, C4



Thread	*t ins ms*	*d in %*
B1	1804	1.11
B3	3647	0.04
B2	5490	0.32
C1	6638	0.76

Thread	*t ins ms*	*d in %*
C2	6638	0.76
C3	6638	0.76
C4	6639	0.75
A2	7296	0.01

Thread	*t ins ms*	*d in %*
A1	7297	0
A3	7297	0
A4	7297	0
A5	7297	0

Testapplikation – Latenz bei Kontextwechseln

	Einzelner Kontextwechsel	*Mehrfache Kontextwechsel*
Latenz in ns	2180	2199
Latenz in Takten	109	≈ 110

- Latenz in vorheriger Implementierung nur 39 Takte.
- Gründe für Erhöhung:
  - Latenz war aufgrund der einfachen Threadverwaltung sehr niedrig.
  - Nun zusätzlich Kommunikation mit dem Scheduler erforderlich.
  - Verhinderung von Race-Conditions zwischen Scheduler und SHAP-CPU.

Auswertung

- Applikationen mehr Rechenkapazität zur Verfügung zu stellen wurde nicht erreicht, da Overhead durch den Scheduler in vorheriger Implementierung bereits sehr gering war.
  - Umschalten zum nächsten Thread: 39 Takte, ein mal pro Millisekunde.
  - Entspricht bei 50 Mhz Systentakt lediglich 0.078% der Rechenkapazität.
- Komplexeres Threadscheduling verfügbar.
- Zweistufiges Scheduling ermöglicht Isolation der Threads verschiedener Prioritätenklassen gegeneinander bezüglich der erhaltenen Rechenkapazität.
  - Dynamisches Erzeugen von Threads in Echtzeitumgebungen möglich.
  - Ziel, die Vorhersagbarkeit von Ausführungszeiten zu verbessern, wurde erreicht.
- Komplexeres Threadscheduling kann aber auch in Software realisiert werden! Geringe Auslastung rechtfertigt den zusätzlichen Hardwareaufwand nicht.
  - Auf Mikrocode – oder Applikationsebene = größtmögliche Flexibilität.
  - Geringerer Hardwareaufwand, nur grundlegende Mechanismen in Hardware.
  - Preis jedoch höhere Latenz und größerer Overhead durch den Scheduler.

Literaturverzeichnis

- Preußer, T. B.; Zabel, M.; Reichel, P.: „*The SHAP Microarchitecture and Java Virtual Machine*“. Forschungsbericht, Technische Universität Dresden, Fakultät Informatik, Institut für Technische Informatik, June 11, 2008.
- Zabel, M.; Preußer, T. B.; Reichel, P.: „*SHAP Reference Manual*“. Technische Universität Dresden, Fakultät Informatik, Institut fär Technische Informatik, June 11, 2008.
- Schoeberl, M.: „*JOP: A Java Optimized Processor for embedded realtime systems*“. Ph.D. dissertation, Vienna University of Technology, 2005.
- Lui, Jane W.S.: „*Real-Time Systems*“. Prentice Hall, 2000. ISBN 0-13-099651-3.
- Tanenbaum, Andrew S.: „*Moderne Betriebssysteme*“. Pearson Studium, 2003, 2. Auflage. ISB 3-8273-7019.
- Ungerer, T; Pfeffer, M.: „*Dynamic Real-Time Reconfiguration on a Multithreaded Java-Microcontroller*“.
- Brinkschulte, U.; Kreuzinger, J.; Ungerer, T.; Pfeffer, M.: „*A Scheduling Technique Providing a Strict Isolation of Real-Time Threads*“.

Literaturverzeichnis

- Kreuzinger, J.; Brinkschulte, U.; Pfeffer, M.; Uhrig, S.; Ungerer, T.: Realtime Eventhandling and Scheduling on a Multithreaded Java Microcontroller
- Kaufmann, M.: „Entwurf eines RISC Prozessorkerns auf dem Sparten-3 Starter-Kit Praktikumsbeleg“. Technische Universität Dresden, Fakultät Informatik, Institut für Technische Informatik, 6. Oktober 2008.
- http://www.rtsj.org: „Real-Time Specification for Java (RTSJ)“, JSR-1.
- http://java.sun.com/javase/6/docs/api/index.html: „JavaT M Platform, Standard Edition 6 API Specification“.
- Hochberger, C.: Java in eingebetteten Systemen. Vorlesung, Technische Universität Dresden, Fakultät Informatik, 2007

Vielen Dank für Ihre Aufmerksamkeit!