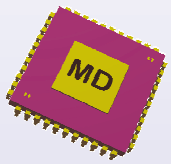
Design issues in the development of a JAVA-Processor for small embedded applications

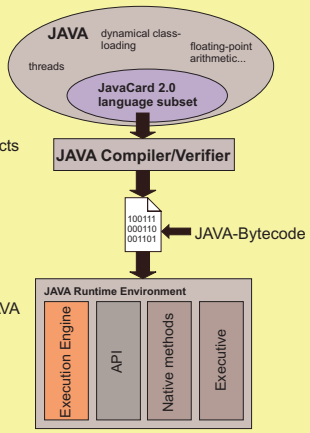
Hagen Ploog, Tino Rachui and Dirk Timmermann
University of Rostock
Institute of Applied Microlectronics and Computer Science
Richard-Wagner-Str. 31
D-18119 Rostock, Germany
email: hp@e-technik.uni-rostock.de
www: http://www-md.e-technik.uni-rostock.de



presented at FPGA'99, Monterey, USA

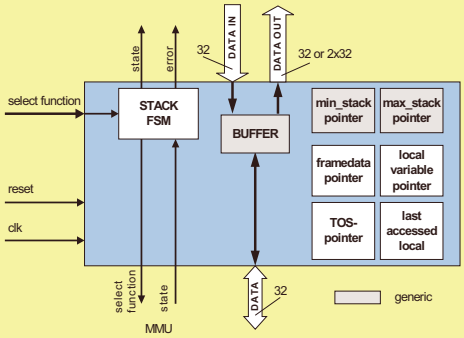
Introduction

- JAVA becomes more and more important by offering features like platform independence
- The JAVA-virtual machine (JVM) acts thereby as a translation-layer between bytecode and the underlying hardware
- This approach leads to resource-expensive implementations
- There are different possibilties to address this problem and make JAVA even more attractive for the small embedded area



STACK

As JAVA is a intensive stack-oriented OOP-language we need a stack with more than PUSH & POP.

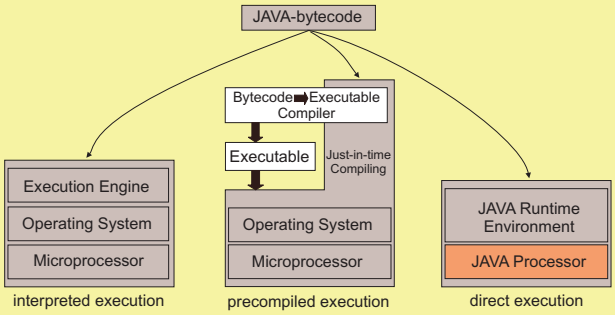


Opcodes which directly access stack data:

nop	aconst_null
lconst_<n>	iload
aload	iload_<n>
aload_<n>	istore
astore	istore_<n>
astore_<n>	pop
pop2	dup
dup_x1	dup_x2
dup2	dup2_x1

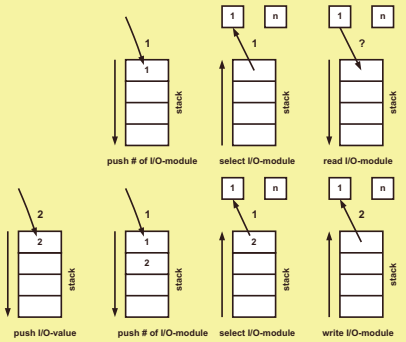
Executing JAVA-bytecode

- There are different approaches for executing JAVA-bytecode
- The fastest approach is to use a JAVA-processor

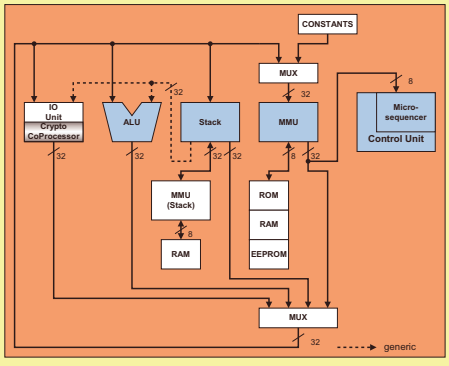


The JAVA -I/O-problem

- JAVA itself can't access HW.
- To address I/O-modules two hidden opcodes are needed:
  - I/O - read
  - I/O - write



Execution Engine



- Loosely coupled state machines (ALU, STACK, MMU, CU) for small footprint architectures

Conclusions

- First benchmark shows an average speed-up of 100 compared with ibutton (Dallas)
- Enables new applications e.g. JAVA-smartcards for mobiles

Future work

- Microcode optimization
- Improved security
- Architecture optimization