BluEJAMM: A Bluespec Embedded Java Architecture with Memory Management

Flavius Gruian\textsuperscript{1} Mark Westmijze\textsuperscript{2}

\textsuperscript{1}Lund University, Sweden  \texttt{flavius.gruian@cs.lth.se}

\textsuperscript{2}University of Twente, The Netherlands  \texttt{m.westmijze@student.utwente.nl}

The 1st International Workshop on Real-Time and Embedded Systems in conjunction with SYNASC’07
Outline

1. Introduction
2. System Architecture
3. BlueJEP, the native Java Processor
4. Memory Management
5. Implementation Results
6. Summary
What are our goals?

1. An embedded Java architecture, as a test platform
2. Evaluate BSV as a design language

BlueSpec System Verilog (BSV)
Rule based, strongly-typed, declarative hardware specification language, making use of Term Rewriting Systems to describe computations as atomic state changes.

3. Outperform other existing Java solutions in terms of
   - design time
   - flexibility
   - execution speed
   - device area

BluEJAMM
BlueSpec Embedded Java Architecture with Memory Management
Constraints and Features

For the BlueJEP Java processor (based on JOP):

- micro-programmed, stack machine core
- predictable rather than high-performance (RT systems)
- given instruction set (bytecodes)
- preset micro-instruction set (for ease of programming)
- given executable image (loaded classes)
- preset back-end (synthesis) tools
- preset implementation platform (FPGA)

Memory management

- given object structure
- both software and hardware (MMU) solutions
Architecture and Configurations Overview

Software memory management
Architecture and Configurations Overview

1. Software memory management
2. Hardware MMU using a dual-port RAM
Architecture and Configurations Overview

1. Software memory management
2. Hardware MMU using a dual-port RAM
3. Hardware MMU using the system bus
Six Stages Pipeline, Stack Machine

Stage 1: BC2 microA
- Fetch Bytecode

Stage 2: micro-ROM
- Fetch micro-I

Stage 3: jump table
- Decode & Fetch Register

Stage 4: Fetch Stack
- Execute

Stage 5: forward bypass
- Write-back

Stage 6: const
- Stack

bus interface (OPB)
Run-Time Environment

BlueJim image generator

- offline class loading and linking
- replaces native calls with custom bytecodes
- throws away unused methods and fields
- adds GC information

**JVM.java**  Java implemented bytecodes.

**Native.java**  Java-hardware interface.

***.java**  Reduced JRE library.
Object structure, address space and garbage collection

- **GC Algorithm: Mark-Compact**
- **Hardware (MMU):**
  - handles all memory management functions
  - tight integration with the processor core (scans the stack for references)
  - stop-the-world GC for now, concurrent later on
Synthesis input, tools, and results

**Input:** BSV code, 1300 lines (BlueJEP) + 600 lines (MMU)

**Tools:**
- BSV compiler 2006.11, \( BSV \rightarrow Verilog \)
- Xilinx EDK 9.1i, \( Verilog + IPs \rightarrow System \)
- Xilinx ISE 9.1i, \( System \rightarrow FPGA \)
- Chipscope, to monitor and debug

**Target:** FPGA, Xilinx Virtex-II (XC2V1000, fg456-4)

**Area:** (no optimization efforts)
- BlueJEP = 3460 slices (68%)
- BlueJEP + MMU = 4340 slices (85%)

**Clock speed:** (few optimization efforts)
- BlueJEP = 85 MHz
- BlueJEP + MMU = 64 MHz
Bytecode and application execution speed

- Execution time in clock cycles for several bytecodes:

```
<table>
<thead>
<tr>
<th>Bytecode(s)</th>
<th>JOP</th>
<th>BlueJEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>iload iadd</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>iinc</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>ldc</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>if_cmpl taken</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>... n/taken</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>getfield</td>
<td>23</td>
<td>38</td>
</tr>
<tr>
<td>getstatic</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>iaload</td>
<td>29</td>
<td>45</td>
</tr>
<tr>
<td>invoke</td>
<td>126</td>
<td>166</td>
</tr>
<tr>
<td>invoke static</td>
<td>100</td>
<td>111</td>
</tr>
</tbody>
</table>
```

Profile for a simple application

<table>
<thead>
<tr>
<th>Profile</th>
<th>SoftGC</th>
<th>MMU</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytecodes</td>
<td>24810</td>
<td>10304</td>
<td>42%</td>
</tr>
<tr>
<td>cc/byte</td>
<td>6</td>
<td>7</td>
<td>117%</td>
</tr>
<tr>
<td>cache fills</td>
<td>1601</td>
<td>675</td>
<td>42%</td>
</tr>
<tr>
<td>mem accesses</td>
<td>9063</td>
<td>3139</td>
<td>34%</td>
</tr>
<tr>
<td>GC clocks</td>
<td>49214</td>
<td>2626</td>
<td>5%</td>
</tr>
<tr>
<td>total cc</td>
<td>168977</td>
<td>73981</td>
<td>44%</td>
</tr>
</tbody>
</table>

- Performance similar to JOP, taking into account the faster clock
- Faster with MMU, even with the reported clock speed degradation
To conclude...

**Summary:** we introduced **BluEJAMM**, which:
- includes a native Java processor
- includes a hardware MMU
- is specified in BlueSpec System Verilog
- proves that BSV is perfect for fast prototyping

**Extensions:**
- Multi-block, multi-method caching [completed]
- Micro-instruction folding [under evaluation]
- Concurrent MMU [under development]
The encoding of the micro-instructions does not affect the assembler (bluejasm)!

The actual encoding is interesting for optimization purposes only.