BlueJEP: A Flexible and High-Performance Java Embedded Processor

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

\textsuperscript{1}\text{Lund University, Sweden}\newline\text{flavius.gruian@cs.lth.se}
\textsuperscript{2}\text{University of Twente, The Netherlands}\newline\text{m.westmijze@student.utwente.nl}

Java Technologies for Real-time and Embedded Systems, 2007
Outline

1. Introduction
2. Design
3. Implementation and Experiments
4. Discussion
5. Summary and Continuing Work
What are we trying to do?

1. Design a Java processor starting from JOP [M. Schöberl]
2. Evaluate BlueSpec System Verilog 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 processors in terms of
   - design time
   - flexibility
   - execution speed
   - device area

BlueJEP

BlueSpec System Verilog Java Embedded Processor
Design features and constraints

Many design features shared between BlueJEP and JOP (VHDL):

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

Complete system overview

- **small**: system on a FPGA
- **flexible**: support exploration
- **real-time**: easily predictable timing
- **portable**: standard interfaces for fast integration (OPB, LMB for Xilinx EDK)

![System Architecture Diagram](attachment://system_architecture.png)
Six Stages Micro-Programmed Pipeline

Stage 1: BC2 microA
- Fetch Bytecode

Stage 2: micro-ROM
- Fetch micro-I
- Decode & Fetch Register

Stage 3: jump table
- Decifo
- fsifo

Stage 4: Fetch Stack
- exifo
- wbifo

Stage 5: Forward bypass

Stage 6: Write-back
- rollback

Bus interface (OPB)
Handling data

- Data dependencies cause stalls (stages 3, 4, 5):
  - searchable FIFOs are used to look for specific destinations
  - stages do not fire if the required sources are destinations in any of the following SFIFOS
- Improved performance through forwarding stack words (from the write-back FIFO – stage 6)
- Register forwarding seems to yield marginal improvements only at the expense of more hardware (therefore not used)
- Bypass Execute (stage 5) for data moving operations
- External memory accessed via registers (MwA, MrA, MD)
Handling control

- Micro-code branches: \texttt{Bz}, \texttt{Bnz}, \texttt{Bp}, \texttt{Bnp}, \texttt{Bm}, \texttt{Bnm}, \texttt{Goto} affect \texttt{pc}

- Java branches are combinations of comparison operations, \texttt{JPC} load/store and micro-branches.

- Speculative execution of micro-branches – always ”not taken”:
  - no need for SFIFOS, no need to stall when \texttt{pc} or \texttt{JPC} changes → simpler hardware
  - context (\texttt{JPC}, \texttt{PC}, \texttt{SP}) must be passed along and restored when needed in the \textit{Writeback} stage → wider FIFOs
  - flushing FIFOs and restoring context is easy → simpler code (hard to debug though...)

- Special register for controlling the load of the method cache (\texttt{CACHECtl}) on invokes and returns.
From assembly to micro-ROM

- The encoding of the micro-instructions does not affect the assembler (bluejasm)!
- The actual encoding is interesting for optimization purposes only.
From application to run-time environment

UserApp.java -> javac -> bjrt.jar

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.
Target System and Tools

**Target**

FPGA
- Xilinx Virtex-II (XC2V1000, fg456-4)

**Tools**
- BSV compiler 2006.11, `BSV → Verilog`
- Xilinx EDK 9.1i, `Verilog + IPs → System`
- Xilinx ISE 9.1i, `System → FPGA`
- Chipscope, to monitor and debug
### Device Area

Synthesis parameters: optimized for speed, distributed RAM.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Taken</th>
<th>Available</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>3460</td>
<td>5120</td>
<td>68%</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>756</td>
<td>10240</td>
<td>7%</td>
</tr>
<tr>
<td>4LUTs</td>
<td>6858</td>
<td>10240</td>
<td>66%</td>
</tr>
<tr>
<td></td>
<td>2422</td>
<td>used as logic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4436</td>
<td>used as RAM</td>
<td></td>
</tr>
</tbody>
</table>

**Observations, compared to JOP**

- Logic takes around the same amount of resources
- RAM takes around five times more resources (the BSV RegFiles are memories with 5 read ports and 1 write port)
Experimental Results

Clock Speed

Maximum clock speeds for BlueJEP and JOP (with OPB):

<table>
<thead>
<tr>
<th></th>
<th>Virtex-II</th>
<th>Spartan3</th>
<th>Virtex5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XC2V-4</td>
<td>XS3-5</td>
<td>XC5VLX30-3</td>
</tr>
<tr>
<td>JOP (OPB)</td>
<td>60 MHz</td>
<td>66 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>BlueJEP</td>
<td>85 MHz</td>
<td>76 MHz</td>
<td>221 MHz</td>
</tr>
<tr>
<td>(\phi)</td>
<td>1.42</td>
<td>1.15</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Clock factor

\[ \phi = \frac{f_{\text{BlueJEP}}}{f_{\text{JOP}}} \]

- BlueJEP running faster than JOP is partly a consequence of increasing the number of stages from 4 to 6!
### Bytecode Execution Speed

<table>
<thead>
<tr>
<th>Bytecode(s)</th>
<th>JOP CC</th>
<th>BlueJEP CC</th>
<th>RS&lt;sub&gt;1.42&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>iload iadd</td>
<td>2</td>
<td>3</td>
<td>0.95</td>
</tr>
<tr>
<td>iinc</td>
<td>11</td>
<td>13</td>
<td>1.20</td>
</tr>
<tr>
<td>ldc</td>
<td>9</td>
<td>12</td>
<td>1.06</td>
</tr>
<tr>
<td>if_icmplt taken</td>
<td>6</td>
<td>23</td>
<td>0.37</td>
</tr>
<tr>
<td>if_icmplt n/taken</td>
<td>6</td>
<td>8</td>
<td>1.06</td>
</tr>
<tr>
<td>getfield</td>
<td>23</td>
<td>38</td>
<td>0.86</td>
</tr>
<tr>
<td>getstatic</td>
<td>15</td>
<td>18</td>
<td>1.18</td>
</tr>
<tr>
<td>iaload</td>
<td>29</td>
<td>45</td>
<td>0.92</td>
</tr>
<tr>
<td>invoke</td>
<td>126</td>
<td>166</td>
<td>1.08</td>
</tr>
<tr>
<td>invoke static</td>
<td>100</td>
<td>111</td>
<td>1.28</td>
</tr>
</tbody>
</table>

- some bytecodes are executed faster, some slower than on JOP
- speculative execution takes its toll on taken branches

**Clock factor**

\[
\phi = \frac{f_{\text{BlueJEP}}}{f_{\text{JOP}}}
\]

**Relative speedup**

\[
\text{RS}_\phi = \phi \frac{CC_{\text{JOP}}}{CC_{\text{BlueJEP}}}
\]
BlueSpec System Verilog issues

**Coding** compared to a VHDL design:
- shorter development time (1/2)
- fewer lines (1/3)
- more readable, maintainable, flexible

**Test & Debug** along with the classic Verilog/VHDL ways:
- easy, software-like test-benches (*StmtFSM*)
- standalone BSV high-level executable
- probes, asserts, debug messages...

**Results** are as expected:
- larger area (needs efficient synthesis tools)
- OK performance (timing is harder to control)
The Design Rationale

Follow the classic JOP design (loosely) in order to compare BSV and VHDL design flows, but exploration led to...

- Six pipeline stages instead of four
  - simpler stages
  - shorter critical path
- Speculative execution
  - simpler control
  - no stalls on success
- OPB bus interface
  - easy integration with other OPB cores in the Xilinx EDK
  - easily replaceable
- Micro-instruction set
  - adapted for our architecture and folding
  - custom micro-assembler back-end
Finally...

Summary  We introduced BlueJEP, which:
- is a native Java embedded processor
- is specified in BlueSpec System Verilog
- has similar performance to existing solutions
- proves that BSV is perfect for fast prototyping

Extensions
- Micro-instruction Folding [under evaluation]
- Memory Management Support [completed]