Formal Verification of a Processor with Memory Management Units

I. Dalinger
dalinger@wjpserver.cs.uni-sb.de

Saarland University, Saarbrücken, Germany
Pacific National University, Khabarovsk, Russia
Overview

- Introduction
- VAMP architecture
  - Physical processor with virtual memory support
- VAMP implementation
  - Memory unit with memory management units
- Correctness criteria
  - Software Conditions
  - MMU Correctness
  - Global Correctness
- Conclusion
• In-order processors [BHK94], [BD94], [ACHK04], [VB00]
• Out-of-order processors: [McM98], [SH98]
• Out-of-order processor VAMP: [BJKLP03], [Bey04]
  ◦ open question: “.... The version of the VAMP we presented here has no hardware support for address translation. However, common operating systems require paging and address translation on the hardware ...” [Bey04]
• Address Spaces and Virtual Memory (P&P) [Hil05]
VAMP Architecture / Specification
VAMP Instruction Set Architecture

RISC architecture with DLX instruction set:

- delayed PC with one delay slot (*two* PCs)
- 3 register files: GPR, SPR, FPR
- Fixed-point arithmetic, comparison, bit-wise logical operations, shifts (32 bit)
- Floating-point arithmetic, comparison, conversion (single or double precision, IEEE-754 compliant)
- Register-indexed loads / stores (8, 16, 32, 64 bits)
- Control-flow instructions
- Special, privileged instructions
Interrupts/Exceptions Overview

- Internal interrupts
  - illegal instruction
  - misaligned memory access
  - page faults (fetch and load / store)
  - trap
  - fixed & floating point arithmetic
- External interrupts
  - reset
  - 19 external interrupts for devices
A physical machine configuration of $DLX_S$ is a 6-tuple $c_S = (GP_{R,S}, SPR_{S}, FPR_{S}, M_S, PC'_S, DPC_S)$:

- **New registers in $SPR_S$:**
  - $mode$: system mode (0) or user mode (1)
  - $pto$: Page table origin
  - $ptl$: Page table length (only for exceptions)
  - $emode$: exception mode

- **$M_S$** physical memory (usually $< 4G$)

- one step in specification executes one instruction (or causes an interrupt)
  - ! now also depends on external interrupts
Memory Accesses

- **Fetch**: $adr = DPC$ (delayed PC)
- **Load / Store**: Effective address
  
  $adr = ea = GPR(RS1) + imm$ (register relative)

$mode = 0$ (system):
- Directly access the memory at $adr$ without address translation.

$mode = 1$ (user):
- Let $va = adr$ interpreted as virtual address.
- If no exception, use translated address instead of $va$.
- Otherwise, exception.
Page-Table Lookup

Let $c$ denote configuration of $DLX_S$

- Virtual address
  \[ va = (px, bx) \]
  
  - $px$: page index
  - $bx$: byte index

- $PT_c(px) = M_{S4}(pto \cdot 4K + 4 \cdot \langle px \rangle)$

- $ppx_c(va)$ physical page index
- $v_c(va)$ valid bit ($\leftrightarrow$ page in $M_S$)
- $p_c(va)$ (write) protection bit
Let $c$ denote configuration of $DLX_s$

- Virtual address
  \[ va = (px, bx) \]
  
  $px$: page index
  $bx$: byte index

- physical memory address:
  \[ pma_c(va) = (ppx_c(va), bx) \]

- Let $mw = 0$ for fetch / load and $mw = 1$ for write.

- exception on address translation:
  \[ excp_c = \langle px \rangle > \langle ptl \rangle \lor mw \land p_c \lor \neg v_c \]
  
  access outside the page table \hspace{1em} page protected \hspace{1em} page in SM
Hardware Implementation
VAMP Implementation

- Tomasulo scheduler
- Five functional units
- Design based on [MP00]
- Written in PVS language
- Translated to Verilog
- Verification [BJKLP03][DHP05]
Hardware Implementation
• Build hardware boxes MMU & a few gates
Memory Interface

CPU – memory system protocol

- Read and write are mutually exclusive
- The interface is live
- The interface inputs are stable if the interface is busy

Inserting two MMUs:

Must obey the protocol at both sides!
MMU controlled by finite state diagram (FSD)
VAMP Correctness
VAMP Correctness

- Software conditions
- Local correctness of MMU
- Global Correctness of VAMP
Software conditions

Self-Modifying Code: executed instruction in the memory unit could overwrite instruction that is already fetched

- insert $\text{sync}$-instruction before fetch of modified address
- $\text{sync}$ stalls following fetch and waits for writeback of all previous instructions
- instruction after $\text{sync}$ fetched correctly
- required to use the $\text{sync}$ instruction
MMU Correctness

- Functional correctness
  - Untranslated read
  - Untranslated write
  - Translated read
  - Translated write
- Liveness
- Satisfies the CPU – memory system protocol
Translated read lemma:
Let \( ts \) and \( te \) denote the start and end of a translated read request, no excep. Let \( t \in \{ts, \ldots, te\} \).

Hypothesis:
The following signals do not change in cycles \( t \) (i.e. \( X^t = X^{ts} \)):

1. \( mr^t_p, mw^t_p, va = addr^t_p \) : from datapath
2. \( pto^t, ptt^t, mode^t \) : from datapath (SPR)
3. \( PT^t(px) \) : page table entry
4. \( M^t(pma(va)) \) : content of the memory

Claim: \( din^{te}_p = M^{ts}(pma^{ts}(va)) \)

Proof: with the theorem prover PVS
1. \(mr_p, mw_p, va = addr_p\): MMU keeps \(busyp\) active during translation

2. \(pto, ptl, mode\): Extra gates: normal \(sync\) before issue not enough.
   If \(rfe\) or update to \(\{mode, pto, ptl\}\) in the pipeline, \textit{must stall} translation of fetch of next instruction.

3. \(PT(px)\): User program cannot modify PT. Preceding system code terminated (by \(rfe\))

4. \(Mt(pma(va))\):
   Fetch: correct by \(sync\) condition
   Load: non-pipelined, in-order memory unit
Global Hardware Correctness

Define scheduling functions $sI(k, T) = i$: instruction $I_i$ is in stage $k$ during cycle $T$.

- Key concept for hardware verification of the VAMP

Correctness criteria:

- $R^i_S$: specified register before step $i + 1$
- $R^T_I$: hardware register in cycle $T$
- $sI(wb, T) = i$: instruction index for the write-back stage

Then: $GPR[j]^T_I = GPR[j]^i_S$;
$SPR[j]^T_I = SPR[j]^i_S$;
$FPR[j]^T_I = FPR[j]^i_S$
Correctness criteria:

- $sI(\text{issue}, T) = i$: instruction index for the issue stage

  Then: $DPC^T_I = DPC^i_S$;
  $PC'^T_I = PC'^i_S$;

- $sI(\text{mem}, T) = i$: instruction index for the memory stage

  Then: $M^T_I = M^i_S$;
Global Hardware Correctness (fetch)

Correctness criteria:

- \( sI(\text{decode}, T) = i \): instruction index for the decode stage

Then:

\[
IR.din^T = \begin{cases} 
M^i_S(DPC^i_S) & \text{if } mode = 0 \\
M^i_S(pma^i_S(DPC^i_S)) & \text{if } mode = 1 \land \neg excp^i_S \\
0^{32} & \text{otherwise}
\end{cases}
\]

\[
IR.mal^T = DPC^i_S[0] \lor DPC^i_S[1]
\]

\[
IR.excp^T = excp^i_S
\]

Formal proof of all correctness criteria:
with the theorem prover PVS
## Verification Effort

<table>
<thead>
<tr>
<th>Index</th>
<th>Name of Theory</th>
<th># Commands</th>
<th># Lemmas</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic Circuits</td>
<td>5505</td>
<td>134</td>
</tr>
<tr>
<td>2</td>
<td>ALU</td>
<td>2202</td>
<td>76</td>
</tr>
<tr>
<td>3</td>
<td>FPU</td>
<td>25936</td>
<td>1046</td>
</tr>
<tr>
<td>4</td>
<td>Tomasulo alg.</td>
<td>5847</td>
<td>186</td>
</tr>
<tr>
<td>5</td>
<td>VAMP without <em>MMUs</em> (w/o Tomasulo alg.)</td>
<td>24753</td>
<td>495</td>
</tr>
<tr>
<td>6</td>
<td>Local <em>MMU</em></td>
<td>7399</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>VAMP with <em>MMUs</em> (w/o Tomasulo alg., MMUs)</td>
<td>36785 / ~ 12000</td>
<td>587 / ~ 90</td>
</tr>
</tbody>
</table>
Conclusion

- VAMP Specification
  - Virtual address support
  - External interrupts
- VAMP Implementation
  - Local MMU
  - Memory unit (fetch and load / store)
  - $SPR$ file
  - External interrupts
- Correctness
  - Formal correctness in PVS
Future work

- **Hardware**
  - MMU: TLB and multilevel translation
  - Computation of the $pte_a$ outside the MMU
  - Hardware disk & swap memory

- **Automated Methods**
  - Control automatons
  - Functional units

- **Software**