Formal Verification of Gate-Level Computer Systems: ECU

Sergey Tverdyshev

Saarland University,
Saarbruecken, Germany

November 18, 2009
Content

- Context: Verisoft System Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - Correctness Criterion & Proof Sketch
- Computer System Examples
- Summary
The Verisoft Stack

Verisoft:
- project funded by the BMBF
- partners from industry and academia
- goal: formal and pervasive verification of computer systems

Academic System:
- goal: implement, model, and verify a computer system from gate-level hardware to application level (email client etc.)
- system includes a processor, devices, compiler, a micro kernel, an operating system, and applications
Related Work

- Processors:
  - In-order processors [Vel05, ADJ04, MS06, ACHK04]
  - Out-of-order processors [SJ02, JM01]
  - The VAMP processor [MP00, Krö01, Jac02, BJK\textsuperscript{+} 03, DHP05, BJK\textsuperscript{+} 05, Dal06]

- Devices:
  - FIFO component of UART Esterel description [BKS03]
  - Functional verification of serial interface [ALD06]

- Computer systems
  - Verification of the famous CLI stack [BJMY89] (no devices)
  - Paper&Pencil formalisations of a system with processor and HDD [HldRP05]
  - Specification of a serial interface device and processor at assembly-level [AHK\textsuperscript{+} 07]
Content

• Context: Verisoft Stack
• Related Work
• The Computer System
  • Specification
  • Implementation
  • Correctness criterion & Proof Sketch
• Computer System Examples
• Summary
Specification

Computer system as seen by an assembly programmer:

- Assembly-level processor model with devices
- Abstraction of the gate-level model
Processor Specification

- Automaton implementing instruction set architecture (ISA)
- ISA processes one complete instruction with every step
- $c_P$ is state of the ISA automaton
- $c_P = (GPR, FPR, SPR, PC, DPC, M)$
- ISA step function $\Delta_P$ is a simple case distinction on the instruction type
- For example execution effect of \texttt{add}$?(c_P)$:

$$c'_P \cdot GPR[RD] = c_P \cdot GPR[RS1] + 32 \cdot c_P \cdot GPR[RS2]$$
Processor communicates with external devices

- Devices are mapped into the processor memory
- Processor can access them by load/store instructions on the device address space \((DA)\)
- Processor places request on \(difi = (a, req, w, data)\)

- Devices place answers on \(difo \in \mathbb{B}^{32}\)
- Devices can signal interrupts on \(eev\)
Devices Specification

- Devices are modelled within a sequential generic framework
- Every device has a unique identifier $idx \in \text{DevN}$
- $c_D: \text{DevN} \mapsto S_{idx}$ state of all devices: maps device identifiers to device states
- Devices communicate with external environment via $eifi/eifo$
- At most one device can make step
- The active device is given by processor-device identifier $idx_{PD} \in \{P\} \cup \text{DevN}$
- Step function $(c_D, difo, eifo, eev) = \Delta_D(idx_{PD}, c_D, difi, eifi)$
  - $idx_{PD} = P$ – processor accesses device. accessed device and access type is coded in $difi$
  - $eifi$ is ignored and $eifo = eifo^e$
  - $idx_{PD} \in \text{DevN}$ – device $idx_{PD}$ makes a step with the input $eifi$
  - $difi$ is ignored
**Processor+Devices Specification**

- State $c_{PD}$ combines processor and device states
- Step function $\Delta_{PD}$ combines processor and device step functions
- The progressed component is given by processor-device identifier $idx_{PD}$
  - $idx_{PD} = P \land \neg difi.req$ – processor executes an instruction without a device access
  - $idx_{PD} = P \land difi.req$ – processor executes an instruction with a device access
  - $idx_{PD} \in \text{DevN}$ – device $idx_{PD}$ makes a step with the input $eifi$
**Processor+Devices Specification**

- **PDS** – processor-device specification system
- Run is defined over *computational sequence* $\sigma \in \mathbb{N} \mapsto PD$

![Diagram of Processor+Devices Specification]

- Recursive application of $\Delta_{PD}$ for $n$ steps
- Inputs from external environment $PDS^n.eifi$ input for $n^{th}$ step
- $PDS^{(n,\sigma)}_{PD}$ – state of the processor and devices after $n$ steps
- $PDS^{(n,\sigma)}_{eifo}$ – output sequence to external environment after $n$ steps
Processor+Devices Specification

- **PDS** – processor-device specification system
- Run is defined over *computational sequence* $\sigma \in \mathbb{N} \mapsto PD$

\[
\sigma = \begin{array}{cccccccc}
P & HDD & P & SI & HDD & Kbd & P & HDD & \cdots
\end{array}
\]

- Recursive application of $\Delta_{PD}$ for $n$ steps
- Inputs from external environment $PDS^n.eifi$ input for $n^{th}$ step
- $PDS^{(n,\sigma)}_{cPD}$ – state of the processor and devices after $n$ steps
- $PDS^{(n,\sigma)}_{eifo}$ – output sequence to external environment after $n$ steps
Content

- Context: Verisoft Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - Correctness criterion & Proof Sketch
- Computer System Examples
- Summary
Processor Implementation

- Base for the system is the VAMP processor
The VAMP Processor

- Pipelined processor
- Out-of-order execution
- Precise interrupts
- Pipelined fetch with delayed PC architecture
- IEEE 754-1985 compliant (floating point)
- Address translation (virtual memory) with TLB
- Byte addressable memory
The Gate-Level Model: Memory

- Memory is not part of the processor; it is an external component (e.g. RAM)
- Memory is modelled by observing memory interfaces:

\[
M^t[a] = \begin{cases} 
    \text{mem\_init}[a] : t = 0 \\
    \text{update}(M^{t-1}[a], mifi^{t-1} \cdot bwb, mifi^{t-1} \cdot din) \\
    M^{t-1}[a] : \text{otherwise}
\end{cases}
\]

- where:
  - \(\text{write}(mifi^{t-1}, a)\) – tests if there is a write access on address \(a\) at cycle \(t - 1\)
  - \(\text{update}\) – update memory cell \(M^{t-1}[a]\) with the written data \(mifi^{t-1} \cdot \text{din}\)
Devices Interfaces

- Device can send interrupts to processor $eev[idx]$
- Processor can read and write device registers
  
  $difi = (a, req, w, din) – \text{processor request to device}$
  
  $difo = (reqp, brdy, data) – \text{device answer to processor}$
- Processor-device protocol is based on the VAMP memory interface protocol [MP00].
Devices Interfaces

- Device can send interrupts to processor $eev[idx]$
- Processor can read and write device registers
  
  $$difi = (a, req, w, din)$$ – processor request to device
  
  $$difo = (reqp, brdy, data)$$ – device answer to processor

- Processor-device protocol is based on the VAMP memory interface protocol [MP00].
Devices Implementation

- Devices are modelled within a generic framework
- Every device has a unique identifier $idx \in DevN$
- $h_D:DevN \rightarrow S_{idx}$ state of all devices: maps device identifiers to device states
- With every hardware cycle all devices make a step
- External interfaces:
  - external interface input $eifs:DevN \rightarrow Eifi_{idx}$
  - external interface output $eifos:DevN \rightarrow Eifo_{idx}$
- $(h'_D, eifos, difo, eev) = \delta_D(h_D, eifs, difi)$
- Processor-device protocol is specified by assumptions
Processor+Devices: The Gate-Level Model

- **VDI** – VAMP-Devices Implementation
- Combined system state:
  - \( VDI^t.h_P \) processor state
  - \( VDI^t.h_D \) state of all devices
  - \( VDI^t.eifis \) input from env.
  - \( VDI^t.eifos \) output to env.
- Processor and devices run in parallel
- Processor and devices are connected via a common bus
- Processor can be interrupted by the devices
- No DMA
- Memory write accesses and accesses to devices are in order
Processor+Devices: The Gate-Level Model

- **VDI** – VAMP-Devices Implementation
- Combined system state:
  - $VDI^t.h_P$ processor state
  - $VDI^t.h_D$ state of all devices
  - $VDI^t.eifs$ input from env.
  - $VDI^t.eifos$ output to env.
- Processor and devices run in parallel
- Processor and devices are connected via a common bus
- Processor can be interrupted by the devices
- No DMA
- Memory write accesses and accesses to devices are in order
Content

- Context: Verisoft Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - Correctness criterion & Proof Sketch
- Computer System Examples
- Summary
Correctness Criterion: Goal

- Goal: prove that gate-level model can be simulated by the assembly-level model.
Correctness Criterion: Processor+Devices

- Scheduling function $sI_{PD}$ maps hardware run to specification run.
- $sI_{PD}$ synchronises the time notion at the gate level with assembly-programmer level
- $sI_{PD}$ is inspired by scheduling function used for processor verification ([SH98, MP00])
- $sI_{PD}$ is based on special hardware events, e.g. instruction is processed
- $\sigma^T = sI_{PD}(T)$

Gate-level run

```
P
HDD
```

Assembly-level run

```
P  HDD  P  SI  HDD  Kbd  P  HDD  
```

$sI_{PD}$
Correctness Criterion

- Devices
  - Relate states via $\text{sim}_D(VDI^T.h_D, PDS^T.c_D)$: depends on the device instances
  - Relate inputs/outputs from/to external environment
    $\text{sync}_\text{eifs}(T)$ – guarantees the equivalence of the inputs up to $T$
    $\text{sync}_\text{eifos}(T)$ – guarantees the equivalence of the outputs up to $T$
The Simulation Theorem

Processor:

- Programmer-visible registers:
  GPR, FPR, SPR, M, PC, DPC

\[
sim_{p}(VDI^{T}.h_{p}, PDS^{\sigma^{T}}.c_{p}) \triangleq
VDI^{T}.h_{p}.GPR = PDS^{\sigma^{T}}.c_{p}.GPR \land
VDI^{T}.h_{p}.FPR = PDS^{\sigma^{T}}.c_{p}.FPR \land
VDI^{T}.h_{p}.SPR = PDS^{\sigma^{T}}.c_{p}.SPR \land
T = 0 \lor JISR^{T-1} \longrightarrow VDI^{T}.h_{p}.PC = PDS^{\sigma^{T}}.c_{p}.PC \land
T = 0 \lor JISR^{T-1} \longrightarrow VDI^{T}.h_{p}.DPC = PDS^{\sigma^{T}}.c_{p}.DPC \land
T = 0 \lor JISR^{T-1} \longrightarrow M(T) = PDS^{\sigma^{T}}.c_{p}.M
\]

- Invisible registers, e.g. registers of function units
  Correctness of these registers is not part of the top-level theorem
The Simulation Theorem

\[ \text{sync}_\text{eifis}(T) \land \]
\[ \text{sim}_P(VD^0_I.h_P, PDS^{\sigma^0} .c_P) \land \]
\[ \text{sim}_D(VD^0_I.h_D, PDS^{\sigma^0} .c_D) \]
\[ \implies \]
\[ \text{sim}_P(VD^T_I.h_P, PDS^{\sigma^T} .c_P) \land \]
\[ \text{sim}_D(VD^T_I.h_D, PDS^{\sigma^T} .c_D) \land \]
\[ \text{sync}_\text{eifos}(T) \]
Proof Sketch

The theorem is proved by induction on hardware cycles

Induction base: trivial

Induction step:

- Verify system components separately:
  - Verify VAMP against ISA: based on PVS proofs [Krö01, Bey05, Dal06]
  - Verify parallel device model against the interleaved one

- Assume-guarantee reasoning:
  - Induction hypothesis guarantees that the gate-level model is correct up to $T$
  - Use proofs for the VAMP to show the correctness of the processor part at $T + 1$
  - Use proofs for the device model to show the correctness of the device part at $T + 1$

- Formally combine the proofs to deduce the correctness of the VAMP-Device model

Formal combination of the proofs reveals an issue with to sample external interrupts:
- Processor can access devices twice at different hardware cycles
- The latter makes ISA incomplete in the scope of a computer system
- Problem is also present in open literature, e.g. MIPS-R3000 Family [Brü91] and [SP88]
Proof Sketch

The theorem is proved by induction on hardware cycles

Induction base: trivial

Induction step:

- Verify system components separately:
  - Verify VAMP against ISA: based on PVS proofs [Krö01, Bey05, Dal06]
  - Verify parallel device model against the interleaved one

- Assume-guarantee reasoning:
  - Induction hypothesis guarantees that the gate-level model is correct up to $T$
  - Use proofs for the VAMP to show the correctness of the processor part at $T + 1$
  - Use proofs for the device model to show the correctness of the device part at $T + 1$

- Formally combine the proofs to deduce the correctness of the VAMP-Device model

Formal combination of the proofs reveals an issue with to sample external interrupts:

- Processor can access devices twice at different hardware cycles
- The latter makes ISA incomplete in the scope of a computer system
- Problem is also present in open literature, e.g. MIPS-R3000 Family [Brü91] and [SP88]
Content

- Context: Verisoft Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - Correctness criterion & Proof Sketch
- Computer System Examples
- Summary
Computer System Examples

Instantiation pattern:

- Instantiate generic frameworks with the devices configurations and step functions
- Prove that devices fulfills the assumptions, e.g. processor-device protocol
- That's it.

Examples:

- Electronic control unit (ECU) for a distributed automotive system in Verisoft:
  - Automotive system consists of several ECUs
  - ECU consists of a processor and an automotive bus controller (ABC device)
  - ECUs communicate via FlexRay-like bus [Con06]
  - A distributed operating system runs on top of the system
  - Derived correctness theorem: ECU is correct with respect to its assembly specification, e.g. buffers of ABC device are read/written correctly.

- System with a serial interface [AHK⁺07] (only assembly level model, to prove driver correctness)

- System with a hard disk drive [Alk09] (only assembly level model, to prove driver correctness)
Content

- Context: Verisoft Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - Correctness criterion & Proof Sketch
- Computer System Examples
- Summary
Tools

- Isabelle/HOL – theorem prover for higher order logic
  - It’s used to implement, specify, and verify the computer system
  - It’s used in Verisoft project

- IHaVelt – hardware design and verification environment [TA08]
  - It’s built in Isabelle/HOL
  - It uses external tools (e.g. NuSMV, SAT) to verify theorems
  - It implements several abstraction and transformation algorithms
  - It can generate VHDL code
Summary

<table>
<thead>
<tr>
<th>Part</th>
<th>Person years</th>
<th>Theorems</th>
<th>Proof steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAMP (no FPU, MU) in Isabelle</td>
<td>1.5</td>
<td>1206</td>
<td>20455</td>
</tr>
<tr>
<td>Devices</td>
<td>0.5</td>
<td>52</td>
<td>967</td>
</tr>
<tr>
<td>Combining Systems</td>
<td>0.7</td>
<td>118</td>
<td>2714</td>
</tr>
<tr>
<td>Total</td>
<td>2.7</td>
<td>1376</td>
<td>24316</td>
</tr>
</tbody>
</table>

- First formally verified computer system at the gate-level
- All models are defined in Isabelle/HOL
- All proofs are carried out in Isabelle/HOL with the help of automatic tools via IHaVeIt
- The hardware designs in Isabelle/HOL can be synthesised on FPGA (e.g. ECU runs on FPGA)
- ECU has been synthesised on FPGA, the size of the design is 5,180,002 gates (without FPUs)
- Current work: connecting three ECUs (three FPGA boards); boards up and running; test results are good
The Last Slide

Gate-level run

Assembly-level run

Reordered sequence [Alk09]

OS Programmer view
References I

Mark Aagaard, Vlad C. Ciubotariu, Jason T. Higgins, and Farzad Khalvati.
Combining equivalence verification and completion functions.

Synchronization-at-retirement for pipeline verification.

Formal device and programming model for a serial interface.

ALDEC – The Design Verification Company.

UART nVS.

Eyad Alkassar.

*add title*.
Sven Beyer.

*Putting It All Together: Formal Verification of the VAMP.*

Instantiating uninterpreted functional units and memory system: functional verification of the VAMP.

Sven Beyer, Christian Jacobi, Daniel Kröning, Dirk Leinenbach, and Wolfgang Paul.
Putting it all together - formal verification of the VAMP.
*STTT Journal, Special Issue on Recent Advances in Hardware Verification*, 2005.

An approach to systems verification.

Gérard Berry, Michael Kishinevsky, and Satnam Singh.
System level design and verification using a synchronous language.
Rolf-Jürgen Brüss.

*RISC Die MIPS-R3000-Familie.*

FlexRay Consortium.

FlexRay – the communication system for advanced automotive control applications.

Iakov Dalinger.

*Formal Verification of a Processor with Memory Management Units.*

I. Dalinger, M. Hillebrand, and W. Paul.

On the verification of memory management mechanisms.

M. Hillebrand, T. In der Rieden, and W.J. Paul.

Dealing with I/O devices in the context of pervasive system verification.
References IV

Christian Jacobi.
Formal verification of complex out-of-order pipelines by combining model-checking and theorem-proving.

Ranjit Jhala and Kenneth L. McMillan.
Microarchitecture verification by compositional model checking.

Daniel Kröning.
*Formal Verification of Pipelined Microprocessors*.

Silvia Müller and Wolfgang Paul.
*Computer Architecture: Complexity and Correctness*.

Panagiotis Manolios and Sudarshan K. Srinivasan.
A framework for verifying bit-level pipelined machines based on automated deduction and decision procedures.
References V

J. Sawada and W. A. Hunt.
Processor verification with precise exceptions and speculative execution.

Verification of FM9801: An out-of-order microprocessor model with speculative execution, exceptions, and program-modifying capability.

James E. Smith and Andrew R. Pleszkun.
Implementing precise interrupts in pipelined processors.

S. Tverdyshev and E. Alkassar.
Efficient bit-level model reductions for automated hardware verification.

Miroslav N. Velev.
Automatic formal verification of liveness for pipelined processors with multicycle functional units.