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

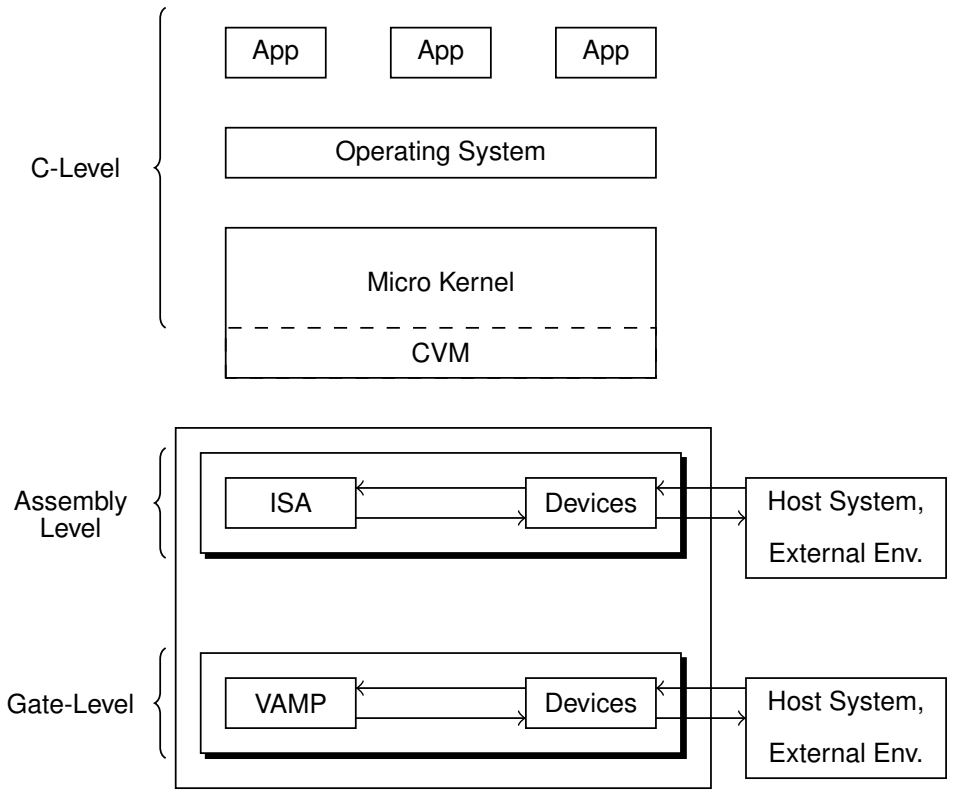
The Verisoft Stack

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+03, DHP05, BJK+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 [HIdRP05]
  - Specification of a serial interface device and processor at assembly-level [AHK+07]

Content

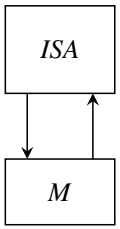
- Context: Verisoft Stack
- Related Work
- The Computer System
  - **Specification**
  - Implementation
  - Correctness criterion & Proof Sketch
- Computer System Examples
- Summary

Specification

Computer system as seen by an assembly programmer:

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

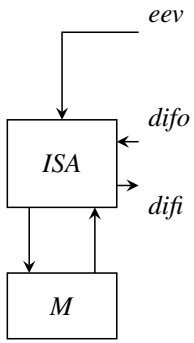
Processor Specification



- 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 Δ_P is a simple case distinction on the instruction type
- For example execution effect of $add?(c_P)$:

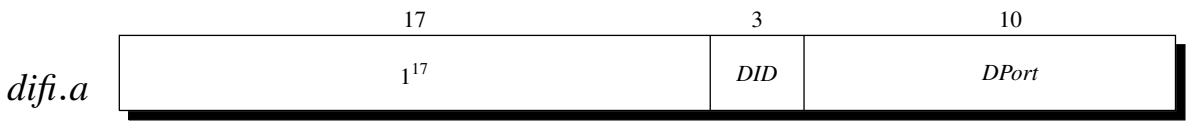
$$c'_P.GPR[RD] = c_P.GPR[RS1] +_{32} c_P.GPR[RS2]$$

Processor Specification



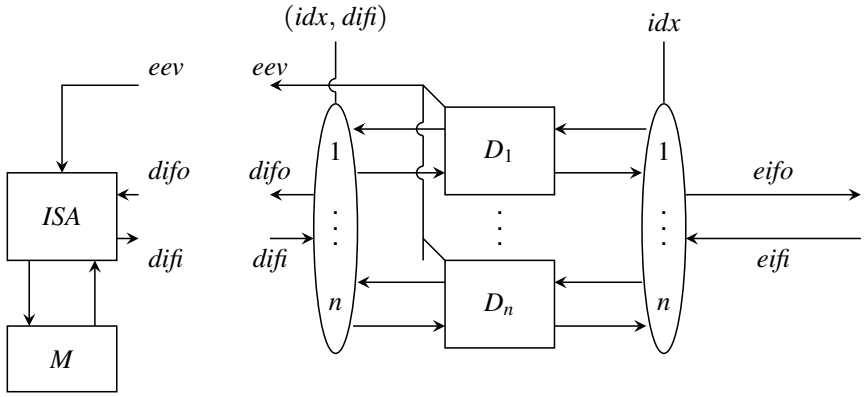
Processor communicates with external devices

- 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 $difi = (a, req, w, data)$



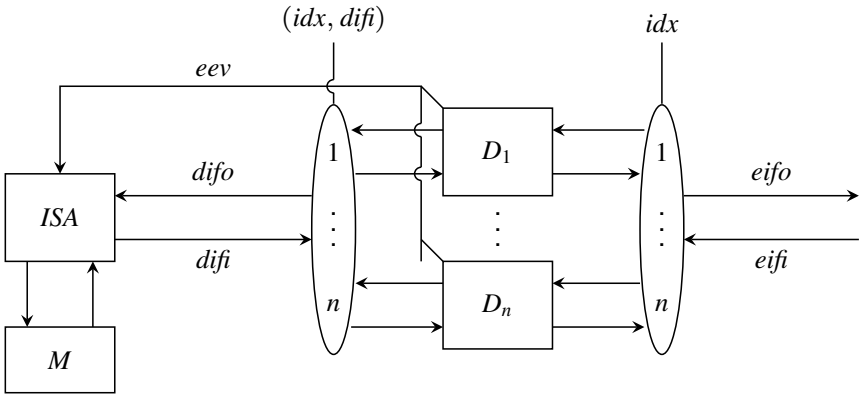
- Devices place answers on $difo \in \mathbb{B}^{32}$
- Devices can signal interrupts on eev

Devices Specification



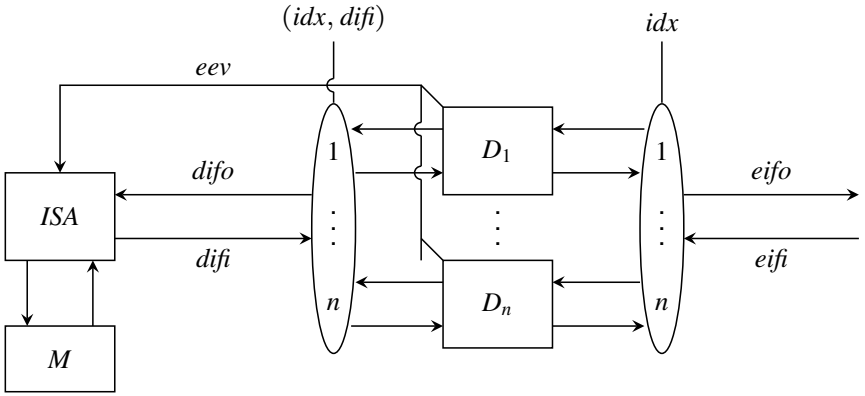
- Devices are modelled within a sequential generic framework
- Every device has a unique identifier $idx \in DevN$
- c_D:$DevN \mapsto S_{idx}$ state of all devices: maps device identifiers to device states
- Devices communicate with external environment via $eifi/eifo$
- At most one device can make step
- The active device is given by processor-device identifier $idx_{PD} \in \{P\} \cup DevN$
- 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 $eifo = eifo^{\epsilon}$
  - $idx_{PD} \in DevN$ – 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 Δ_{PD} combines processor and device step functions
- The progressed component is given by processor-device identifier idx_{PD}
  - $idx_{PD} = P \wedge \neg difi.req$ – processor executes an instruction without a device access
  - $idx_{PD} = P \wedge difi.req$ – processor executes an instruction with a device access
  - $idx_{PD} \in DevN$ – device idx_{PD} makes a step with the input $eifi$

Processor+Devices Specification

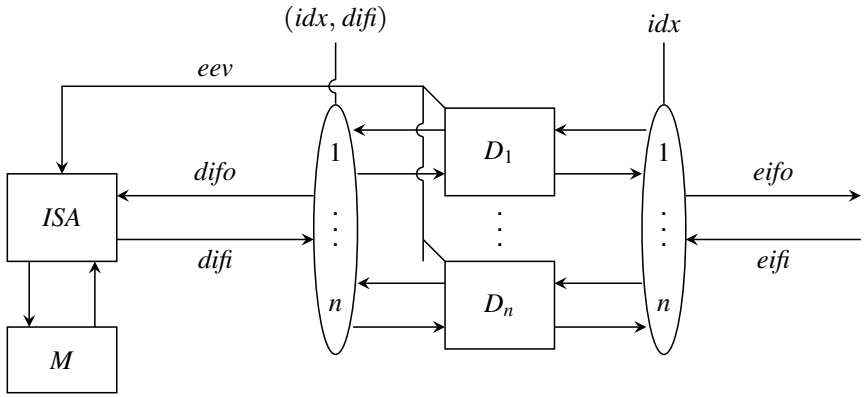


- PDS – processor-device specification system
- Run is defined over *computational sequence* $\sigma \in \mathbb{N} \mapsto PD$

σ	P	HDD	P	SI	HDD	Kbd	P	HDD	$\cdots$

- Recursive application of Δ_{PD} for n steps
- Inputs from external environment $PDS^n.eifi$ input for n^{th} step
- $PDS^{(n,\sigma)}.c_{PD}$ – 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* $\sigma \in \mathbb{N} \mapsto PD$

σ	P	HDD	P	SI	HDD	Kbd	P	HDD	$\cdots$

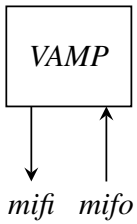
- Recursive application of Δ_{PD} for n steps
- Inputs from external environment $PDS^n.eifi$ input for n^{th} step
- $PDS^{(n,\sigma)}.c_{PD}$ – state of the processor and devices after n steps
- $PDS^{(n,\sigma)}.eifo$ – output sequence to external environment after n steps

Content

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

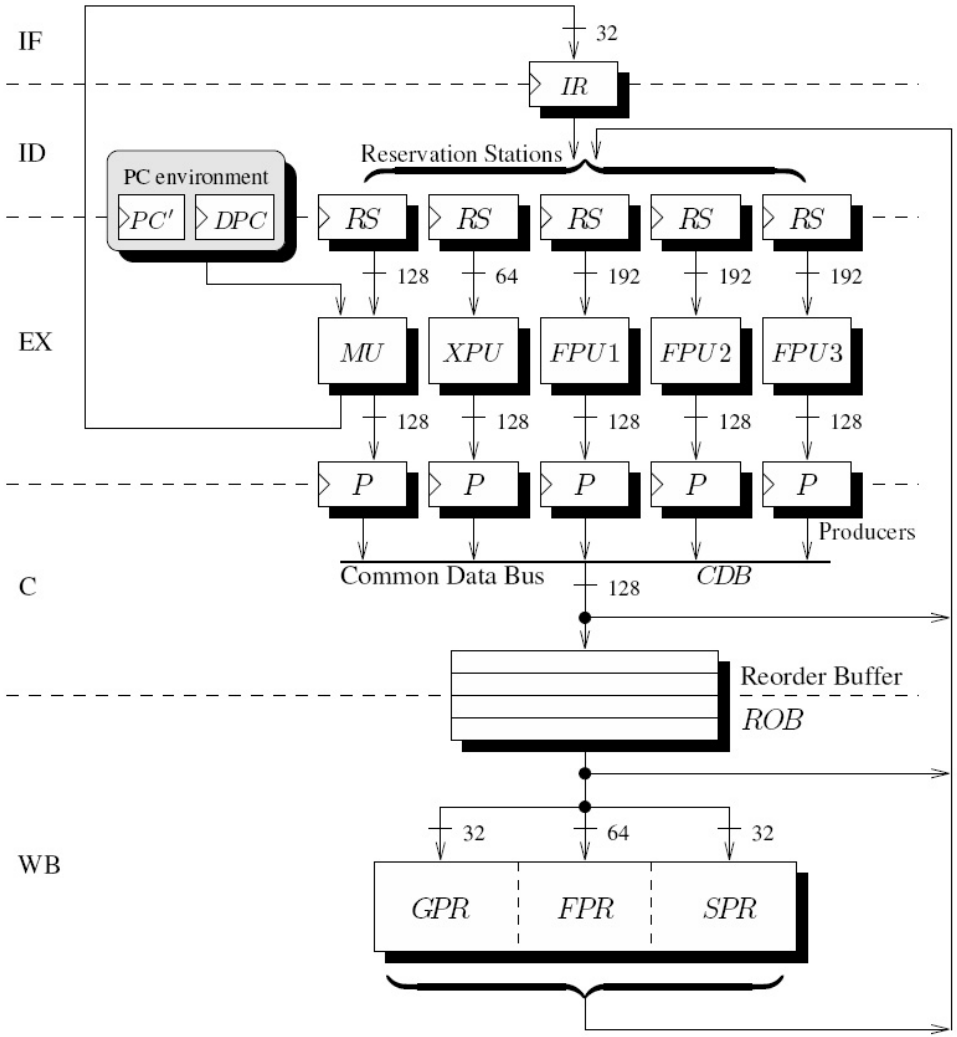
Processor Implementation

- Base for the system is the VAMP processor



The VAMP Processor

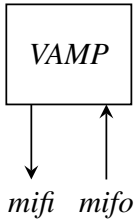
- Pipelined processor
- Out-of-order execution
- Precise interrupts
- Pipelined fetch with delayed PC architecture
- IEEE 754-1985 compliant (floating point)
- Address translation (virtual memory) with TLB
- Byte addressable memory



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:

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



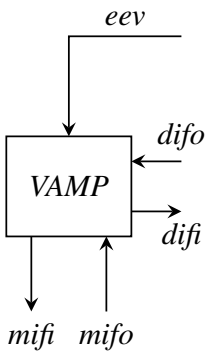
- where:
  - $write(mifi^{t-1}, a)$ – tests if there is a write access on address a at cycle $t-1$
  - $update$ – update memory cell $M^{t-1}[a]$ with the written data $mifi^{t-1}.din$

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].

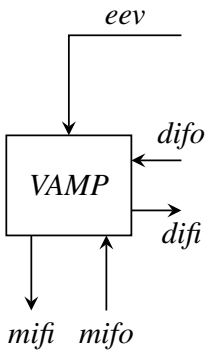


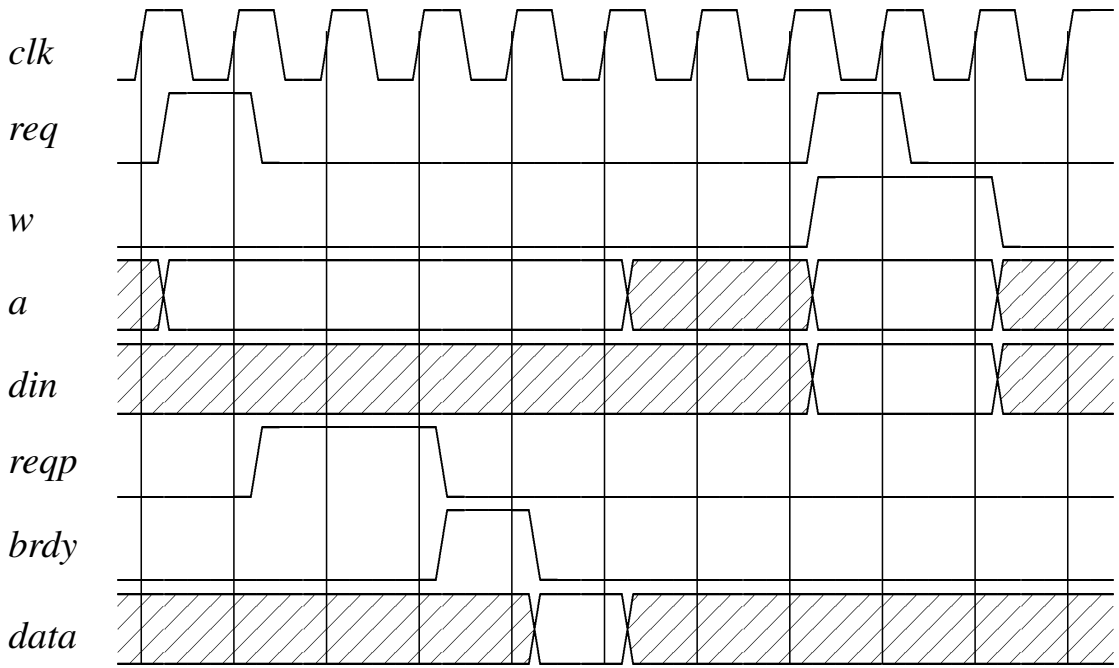
Devices Interfaces

- 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].



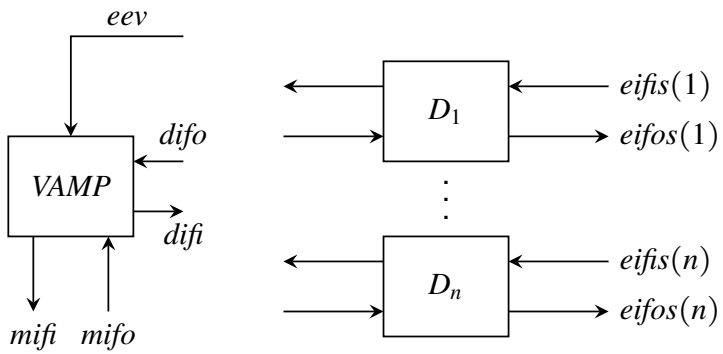


Devices Implementation

- Devices are modelled within a generic framework
- Every device has a unique identifier $idx \in DevN$
- $h_D{:}DevN \mapsto S_{idx}$ state of all devices: maps device identifiers to device states
- With every hardware cycle all devices make a step
- External interfaces:

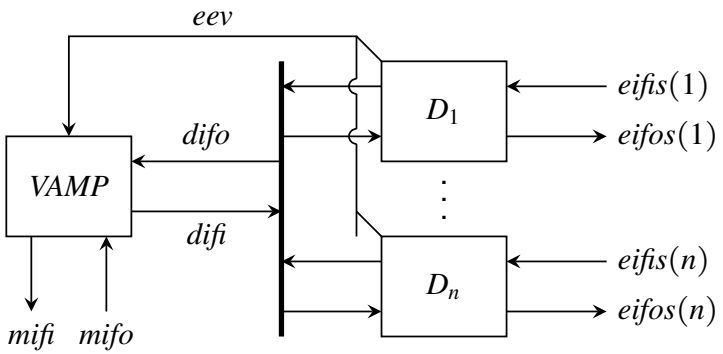
  external interface input $eifis{:}DevN \mapsto Eifi_{idx}$

  external interface output $eifos{:}DevN \mapsto Eifo_{idx}$
- $(h'_D, eifos, difo, eev) = \delta_D(h_D, eifis, difi)$
- Processor-device protocol is specified by assumptions



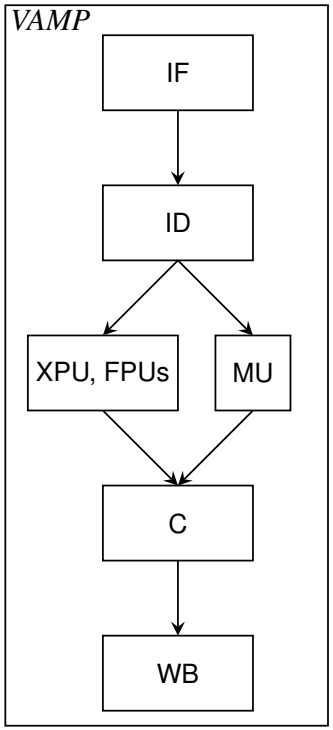
Processor+Devices: The Gate-Level Model

- 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
  - $VDI^t.eifis$ 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



Content

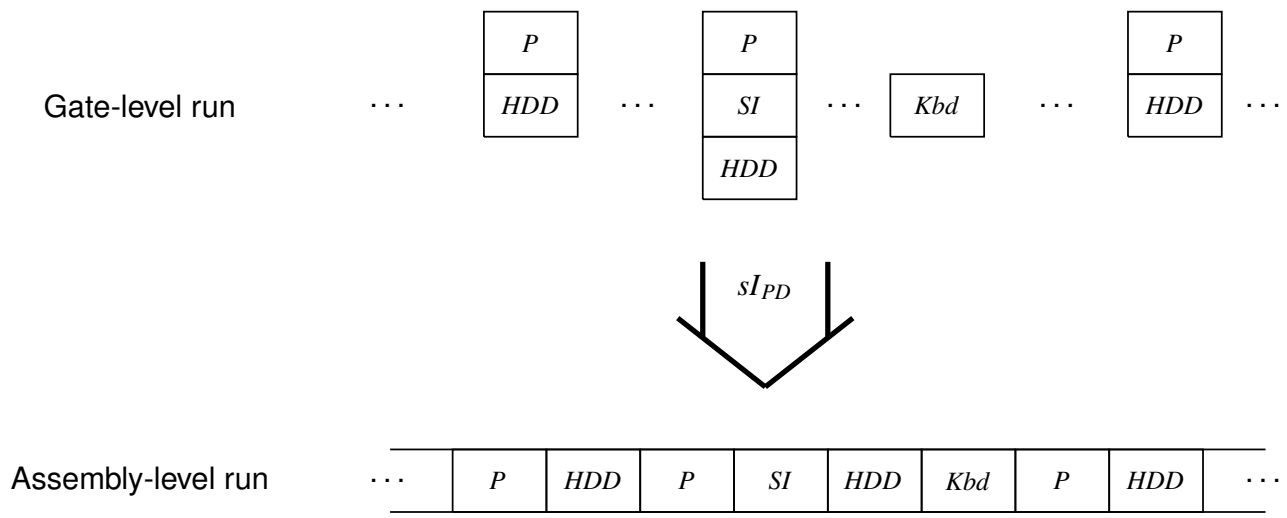
- Context: Verisoft Stack
- Related Work
- The Computer System
  - Specification
  - Implementation
  - **Correctness criterion & Proof Sketch**
- Computer System Examples
- Summary

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
- $\sigma^T = sI_{PD}(T)$



Correctness Criterion

- Devices
  - Relate states via $sim_D(VDI^T.h_D, PDS^{\sigma^T}.c_D)$: depends on the device instances
  - Relate inputs/outputs from/to external environment
    $sync\_eifis(T)$ – guarantees the equivalence of the inputs up to T
    $sync\_eifos(T)$ – guarantees the equivalence of the outputs up to T

The Simulation Theorem

Processor:

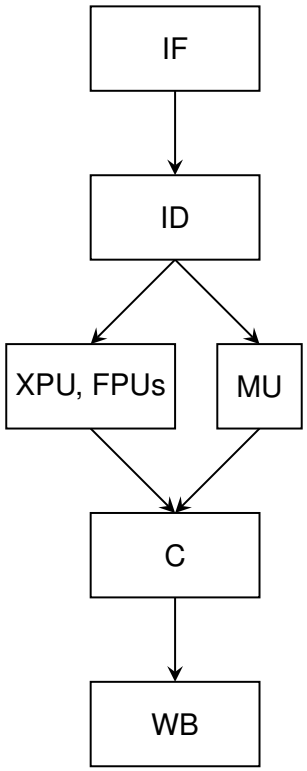
- Programmer-visible registers:

  GPR, FPR, SPR, M, PC, DPC

$$
\begin{aligned}
&sim_P(VDI^T.h_P, PDS^{\sigma^T}.c_P) \triangleq \\
&\quad VDI^T.h_P.GPR = PDS^{\sigma^T}.c_P.GPR \wedge \\
&\quad VDI^T.h_P.FPR = PDS^{\sigma^T}.c_P.FPR \wedge \\
&\quad VDI^T.h_P.SPR = PDS^{\sigma^T}.c_P.SPR \wedge \\
&\quad T = 0 \vee JISR^{T-1} \longrightarrow VDI^T.h_P.PC = PDS^{\sigma^T}.c_P.PC \wedge \\
&\quad T = 0 \vee JISR^{T-1} \longrightarrow VDI^T.h_P.DPC = PDS^{\sigma^T}.c_P.DPC \wedge \\
&\quad T = 0 \vee JISR^{T-1} \longrightarrow M(T) = PDS^{\sigma^T}.c_P.M
\end{aligned}
$$

- Invisible registers, e.g. registers of function units

  Correctness of these registers is not part of the top-level theorem



The Simulation Theorem

$$
\begin{aligned}
&sync\_eifis(T) \wedge \\
&sim_P(VDI^0.h_P, PDS^{\sigma^0}.c_P) \wedge \\
&sim_D(VDI^0.h_D, PDS^{\sigma^0}.c_D) \\
&\Longrightarrow \\
&\quad sim_P(VDI^T.h_P, PDS^{\sigma^T}.c_P) \wedge \\
&\quad sim_D(VDI^T.h_D, PDS^{\sigma^T}.c_D) \wedge \\
&\quad sync\_eifos(T)
\end{aligned}
$$

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

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

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
- IHaVeIt – 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

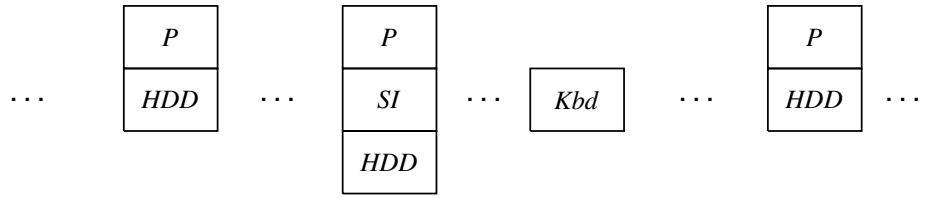
Summary

Part	Person years	Theorems	Proof steps
VAMP (no FPU, MU) in Isabelle	1.5	1206	20455
Devices	0.5	52	967
Combining Systems	0.7	118	2714
Total	2.7	1376	24316

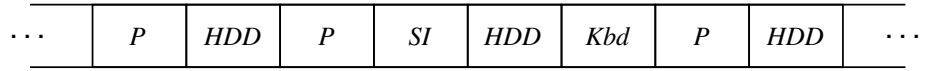
- First formally verified computer system at the gate-level
- All models are defined in Isabelle/HOL
- All proofs are carried out in Isabelle/HOL with the help of automatic tools via IHaVeIt
- The hardware designs in Isabelle/HOL can be synthesised on FPGA (e.g. ECU runs on FPGA)
- ECU has been synthesised 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

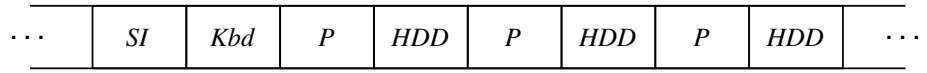
Gate-level run



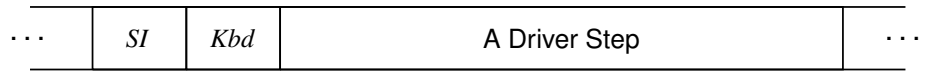
Assembly-level run



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.

In *FMCAD*, pages 98–112, 2004.

Mark Aagaard, Nancy A. Day, and Robert B. Jones.

Synchronization-at-retirement for pipeline verification.

In *FMCAD*, pages 113–127, 2004.

E. Alkassar, M. Hillebrand, S. Knapp, R. Rusev, and S. Tverdyshev.

Formal device and programming model for a serial interface.

In B. Beckert, editor, *Proceedings, 4th International Verification Workshop (VERIFY), Bremen, Germany*, pages 4–20. CEUR-WS Workshop Proceedings, 2007.

ALDEC – The Design Verification Company.

UART nVS.

`http://www.aldec.com/products/ipcores/_datasheets/nSys/UART_nVS.pdf`, 2006.

Eyad Alkassar.

add title.

PhD thesis, Saarland University, Saarbrücken, 2009.

References II

Sven Beyer.

Putting It All Together: Formal Verification of the VAMP.

PhD thesis, Saarland University, Saarbrücken, 2005.

S. Beyer, C. Jacobi, D. Kröning, D. Leinenbach, and W.J. Paul.

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

In D. Geist and E. Tronci, editors, *CHARME 2003*, volume 2860 of *LNCS*, pages 51–65. Springer, 2003.

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.

William R. Bevier, Warren A. Hunt Jr., J. Strother Moore, and William D. Young.

An approach to systems verification.

Journal of Automated Reasoning, 5(4):411–428, 1989.

Gérard Berry, Michael Kishinevsky, and Satnam Singh.

System level design and verification using a synchronous language.

In *ICCAD*, pages 433–440, 2003.

References III

Rolf-Jürgen Brüss.

RISC Die MIPS-R3000-Familie.

Architektur, Systembausteine, Compiler, Tools, Anwendungen. Siemens, 1991.

FlexRay Consortium.

FlexRay – the communication system for advanced automotive control applications.

`http://www.flexray.com/`, 2006.

Iakov Dalinger.

Formal Verification of a Processor with Memory Management Units.

PhD thesis, Saarland University, Saarbrücken, 2006.

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

On the verification of memory management mechanisms.

In D. Borrione and W. Paul, editors, *CHARME 2005*, LNCS, pages 301–316. Springer, 2005.

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

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

In *23nd IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 2005), 2-5 October 2005, San Jose, CA, USA, Proceedings*, pages 309–316. IEEE, 2005.

References IV

Christian Jacobi.

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

In *Computer Aided Verification (CAV 02)*, volume 2404 of *LNCS*, pages 309–323. Springer, 2002.

Ranjit Jhala and Kenneth L. McMillan.

Microarchitecture verification by compositional model checking.

In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 396–410, London, UK, 2001. Springer.

Daniel Kröning.

Formal Verification of Pipelined Microprocessors.

PhD thesis, Saarland University, Saarbrücken, 2001.

Silvia Müller and Wolfgang Paul.

Computer Architecture: Complexity and Correctness.

Springer, 2000.

Panagiotis Manolios and Sudarshan K. Srinivasan.

A framework for verifying bit-level pipelined machines based on automated deduction and decision procedures.

Journal of Automated Reasoning, 37(1–2):93–116, 2006.

References V

J. Sawada and W. A. Hunt.

Processor verification with precise exceptions and speculative execution.

In *Proc. 10th International Computer Aided Verification Conference*, pages 135–146, 1998.

Jun Sawada and Warren A. Hunt Jr.

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

Formal Methods in System Design, 20(2):187–222, 2002.

James E. Smith and Andrew R. Pleszkun.

Implementing precise interrupts in pipelined processors.

IEEE Trans. Comput., 37(5):562–573, 1988.

S. Tverdyshev and E. Alkassar.

Efficient bit-level model reductions for automated hardware verification.

In *15th International Symposium on Temporal Representation and Reasoning: TIME 2008, to appear*. IEEE, 2008.

Miroslav N. Velev.

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

In *CHARME*, pages 97–113, 2005.