



Development and Use of an Instruction Set Simulator of 68000-compatible Processor Core

Filip Rak

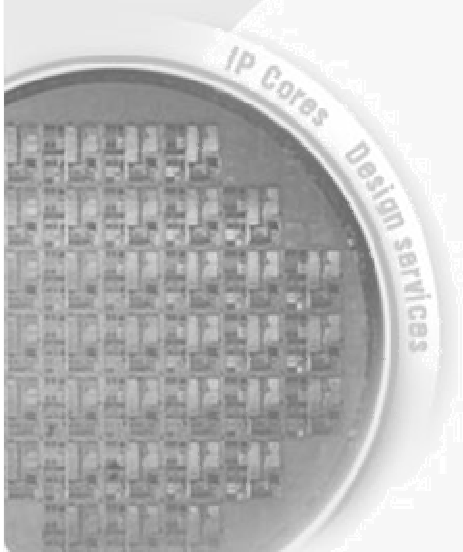
Evatronix SA, Bielsko-Biala, Poland

Wojciech Sakowski

Institute of Electronics,
Silesian University of Technology
Gliwice, Poland

IP 2007

Grenoble, France



Outline

- ▶ C68000-OCDS CPU core overview
- ▶ C68000-ISS architecture
- ▶ C68000-ISS usage and performance
- ▶ Interaction of C68000-ISS and C1394A-TLM
- ▶ Conclusions

C68000-OCDS CPU overview

- ▶ CPU core designed at Evatronix as MC68000™ replacement
- ▶ Features:
 - fully 32-bit internal architecture
 - external 16-bit data and 23/32-bit address bus
 - 55 instructions and 14 possible addressing modes
 - eight 32-bit data and eight 32-bit address registers
 - seven priority interrupt controller
 - On-Chip Debug System to control program flow

C68000-ISS model overview

- ▶ ISS developed at Evatronix as an (abstract) functional equivalent of C68000-OCDS
- ▶ Features:
 - instruction-accurate interpretive ISS without time dependencies (PV model)
 - raw C++ language with SystemC TLM wrapper
 - functional conformity with C68000-OCDS without micro-architectural details
 - use of native C++ types
 - optimization for high simulation speed
 - flexible external C++ / TLM interfaces
 - modular construction

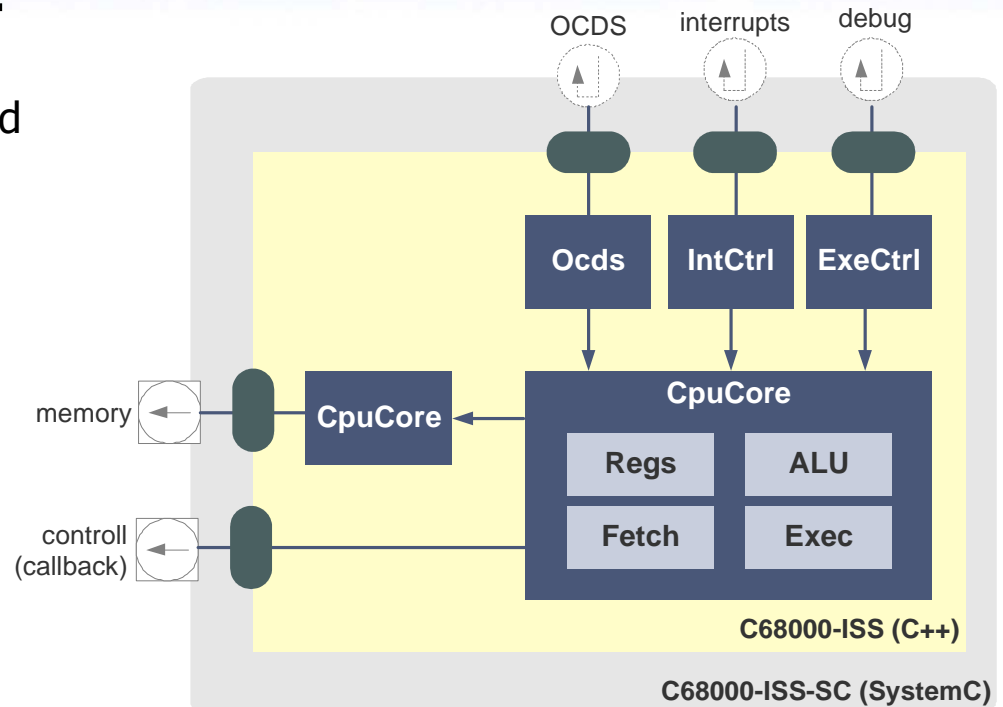
C68K_ISS architecture

► Division into sub-modules:

- **CpuCore** – main CPU logic
- **ExeCtrl** – Instruction flow and debug features
- **MemAcc** – external memory access and data breakpoints
- **IntCtrl** – Incoming interrupt request service
- **Ocds** – OCDS functionality

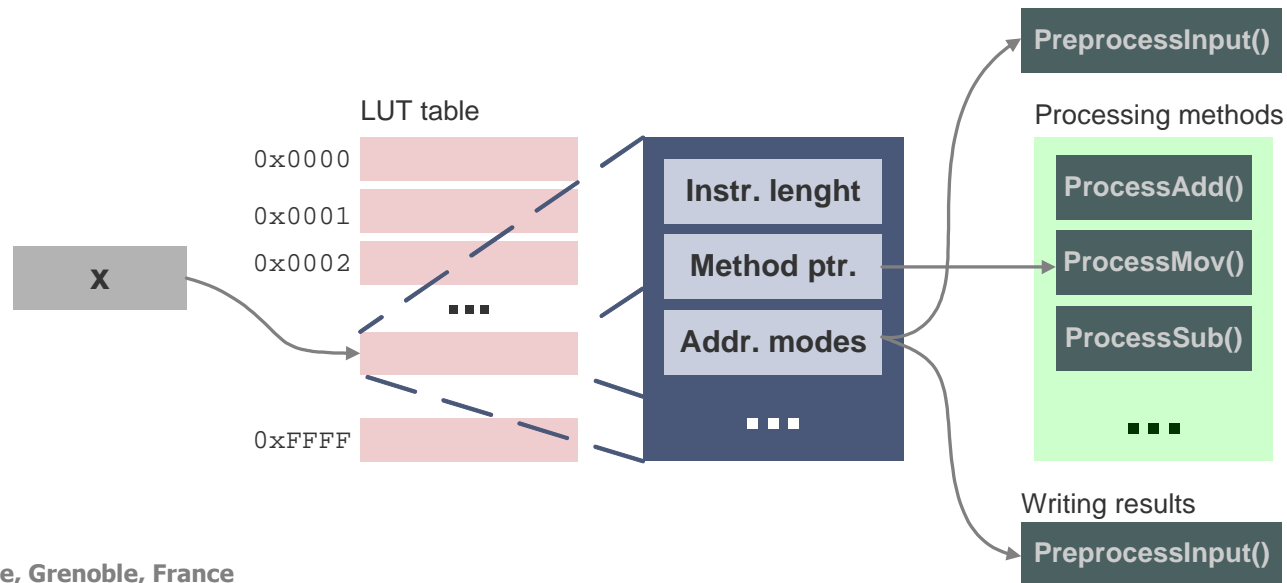
► External interfaces:

- external memory interface
- interrupt interface
- debug interface
- control interface



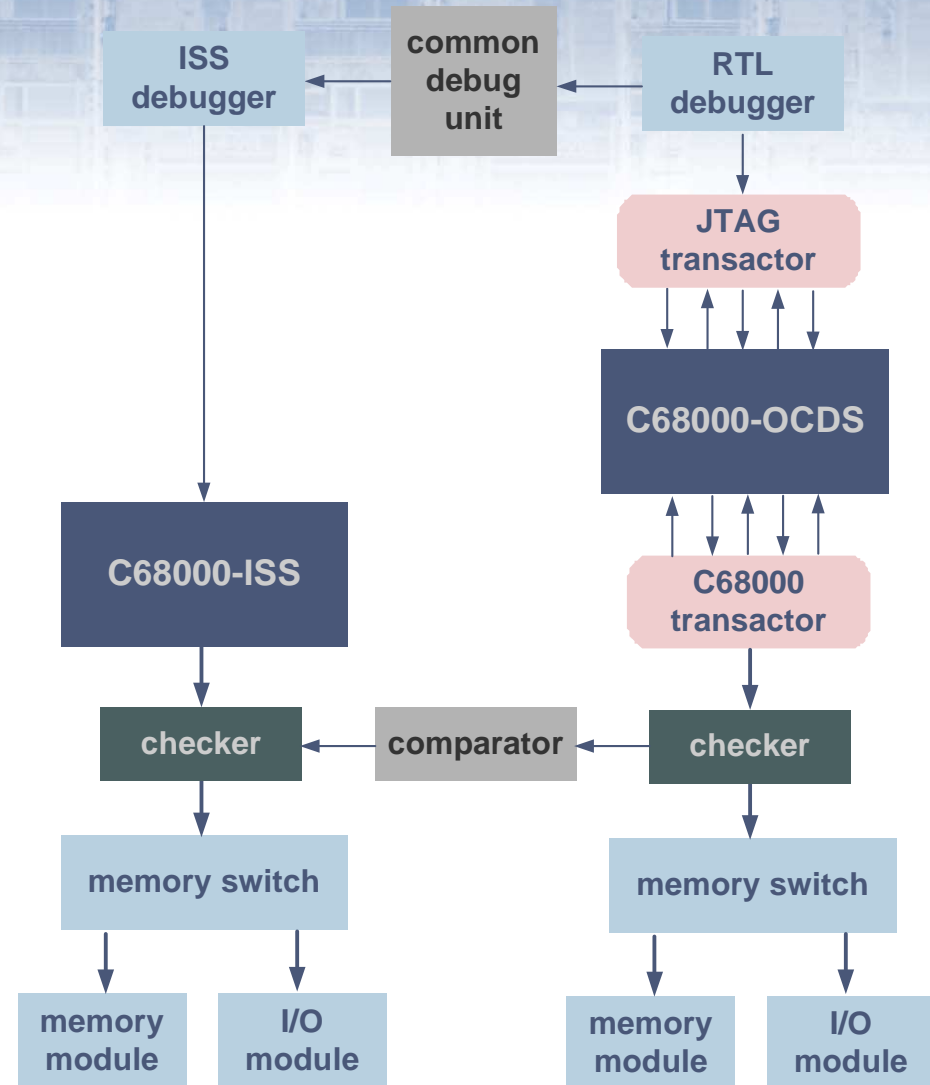
C68000-ISS architecture details

- ▶ Instruction decoder is implemented as Look-Up Table, which is indexed by the first instruction word
- ▶ Data kept in the descriptor:
 - op-code kind (ADD / MOV / ...)
 - method pointer
 - number of instruction words (total / for addressing mode)
 - operand size (for source / destination)
 - addressing mode (source / destination) with register index
 - auxiliary value (register index or immediate value)
 - instruction flags (read / write source / destination)



C68000-ISS testing environment

- ▶ Parallel simulation of C68000-ISS and an original RTL code (C68000-OCDS core)
- ▶ Execution of the same instruction streams
- ▶ Transactor used to enable on the fly checking of the functional equivalence of both models
- ▶ use of virtual debug unit for inspection of both models



C68000-ISS usage

– embedded software design and verification

- ▶ C68000-ISS instead of hardware prototype
 - Enables HW / SW co-simulation
 - Direct interaction with sw debugging environment
 - Possible interaction with RTL or TLM models of other SoC modules
- ▶ Advantages & disadvantages of modelling SoCs in systemC
 - much faster simulation than with RTL equivalent
 - easier integration
 - easy CPU state introspection and program execution flow
 - Performance estimation is not as exact as with RTL or hw prototype

C68000-ISS performance

- ▶ Environments designed for performance tests:
 - **simple**: ISS + memory module
 - **complex**: ISS + memory switch + memory module + I/O controller
- ▶ Test results:

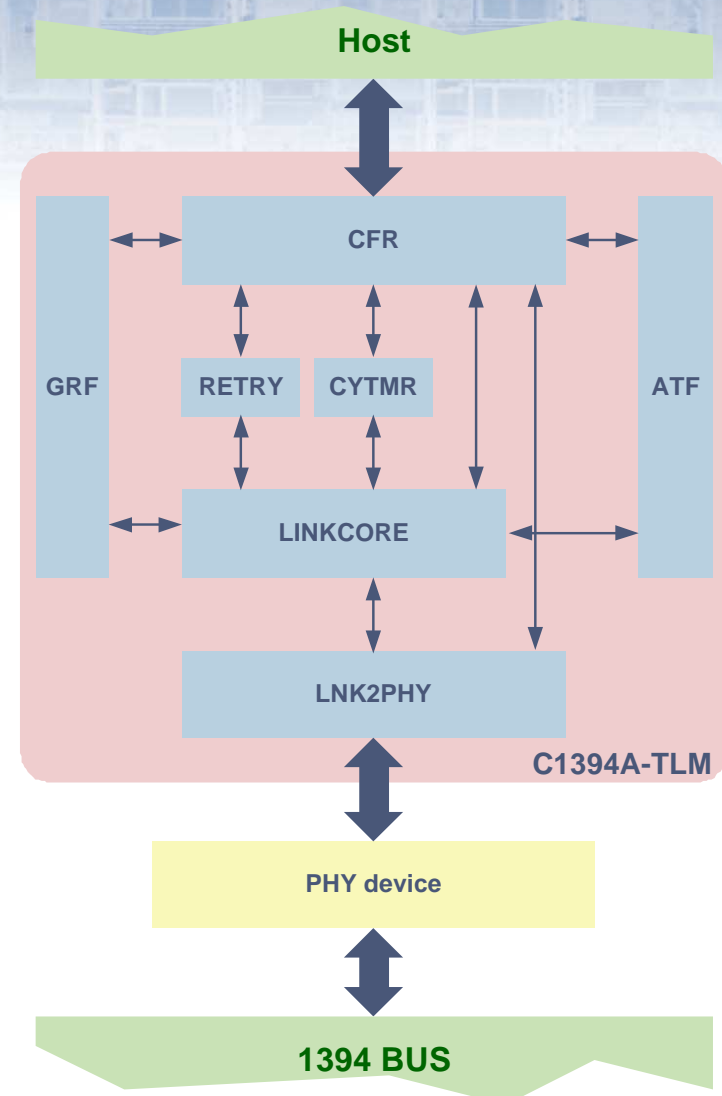
Test type		Instr / sec
C++	simple	3.33 M
	complex	2.86 M
SystemC	simple	1.33 M
	complex	0.95 M

Integration of C68000-ISS with TLMs

- ▶ Static design without internal SystemC threads / events
- ▶ No internal time statements
- ▶ Fully controlled by the external debug interface
- ▶ Flexible external interfaces:
 - external memory access
 - transport layer - using `tlm_transport_if` interface
 - user layer with `ReadMem / WriteMem` API
 - interrupt request
 - transport layer – using `tlm_transport_if` interface
 - user layer with `RequestInt` API
- ▶ C1394A-TLM used for “proof of concept”

C1394A-TLM overview

- ▶ C1394A – Link layer controller core implementing IEEE 1394-1995 Std. and IEEE 1394-2000a Std., written in VHDL
- ▶ C1394A-TLM – PV model of C1394A, written in SystemC
- ▶ Features:
 - C1394A functionality (without Data Mover interface)
 - pure PV architecture without time dependencies
 - data granularity at packet / quadlet level
 - register bit accuracy
 - TLM compatible external interfaces
- ▶ Architecture – division into seven sub-modules:
 - **Cfr** – access to control registers
 - **LinkCore** – main controller logic
 - **Link2Phy** – communication with PHY device
 - **Retry** – packets retransmission
 - **CyTmr** – bus time management
 - **Atf** – asynchronous transmission FIFO buffer
 - **Grf** – general reception FIFO buffer



C1394A Software Stack overview

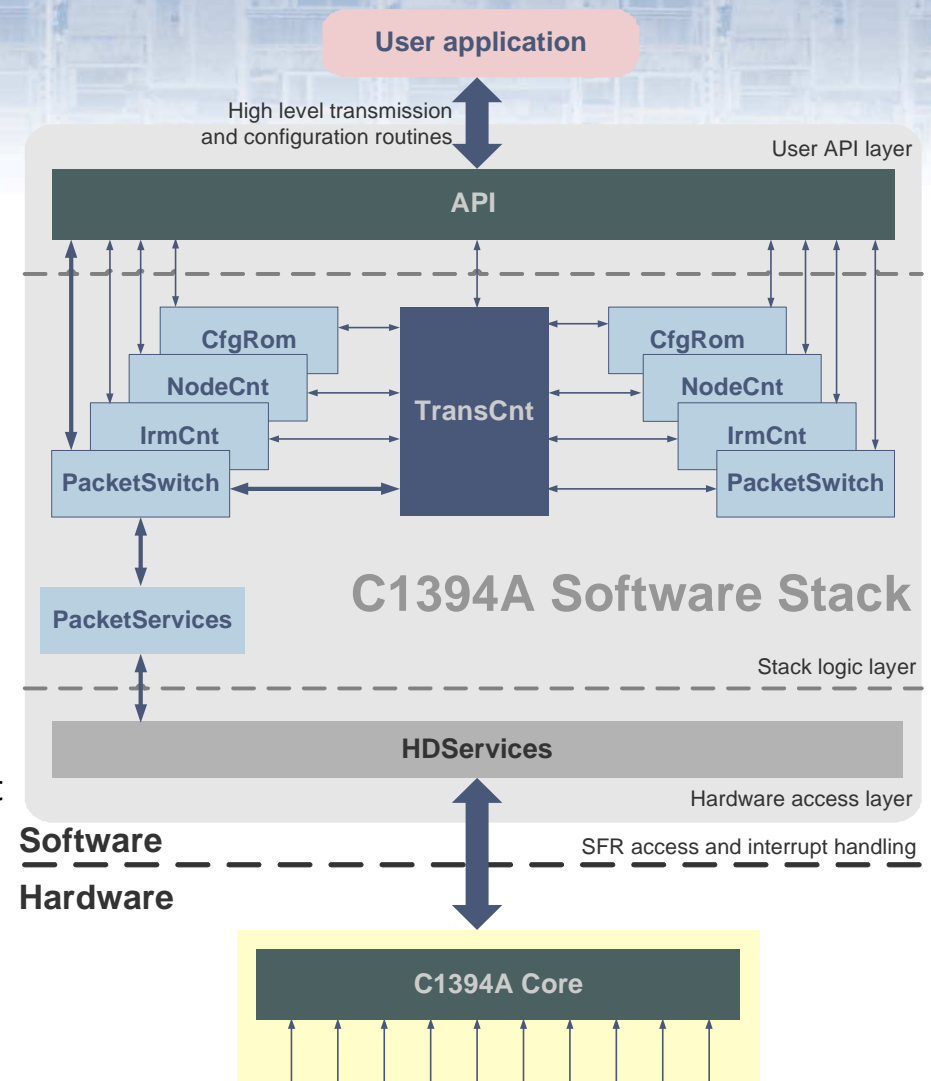
► C implementation of higher 1394 bus protocol layers

► Division into sub-modules:

- architecture independent parts
 - API – Set of functions provided for the user
 - TransCnt – Transaction handling
 - CycleCnt – Bus time managing
 - BusCnt – Bus topology information
 - IsoCnt – Isochronous transmission related routines
 - DeviceCnt – Device data extraction functionality
 - CfgRom – Configuration ROM handling
 - NodeCnt – Local node CSR access handling
 - IrmCnt – Isochronous Resource Manager routines
 - PacketSwitch – Packet identification
 - PacketServices – C1394A communication functionality
- architecture dependent functionality
 - HdServices – Direct access to the C1394A device

► Features

- pure ANSI C with separated architecture-dependent part
- support for asynchronous, streaming and PHY transmissions
- implementation of all necessary node's CSRs
- easy extensibility via callback mechanism



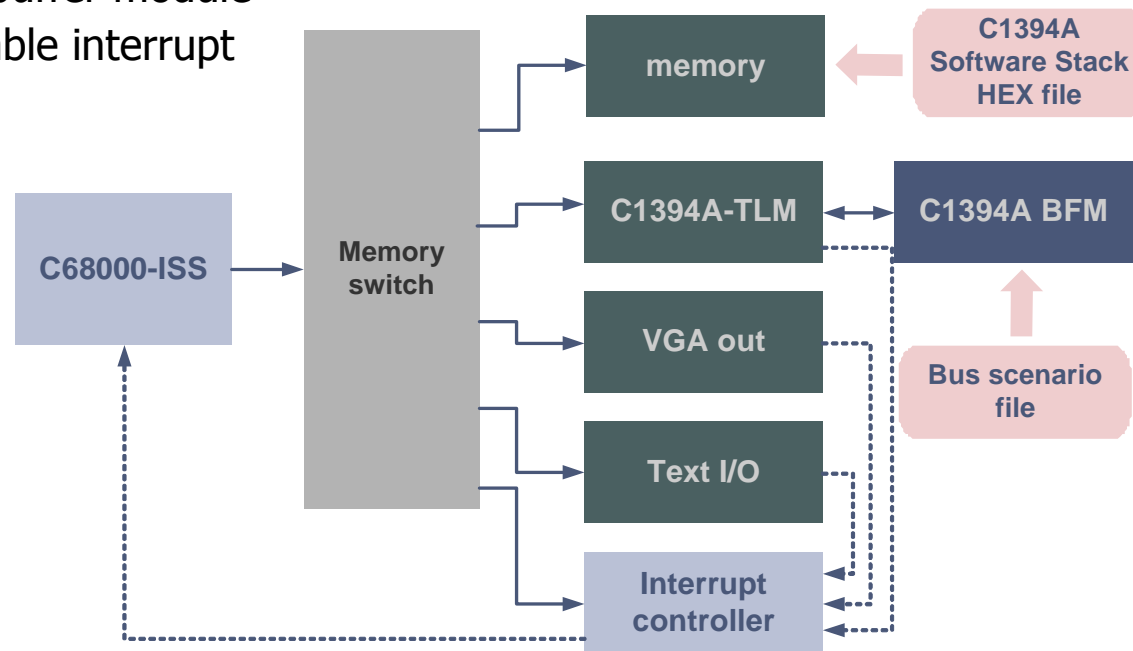
C68000-ISS + C1394A-TLM "SoC model"

► „SoC model“ consists of:

- C68000-ISS CPU
- memory switch
- memory module
- C1394A-TLM controller
- text I/O module
- VGA framebuffer module
- programmable interrupt controller
- 1394 BFM

► Purpose of the " SoC model"

- Evaluation of the simulation speed increase when compared to RTL model
- Evaluation of ISS usefulness as software driver development platform



Conclusions

- ▶ (Our C68000-ISS) Instruction Set Simulator together with transaction level models of peripheral devices seems to be useful for building virtual software development platform
- ▶ Such platforms are flexible and effective in early (pre-silicon) embedded software development and verification process
- ▶ While obviously useful for SoC integrator it may also bring benefits to IP core developer (testing software drivers associated with peripheral controller compiled to target architecture)



Thank you for your attention !