

# System Level Design Experiences and the Need for Standardization

Vesa Lahtinen  
Nokia, Technology platforms

14.11.2006, Tampere  
International Symposium on System-on-Chip

# Structure

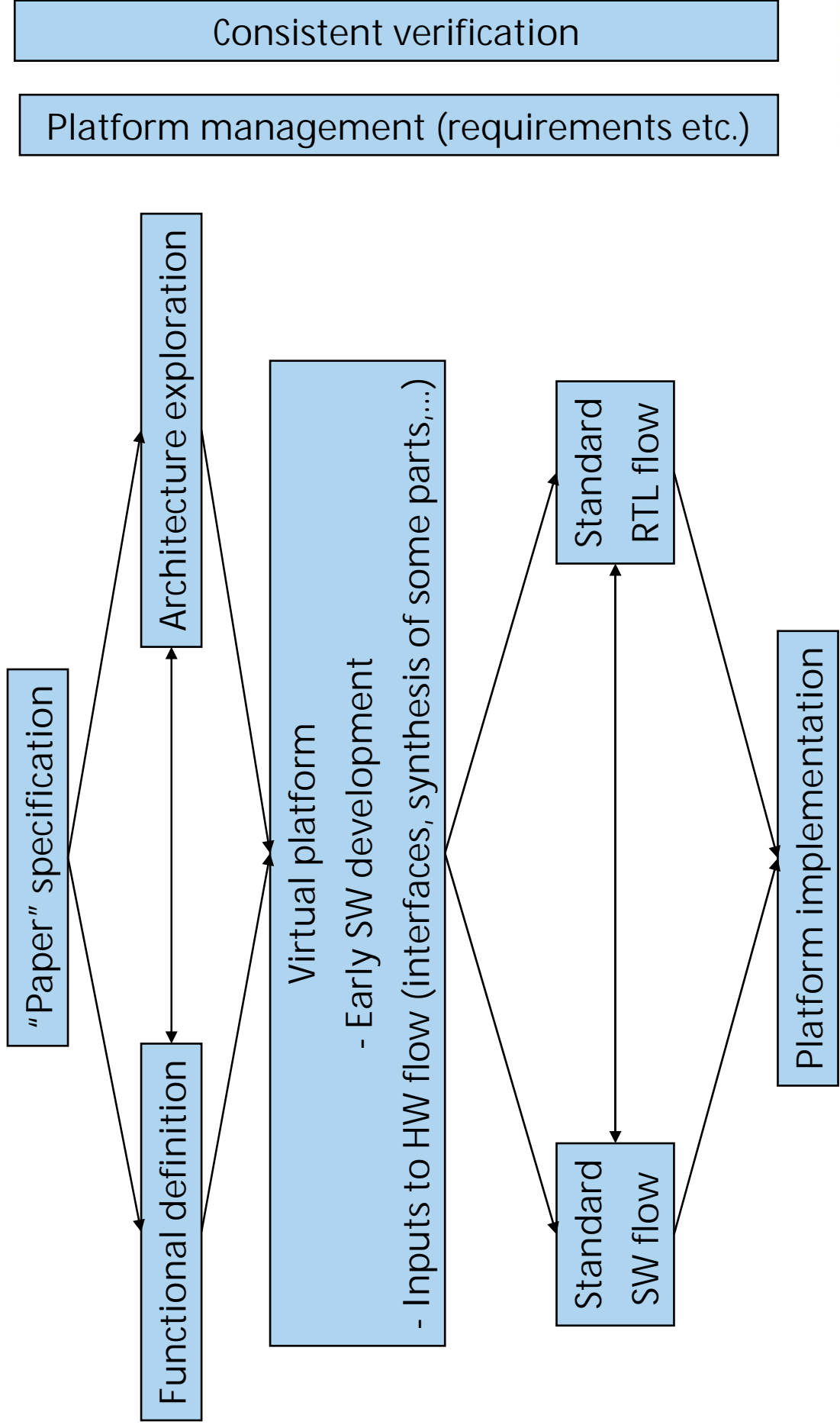
- What is system-level design or ESL?
- Nokia experiences in system-level design, what are the findings so far?
- Why to standardize?
- Where to standardize?
- Conclusions

# System level design

# ESL (=Electronic System Level) design?

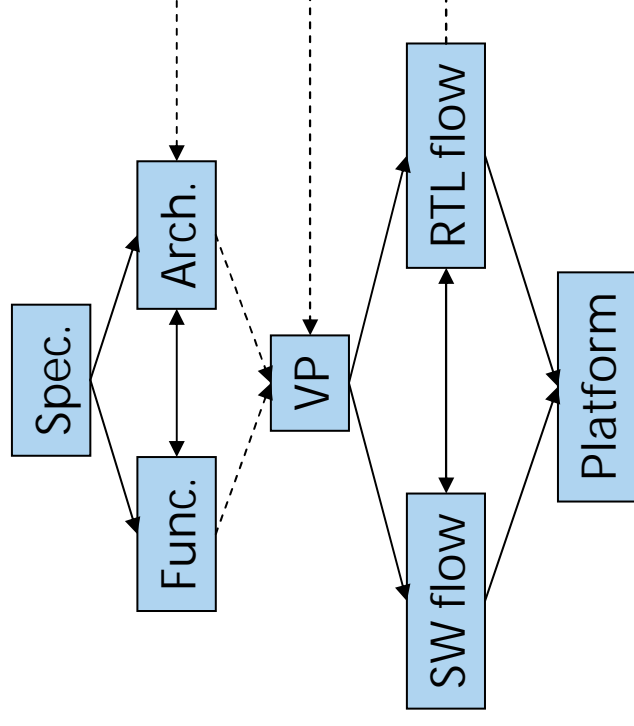
- Many definitions: ~ Design methodology for the concurrent design of HW and SW parts of an electronic product using high abstraction levels, usually based on SystemC
- Some ESL use cases
- 1. Virtual platforms
  - High-level (SystemC) model of a platform engine
  - Early HW feedback and SW development with fast simulation and full visibility/controllability
- 2. Architecture exploration
  - Gives early analysis capability of HW architecture to catch bugs and bottlenecks
  - Emphasis is not on functionality
- 3. Platform management
  - Help in building and maintaining complex HW platforms
  - Offers a common (Spirit XML) metadata format and configuration interface => tool interoperability
- 4. Verification
  - Offers a consistent verification view from high to low levels
  - Equivalence between models on different abstraction levels
- 5. High-level synthesis
  - Offers productivity gains with better synthesis results
- 6. ...

# ESL flow (many possibilities, this is just an example)



# Common flow misconceptions

- New projects rarely (=never) start from scratch
  - Legacy HW and SW is a big issue
  - How to build up the model base (of RTL already existing?)
- Proprietary point tools are not acceptable
  - Tools having their own input language and no output to the “standard” flow
  - Standard solutions preferred, even when they are not the most optimal ones

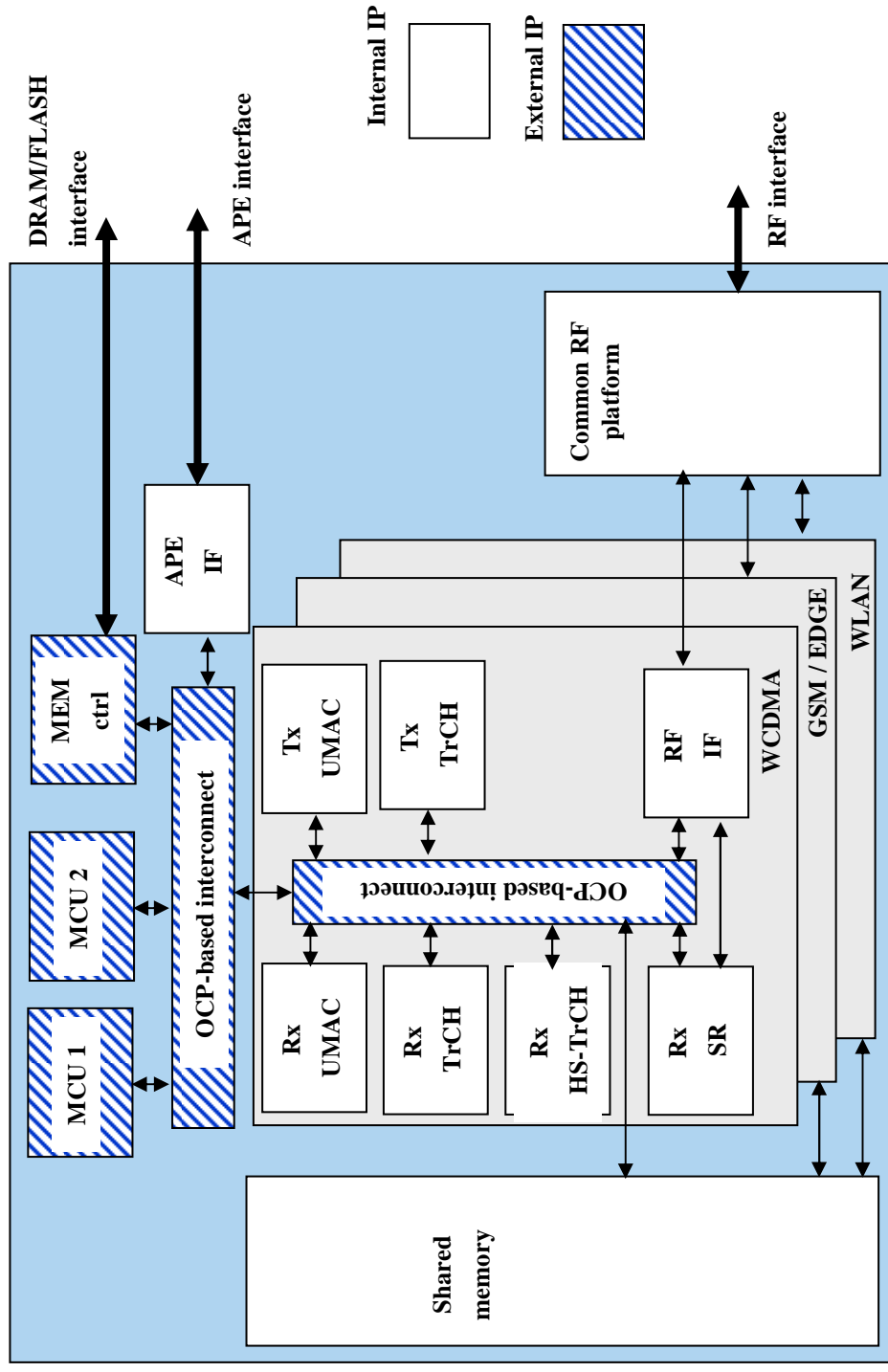


# Nokia experiences: Case example

# The NeMo (New Modem) architecture project

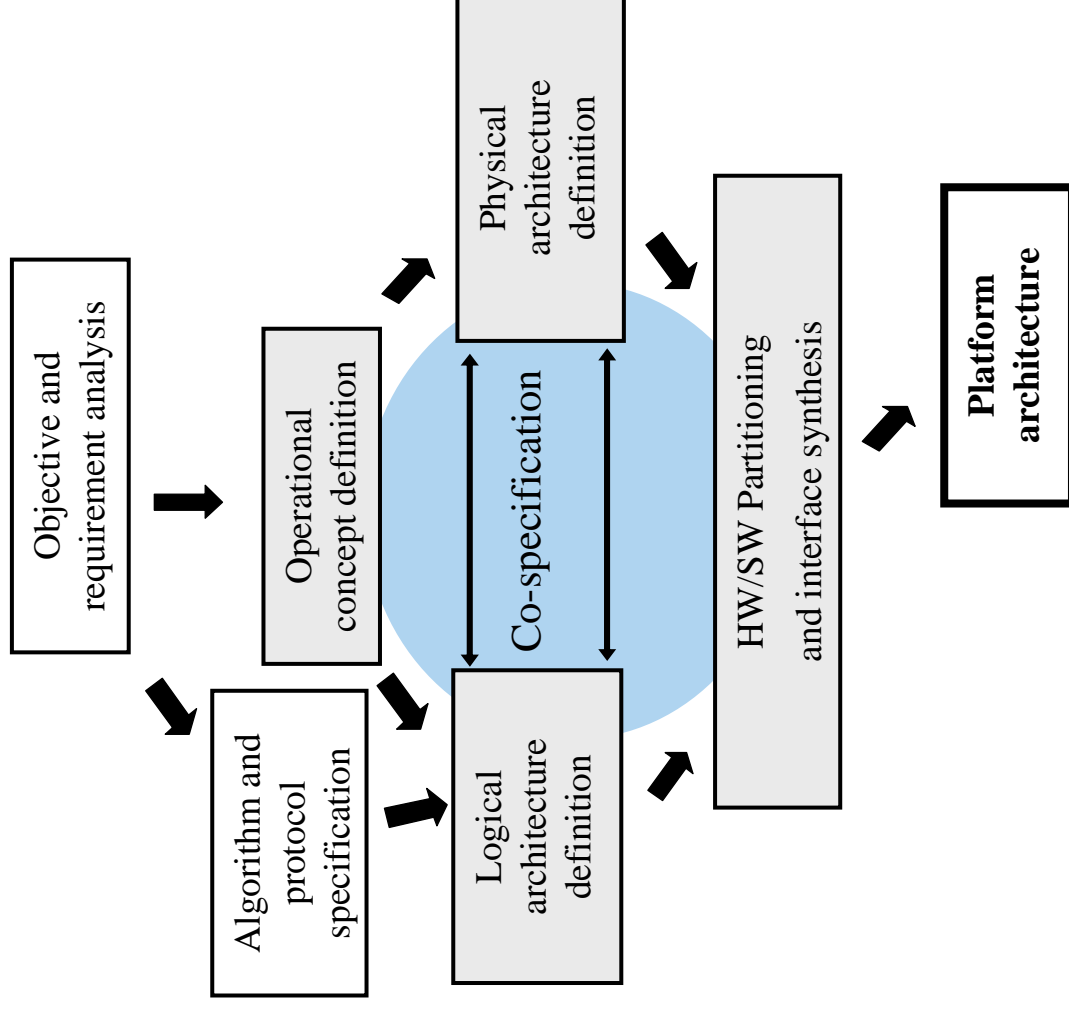
- Done in 2002-2005 at Nokia Research Center
- Tasks
  - To develop a new platform architecture for communication engines
  - To model it with SystemC
  - The primary goal was 3G but also multimode issues were taken into account
- The targets for the architecture development experiment were
  - Support for early SW development
  - Modular design style
  - Separate logical and physical architecture development
  - Early integration and verification
  - Coordinated management of design objectives and requirements
  - Maintenance of multiple abstraction levels for architecture models
  - Support for architectural exploration and performance evaluation

# The NeMo architecture



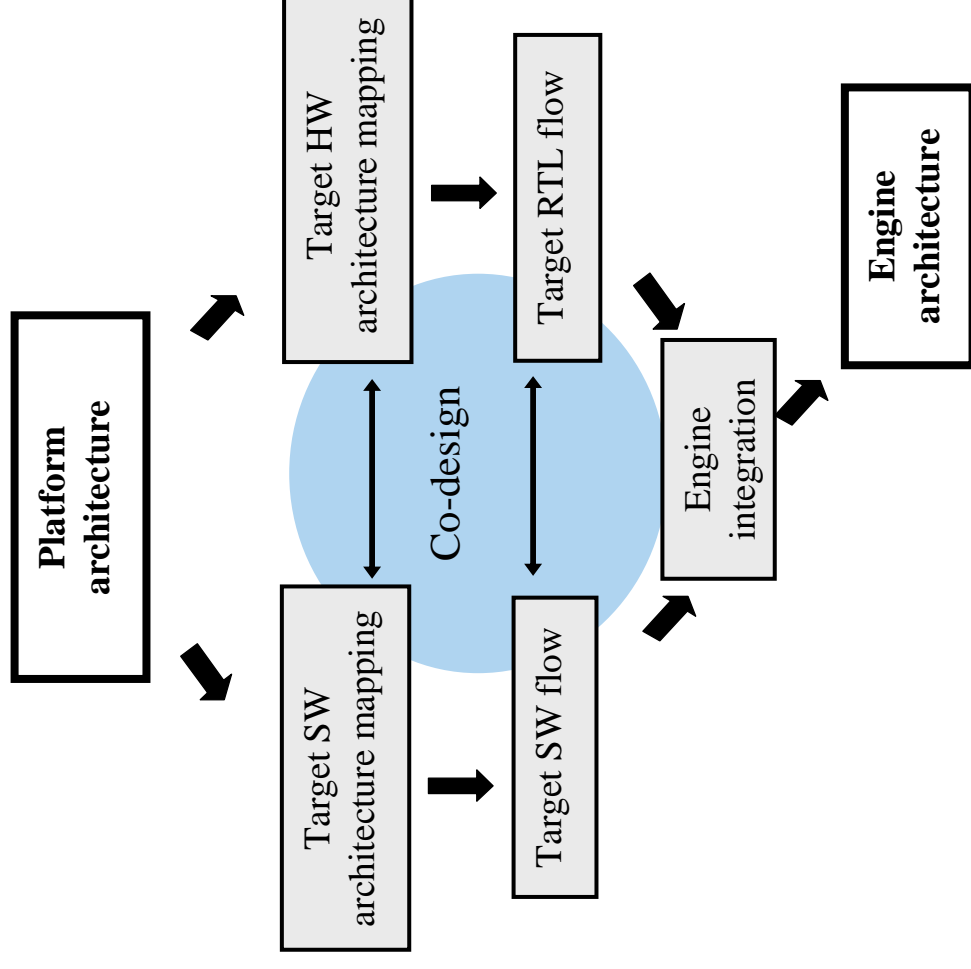
# The (NeMo) modeling process (1/2)

- Platform architecture is developed in the co-specification phase
- Logical and physical architecture development progresses in parallel
- The logical architecture defines the functionality and the functional clustering (here using pure C++)
- Physical architecture defines the physical resources (processing units, storage units, interconnects,...)
- The result of HW/SW partitioning of the logical architecture and the mapping of it to the physical architecture is the platform architecture
- SW parts remain as C++ and HW is described in (TLM) SystemC
- The main tasks here is architecture analysis



# The (NeMo) modeling process (2/2)

- The platform architecture is represented as a virtual platform
  - Executable specification to the HW development process
  - Environment for early SW development
  - Forms the basis for the co-design phase
- The main requirement to the virtual platform: functional accuracy from the SW point of view
  - Register interface
  - Internal (algorithm) behavior/functionality
  - Timing accuracy based on delay estimates, rather than clock-cycle accuracy
  - Short run-times of sufficiently extensive simulations preferred over accurate modeling of time



## Experiences: The positive first

- SW development can start early. This enables early feedback to HW designers before they have reached the prototyping phase. This makes it possible to do faster design iterations.
- The model acts as an executable HW specification and test-bench and enables early, although rough, performance estimates. This minimizes the number of required iterations.
- It makes derivative product creation very fast. In these cases the models are inherited from previous platform design projects and only require a small amount of re-design.
- The model enables fast simulation times compared to RTL simulations or even emulation using dedicated HW.

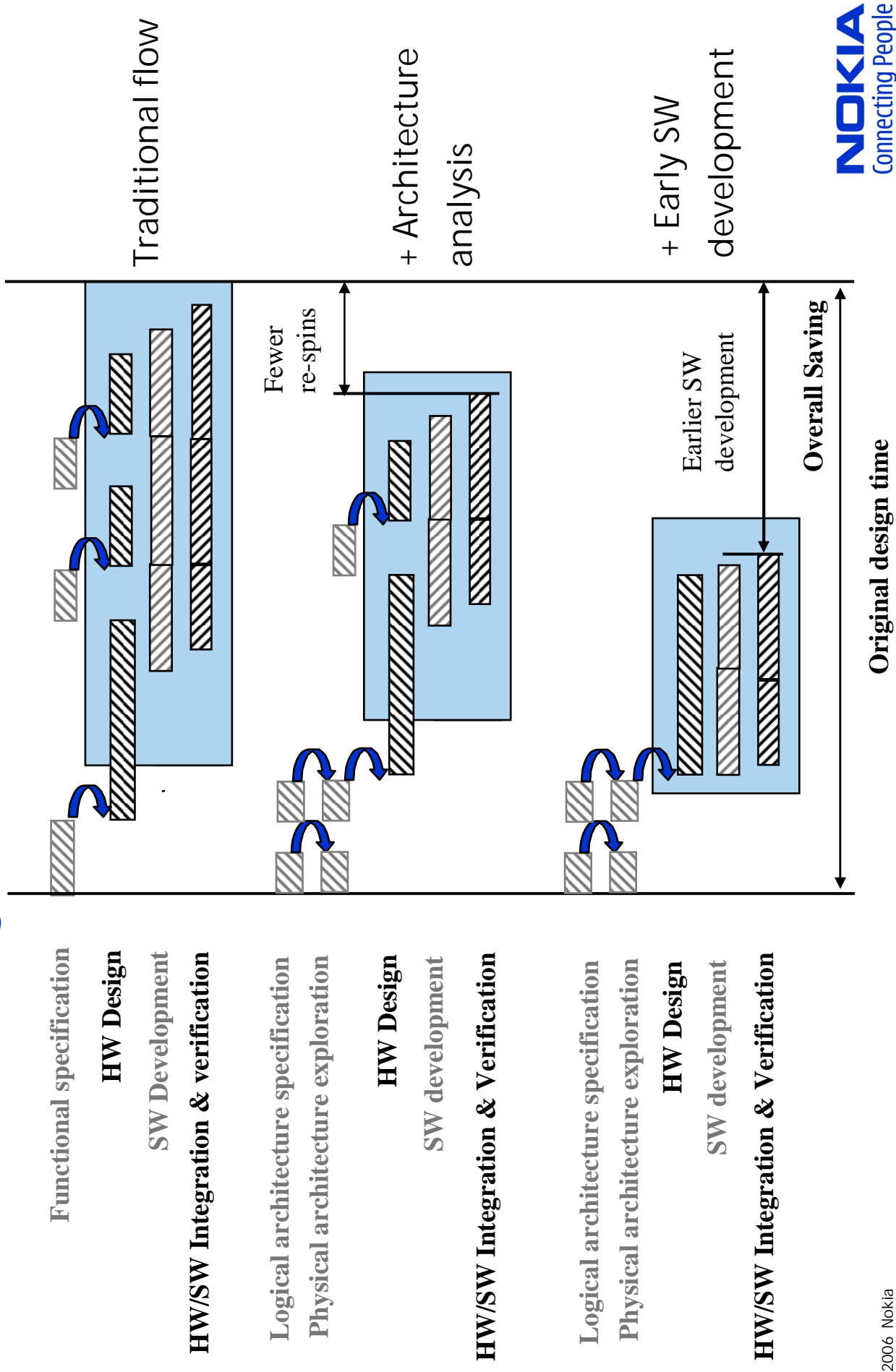
## Experiences: Little negative also

- Design effort is too high for analyzing radical changes in the architecture. The initial work to build the required models takes a long time and modifications are not always straight-forward.
- Maintaining and writing the models causes extra work to the traditional design flows. In order to make the model creation and simulation time fast they have to be made separately from HW design. The responsibility for the model and implementation equivalence was also seen problematic.
- The biggest obstacle was the addition of new phases to the traditional design flow which means new tools, new methodologies, new licenses, and so on. A new mindset and a lot of training are required for a successful adoption of these new design techniques and their integration into the current design flows.

# Practical issues with SystemC modeling

- Tight control over the used modeling style is required
  - Need to define a common modeling style and model interface
  - Adapters cause extra work, problems and slows down the simulations
- The standardization of interfaces is a key requirement
  - Preferred interface standard and the features that can and should be used
- Equivalence checking between the model and the actual implementation needs to be developed further
- High-level synthesis is seen as a needed feature (by the HW-oriented people)
- Tools are still somewhat immature, although getting better all the time
- Tool licensing can be expensive, since the goal is to use these tools for SW development with potentially a large number of users

# Benefits in time saving



# Conclusions of the experiences

- The problem of traditional design flows that rely on RTL for co-simulation and emulation is that too much time is spent in waiting for the RTL to finish and iterations are too costly
- One solution is to use transaction level modeling and SystemC
  - Separation of logical and physical architecture enables an efficient architecture specification process
  - This specification can be used as a platform for SW development and an executable specification and test-bench for HW design
  - Because the SW development and HW/SW integration and verification can start much earlier, significant amount of time is saved although the initial architecture design requires extra work
- To introduce TLM into a design process, a standardized and well-documented modeling style is needed
  - Coding structures, abstraction levels, timing models, interfaces,...

## Current work

- SystemC modeling conventions
- Production use pilot programs
- Tool evaluations
- Co-operation with EDA, system, IP, and IC houses
- Methodology and flow (further) development
- Standardization

# Why to standardize?

# Standardization in general

- Leave to others?
  - Slow and expensive (meetings with multiple attendees, traveling, annual fees, etc.)
  - Requires resources (dedicated standardization people, feedback from users, test / development of the deliverables, etc.)
  - Results not always usable for all participants (“democratic” way to deciding the goals, some input still required from all)
  - Standardizing too early leads to living standards that nobody wants to use
- Take part and contribute?
  - Communication with other leading players (in the ESL case: EDA, IC, system, and IP vendors)
  - Build a community and market around the standardized area
  - Way to promote and emphasize wanted areas (otherwise other players will make the decisions)
  - Development costs shared (fair play inside the community)
  - Interoperable solutions (vs. single-source)
  - Open, royalty free, transparent solutions (vs. single source)
  - Reliable, high quality results (At least it should be so...)

# ESL and standardization

- ESL still requires a lot of communication and specifically a system house view
  - Immature and initially proprietary technologies being developed by multiple companies
  - On-going standardization effort in several bodies (coherency an issue)
  - Many fields and ideas, what should be developed first?
  - EDA houses have so far dominated ESL development but require adopter feedback
- ESL standardization work in these independent not-for-profit organizations
  - Open SystemC Initiative (OSCI)
  - Open Core Protocol-International Partnership (OCP-IP)
  - Structure for Packaging, Integrating and Re-using IP within Tool-flows (SPIRIT)
- Other needed ESL standardization forums
  - Verification
    - Now in all the three abovementioned consortiums as a sidetrack
  - Processor and memory models
    - Non-standard tool dependent solutions still dominate

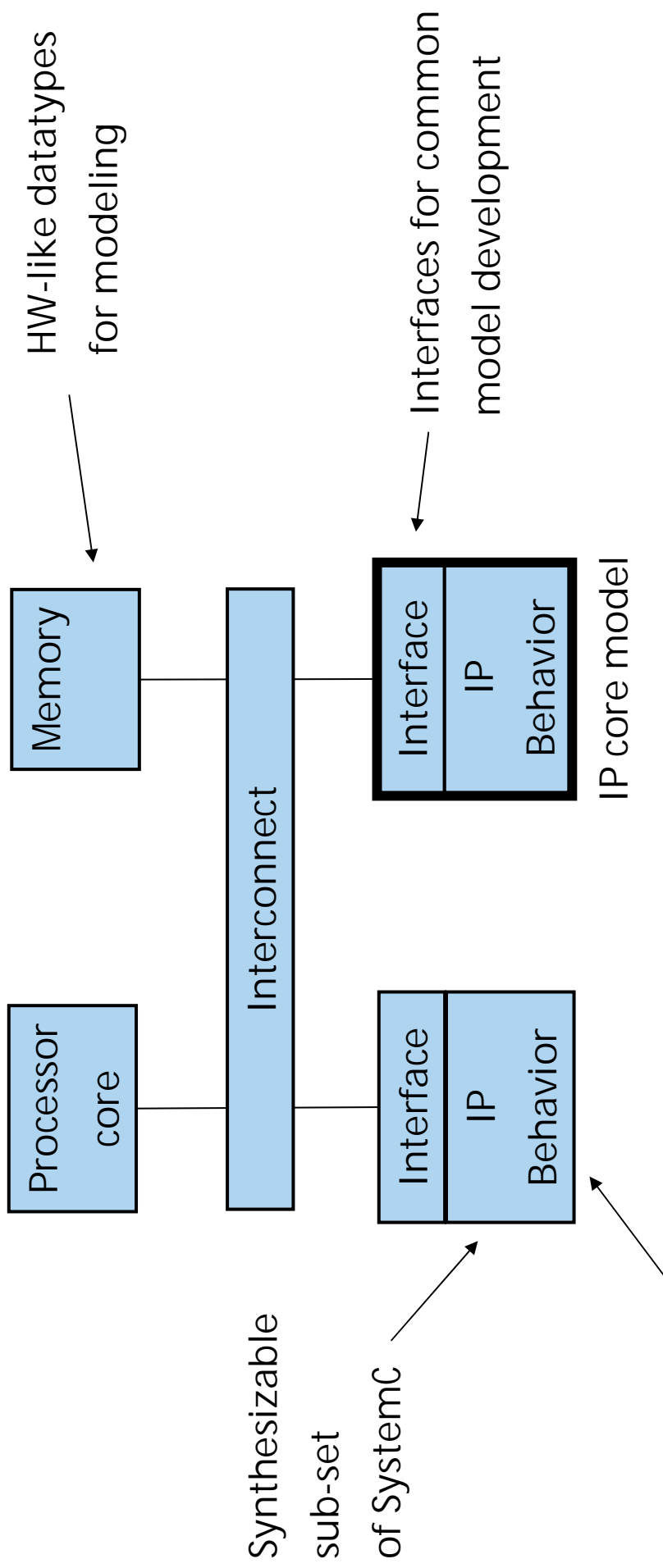
Where to standardize?

## OSCI ([www.systemc.org](http://www.systemc.org))

- Dedicated to supporting and advancing SystemC as an open source standard for system-level design
  - SystemC provides hardware-oriented constructs as a class library in standard C++
- Key companies: Philips, ARM, STMicroelectronics, Synopsys, Cadence, Mentor Graphics, CoWare, Summit Design, Forte Design Systems
- Working groups of OSCI
  - Language Working Group
  - Transaction-level Modeling Working Group
  - Verification Working Group
  - Synthesis Working Group
  - Analog/Mixed-signal Working Group
- Develops the most adopted ESL language

# Examples of what OSCI defines

Library supporting verification



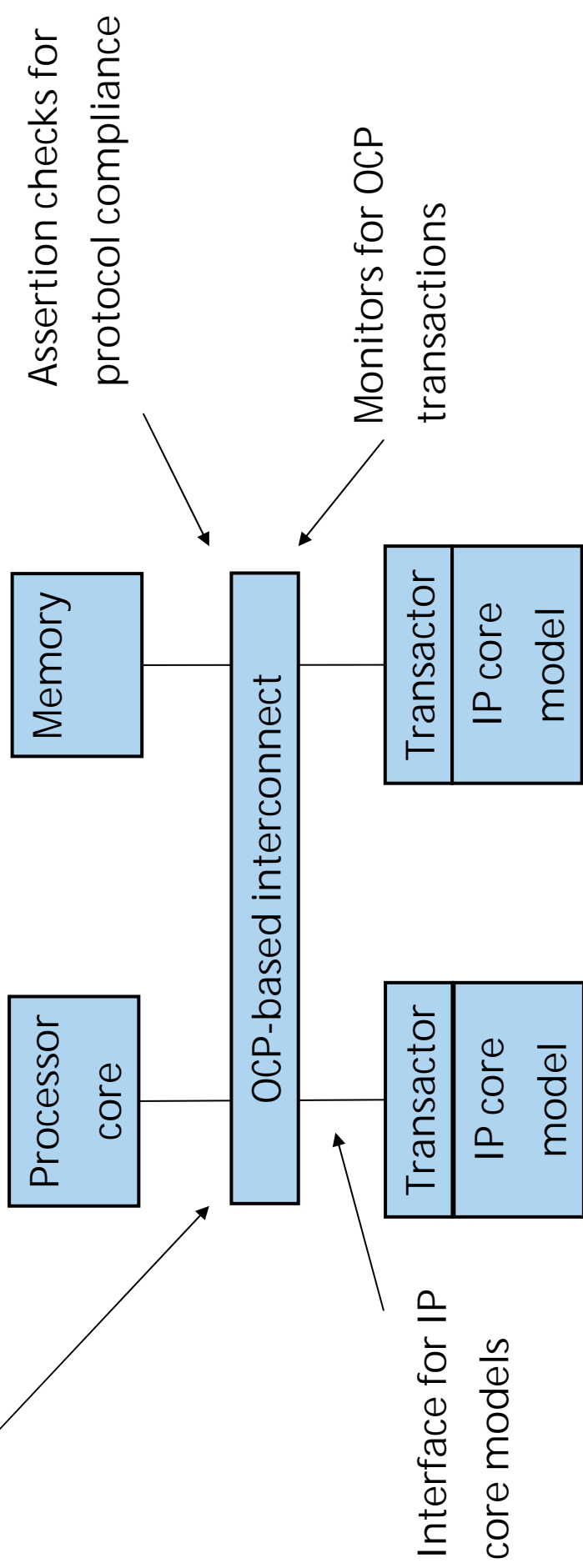
HW-required constructs to C++ (e.g. support for concurrency)

## OCP-IP ([www.ocpip.org](http://www.ocpip.org))

- Drives a common standard for intellectual property (IP) core interfaces enabling rapid creation and integration of interoperable cores that facilitate "plug and play" System-on-Chip (SoC) design
- Key companies: Nokia, TI, Toshiba, Sonics
- Working groups of OCP-IP
  - Specification Working Group
  - System Level Design Working Group
  - Debug Working Group
  - Functional Verification Working Group
- Defines the interfaces (not the bus!) of HW models

# Examples of what OCP-IP defines

OCP SystemC channel models  
on several abstraction levels



# SPIRIT ([www.spiritconsortium.org](http://www.spiritconsortium.org))

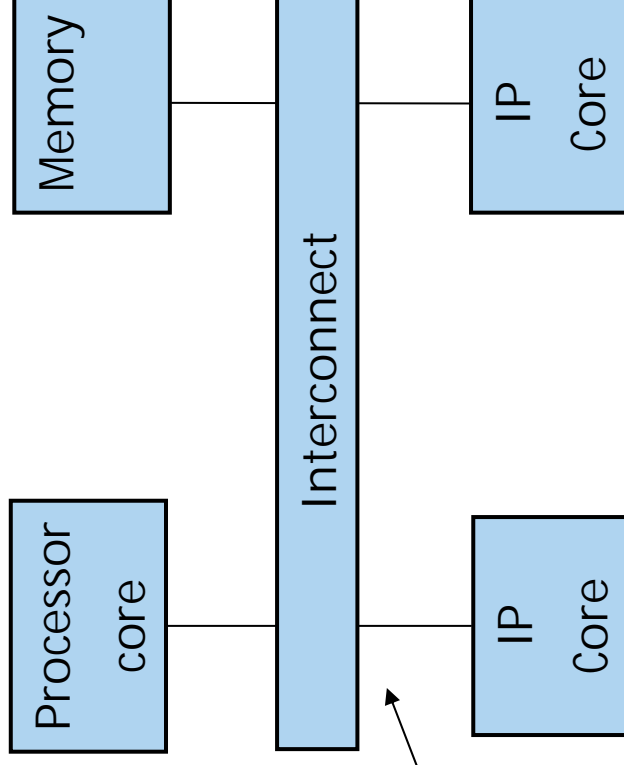
- Drives a standard mechanism for describing and handling multi-sourced IP that enables automated design integration and configuration within multi-vendor tool flows
  - XML schema for describing IP blocks and platforms
  - Generator interface for configuration
- Key companies: Philips, STMicroelectronics , LSI Logic, ARM, Synopsys, Cadence, Mentor Graphics
- Working groups of SPIRIT
  - Schema Working Group
  - ESL Working Group
  - Verification Working Group
  - Debug Working Group
- Defines IP packaging and platform description methods for ESL tools

# Examples of what Spirit defines

Metadata information

in a standard

XML format



Standard format  
for storing net list

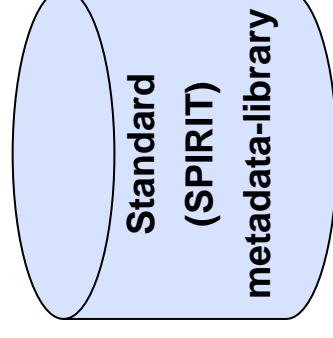
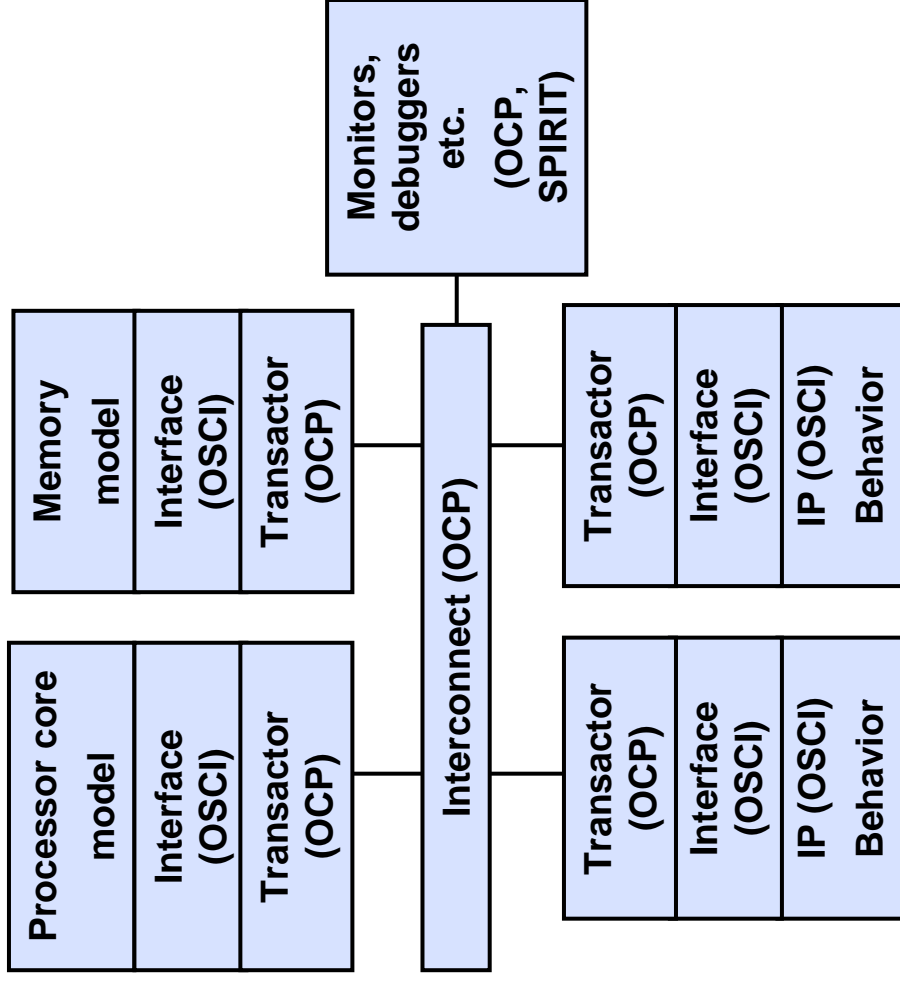
Description of  
standard  
interfaces

Tool-independent  
configuration interface  
(for data width etc.  
parameter definition)

Seamless passing of information (e.g.  
requirements) between tools

# The goal

- Standardized modeling methodology
- Consistent model base of all components for all purposes with a very limited set of modeling style options
- Standardized interfaces, tool independency
- Standard meta-data format for IP interfaces, requirements, constraints etc.
- Processor and memory model standardization (Lisa?, Denali?)
- Verification standardization (Accelera?)



# Conclusions

## What have we learned?

- ESL tools and methods offer great opportunities
- Initially the adoption requires massive effort in the form of model base build up and learning the new approaches + tools
- The needed resources cannot be justified, if a different model is required for each use case and tool set
- ESL adoption requires a common, standardized, open, and royalty free methodology for modeling
  - IP providers can make the models
  - EDA vendors can provide the tools (a lot of opportunities for point tool development)
  - Standard interfaces and meta-data formats are in a key role
- After that system houses can and will deploy the tools into production use

Thank you! Questions?