# HW-SW Interfaces Design for Multiprocessor SoC


## Dr. Ahmed Amine JERRAYA

## TIMA Laboratory

**46 Avenue Felix Viallet**

**38031 Grenoble Cedex France**

**Tel: +33 476 57 47 59**

**Fax: +33 476 47 38 14**

**Email: Ahmed.Jerraya@imag.fr**

# HW-SW Interfaces for MPSoC: Summary

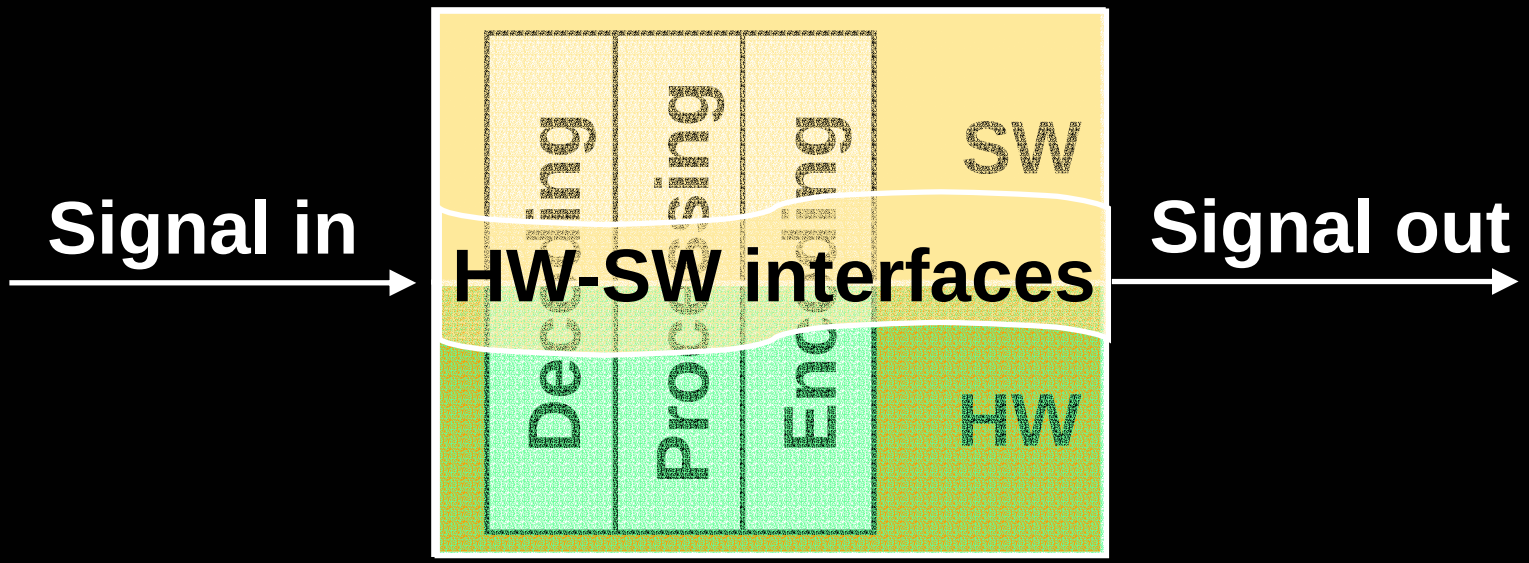
1. **SoCs are made of heterogeneous components:**
   **Heterogeneous Interconnect is the enabler for Higher than RTL design**
2. **Coordination between hardware and software is hard to master**
   **Difficult to calibrate, Diversity, Complexity**
3. **MPSoC requires sophisticated application specific HW-SW Interfaces: Multithreading, Interrupts, application specific communication**
4. **HW-SW interfaces design is the non rewarding part of the design flow: Designing yet another Driver or Bridge**
5. **HW-SW interfaces design may be automated:**
   **abtract HW and SW interfaces model is the key issue**
6. **Abstracting HW-SW Interfaces makes easier all the design steps: Architecture exploration, SW design, HW design, HW-SW component Integration, SoC Debug, SoC Validation.**

# Outline

1. The challenges of HW-SW interfaces
2. HW-SW Interfaces for MPSoC
3. HW-SW abstraction for MPSoC: the concept of virtual architecture
4. ROSES: Automatic generation of HW-SW interfaces for MPSoC
5. HW-SW Interfaces in the design flow
6. Summary

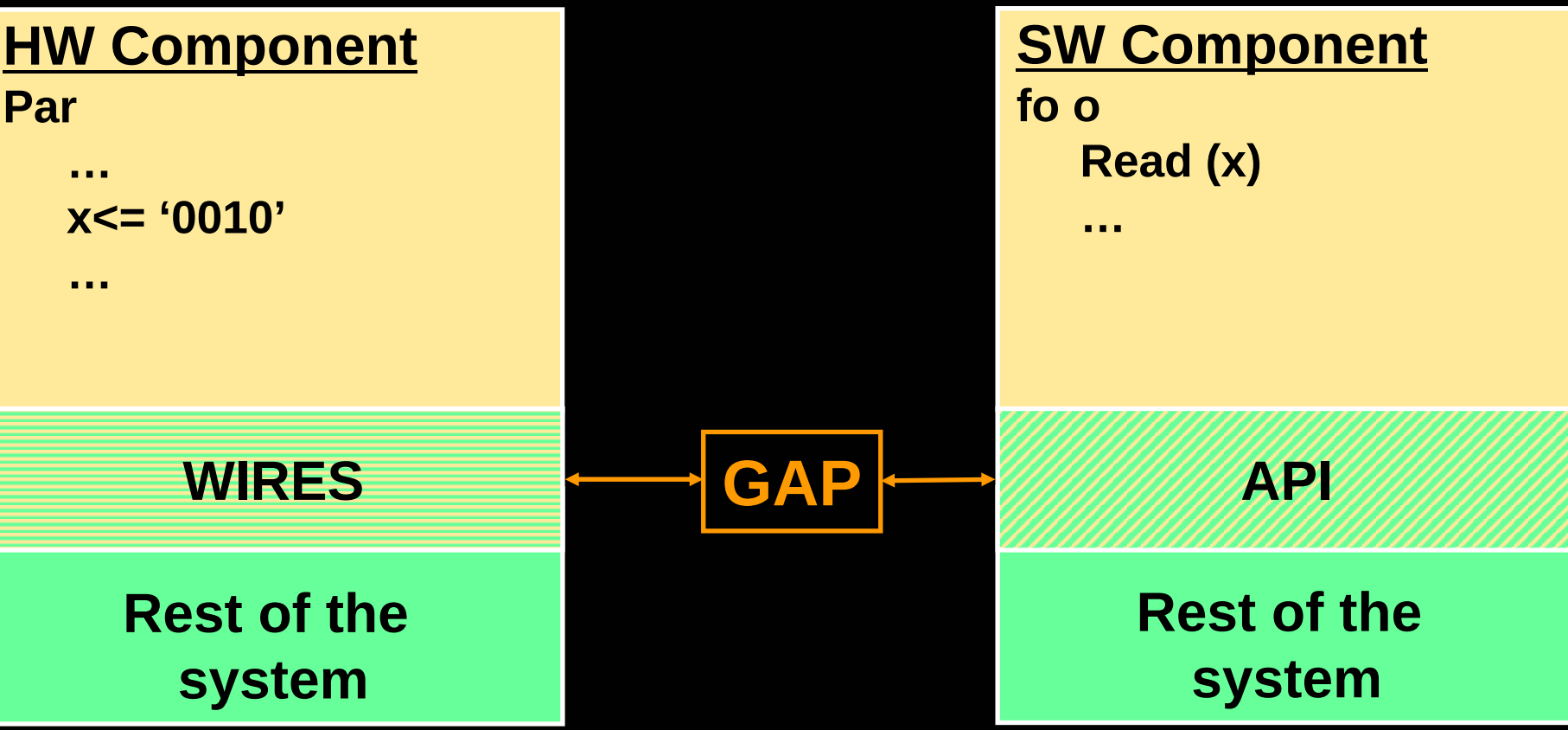
# Mixed HW-SW Systems

- Mixed implementation of integrated system
  - A function implemented partially in HW and partially in SW.
- HW-SW interfaces abstract interaction between the HW and SW parts.

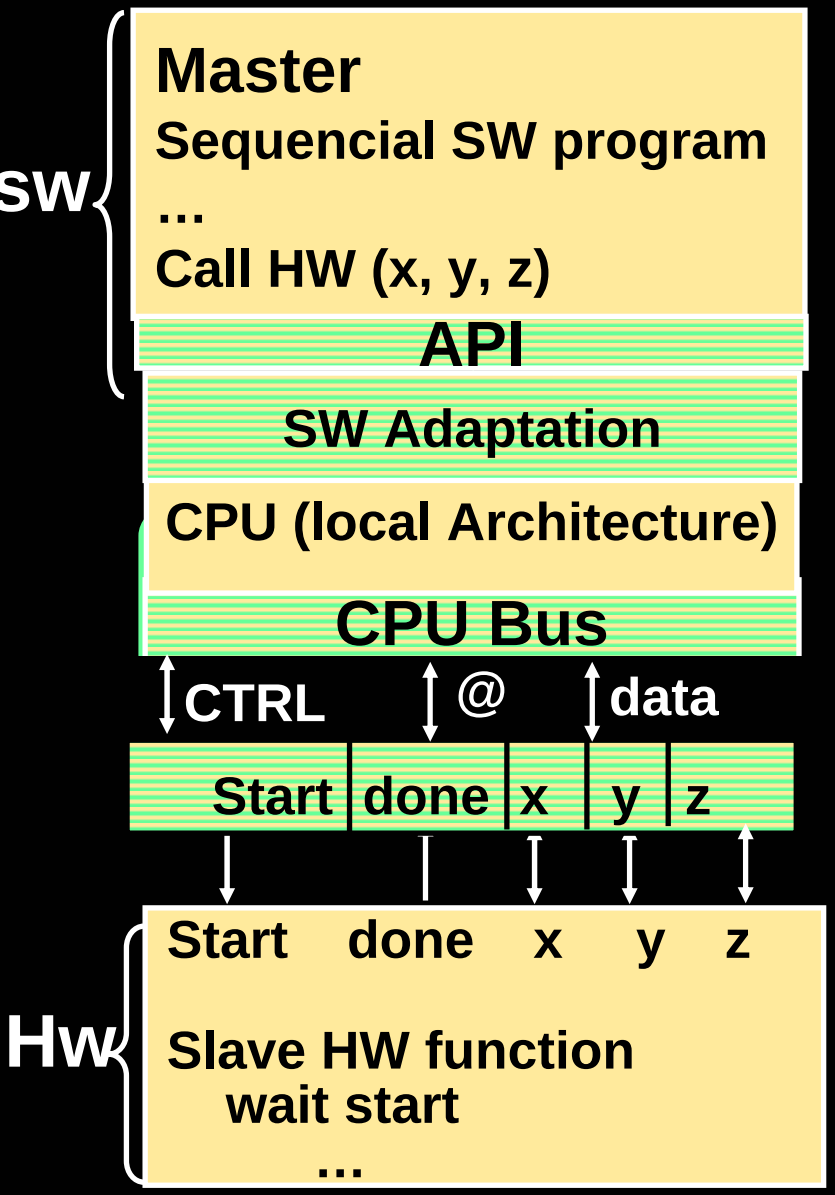


# HW and SW Make Use of Different Concepts to Abstract Interfaces

- SW communicates with the rest of the system through API.
- HW communicates with the rest of the system through wires.



# HW-SW Interfaces and Coordination




- Master/slave single thread systems are usually easy to model and to design
- API: Synchronous SW procedure call
- SW Adaptation
  - Call parameters match HW data ports
  - IO Driver (Busy wait for results)
- HW Adaptation
  - R/W to fixed addresses
  - HS protocol to Sync Communication
- Multi-task multiprocessor systems may require complex coordination

# HW and SW Interfaces Issues for MPSOC

For multi-threaded software, the SW layer implementing the API is very complex.

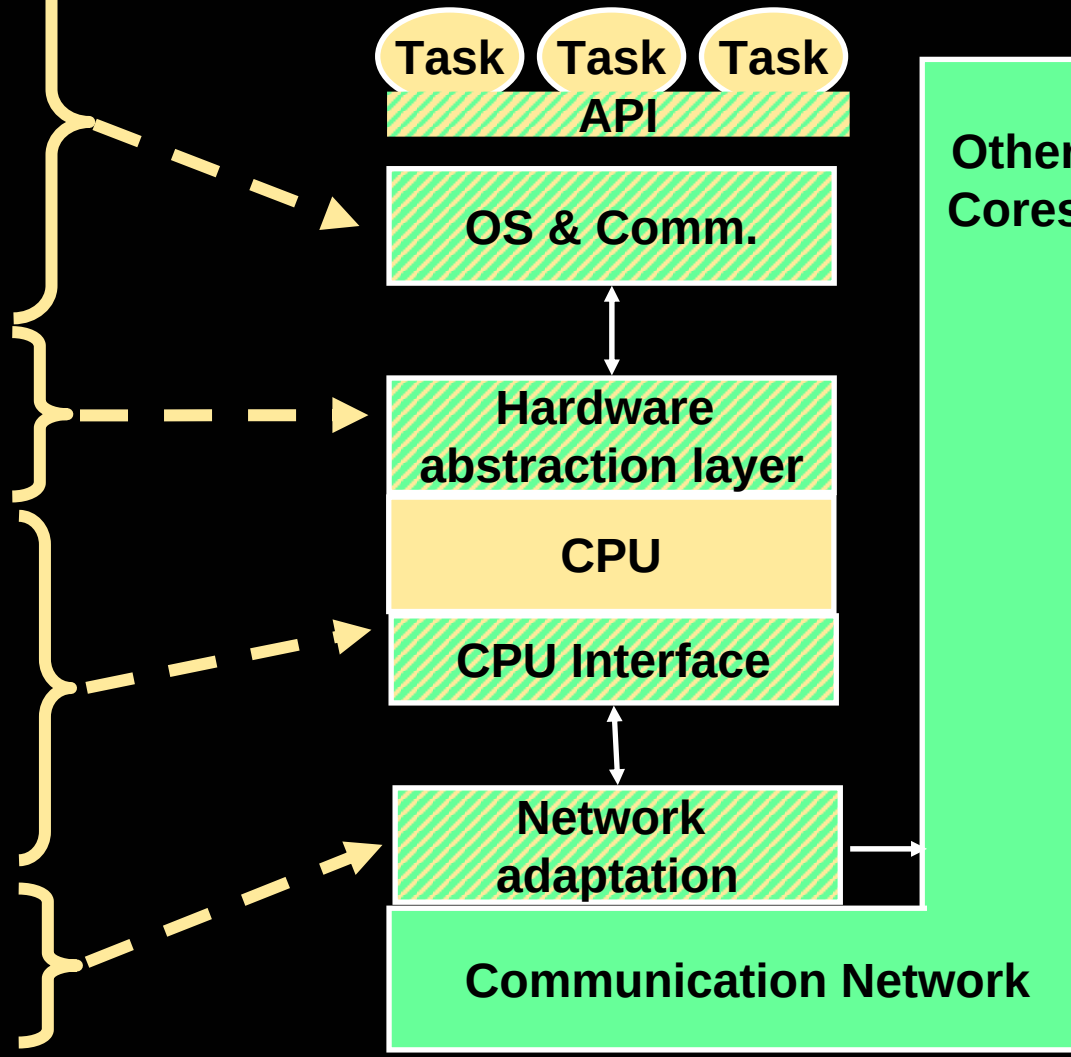
- Abstract Communication
- Concurrency management
- Fast reaction to Events
- Sophisticated I/O

Adapting to different OS or CPU may require an additional HW abstraction layer.

Each CPU may need to communicate with more than one device.

- Bus conflicts management
- Data conversion
- Buffering

Heterogeneous multiprocessor may require complex communication network.

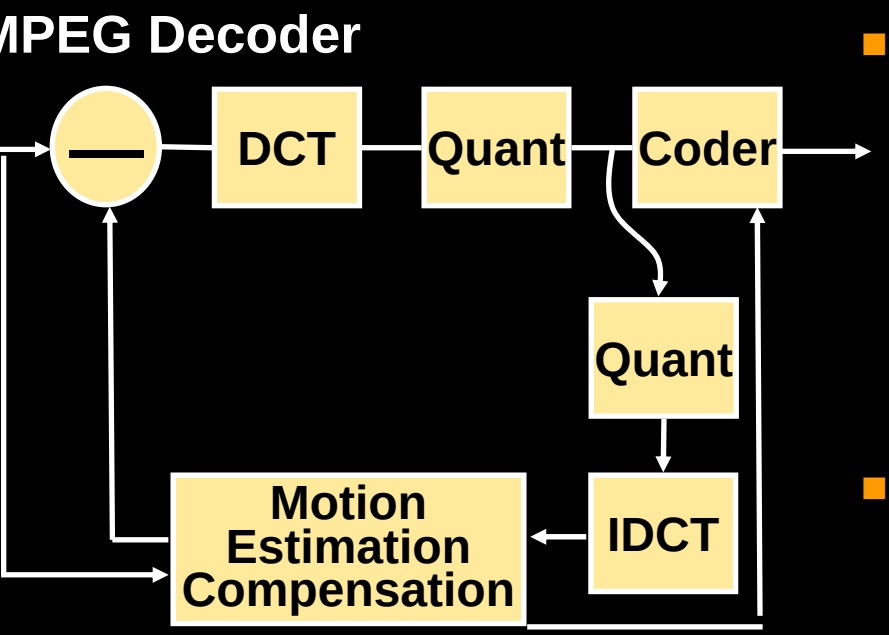


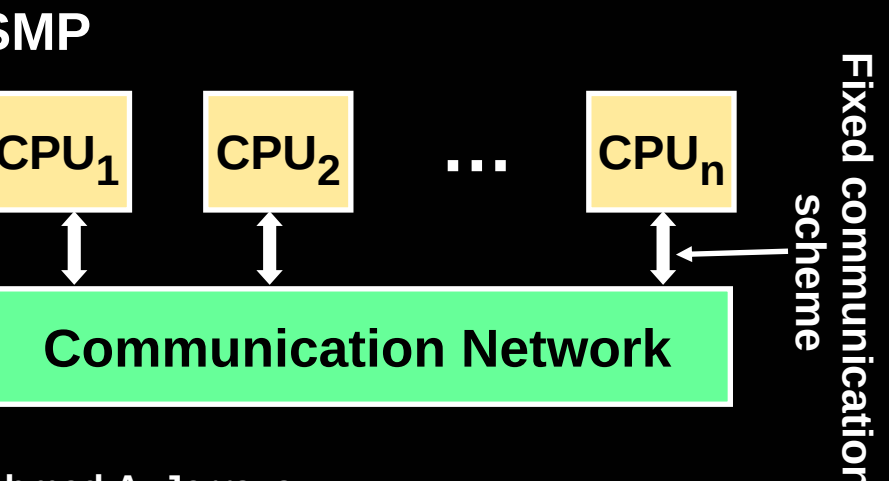
# Problems to be Solved by SW and HW Adaptation Layer

- SW Adaptation usually provided by OS & HAL
  - Resources sharing, multi-task management
  - Real-time services
  - I/O = adapts to different I/O schemes
  - Synchronization = interrupts management
  - Task inter-dependence = avoids dead locks
- HW Adaptation Layers: links between different data, control & clock signals
  - Arbitration
  - Timers
  - I/O Control (protocol conversion)
  - Synchronisation, event Management
  - Ensure data coherency

**Coordination between hardware and software is hard to master: Difficult to calibrate, Diversity, Complexity**

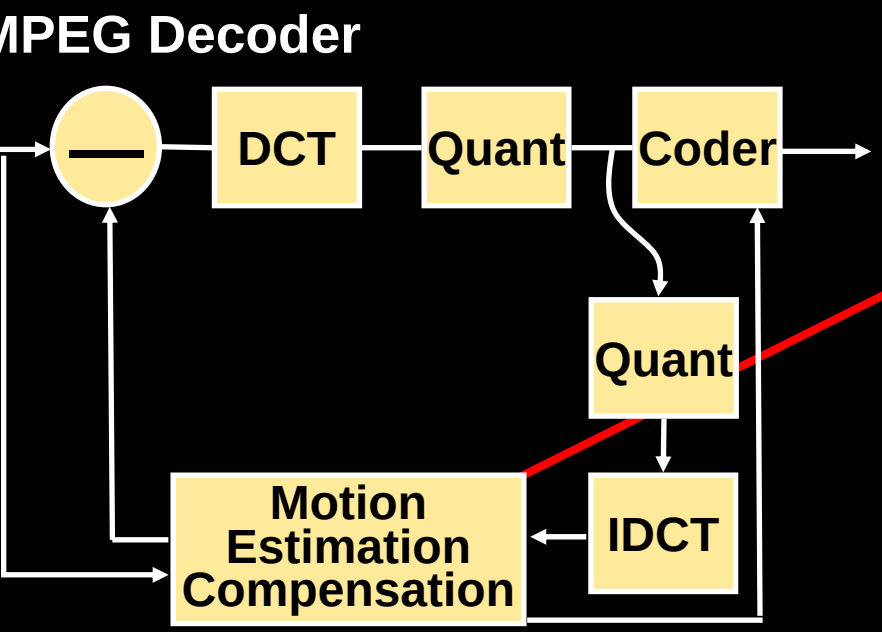
# Pure HW Design and Pure SW Design



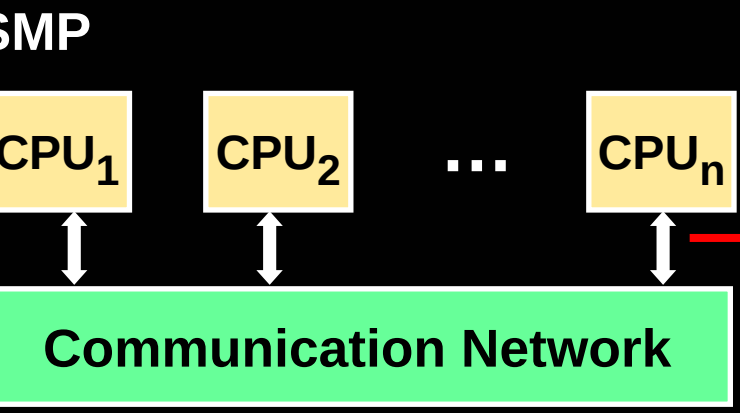




- Pure hardware design is much easier than mixed HW-SW design.
  - Example: MPEG decoder for a fixe standard and a fixed application.
  - In the absence of multiple use or evolution, the most complex functions are easier designed as pure hardware.
- Pure software design using SMP is easier for non constrained communication.
  - Example: MPEG decoder with no real-time constraints
  - If performances and/or hardware cost are not an issue, the most complex function is better designed as pure software eventually distributed for performances.

# Taking Flexibility and Performances into Account

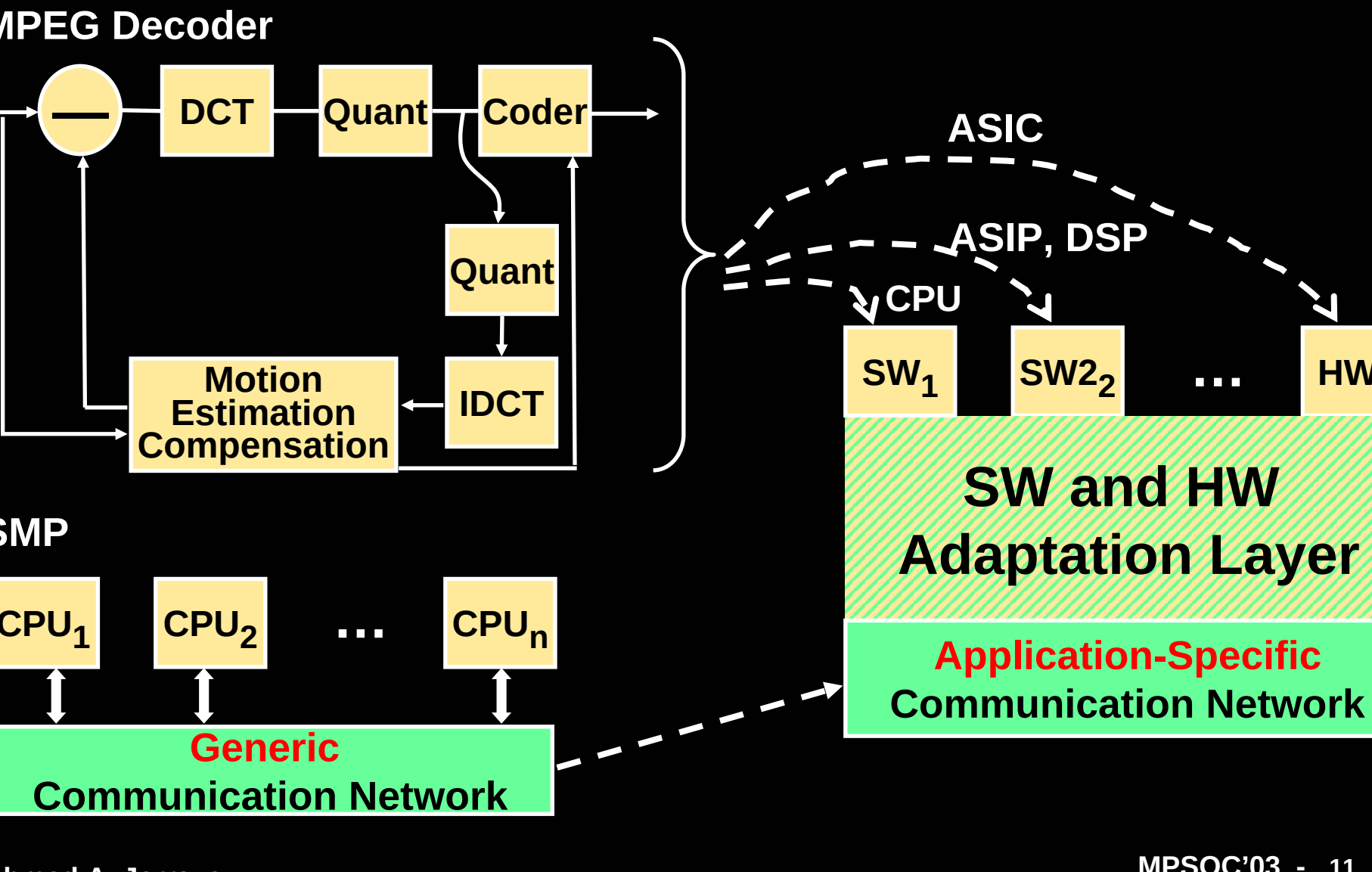



- Need to accommodate different coding or compensation schemes.
- Performances & cost
  - Need a specific processor to accommodate a given computation.
  - Need a specific communication scheme to accommodate application.



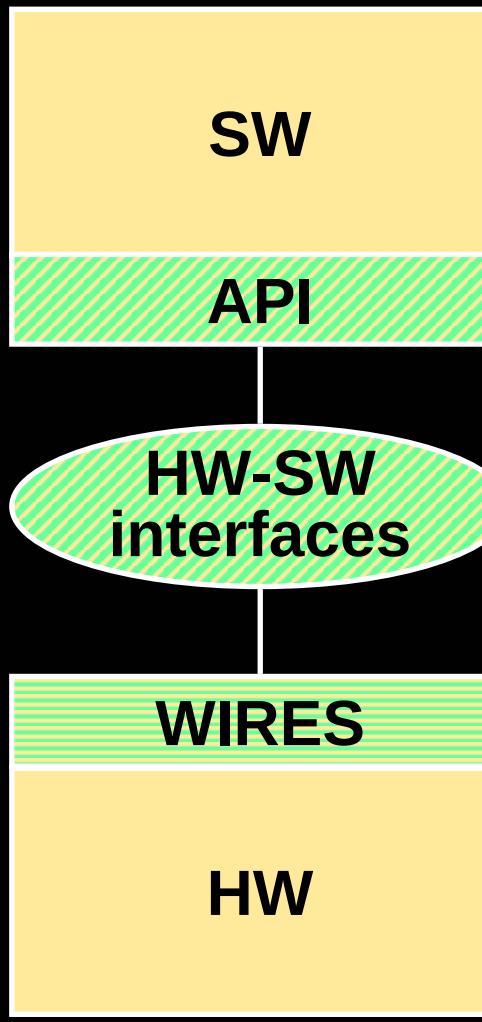

**HW-SW interfaces is a non avoidable aspect of SoC design**

# Application-Specific MP SoC with Heterogeneous Processors and Network



# HW-SW Interfaces is a Useful Concept for MPSOC

- Simplification
  - Design of different parts may be separated.
  - Separation between communication and computation ease component reuse.
- Modularity & flexibility
  - Within an architecture a component may be replaced, or implemented in different technology.
  - Simpler control & synchronization scheme.
- Allow to tune architecture performances to application
  - Using predefined communication structure may induce overhead.
  - We may take advantage of specific communication infrastructure (buffering, interrupts, …) to handle multi-tasking efficiency.
  - Even for single thread CPU, busy waiting may be avoided to save power.



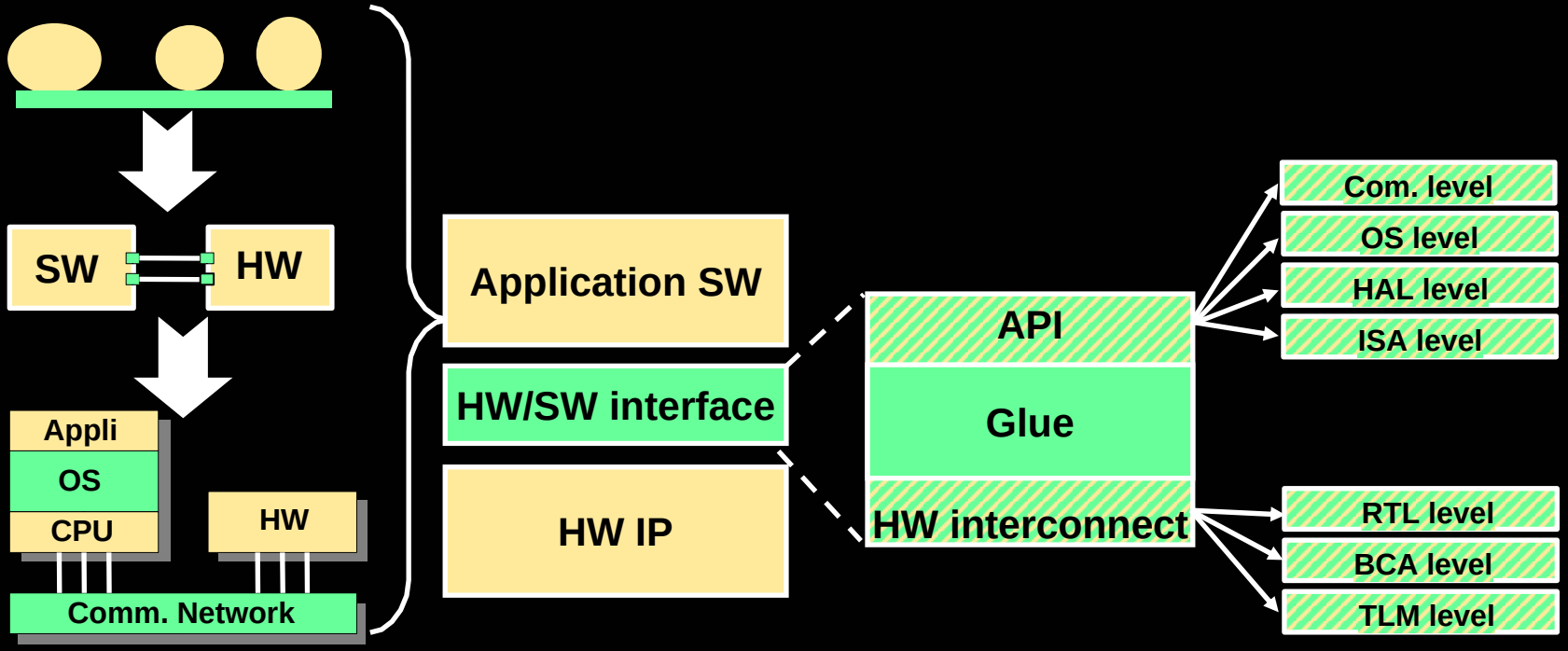
# Hardware-Software Interfaces Summary

- HW-SW interfaces are needed for SoC including CPU.
  - Non avoidable aspect of the design problem.
- Coordination between hardware and software is hard to master.
  - Difficult to calibrate
  - Diversity
  - Complexity
- Pure SW design is non effective for SoC.
- Pure hardware design is not flexible enough to ensure SoC ROI
- HW-SW interfaces is a useful concept for MPSOC design.

# Outline

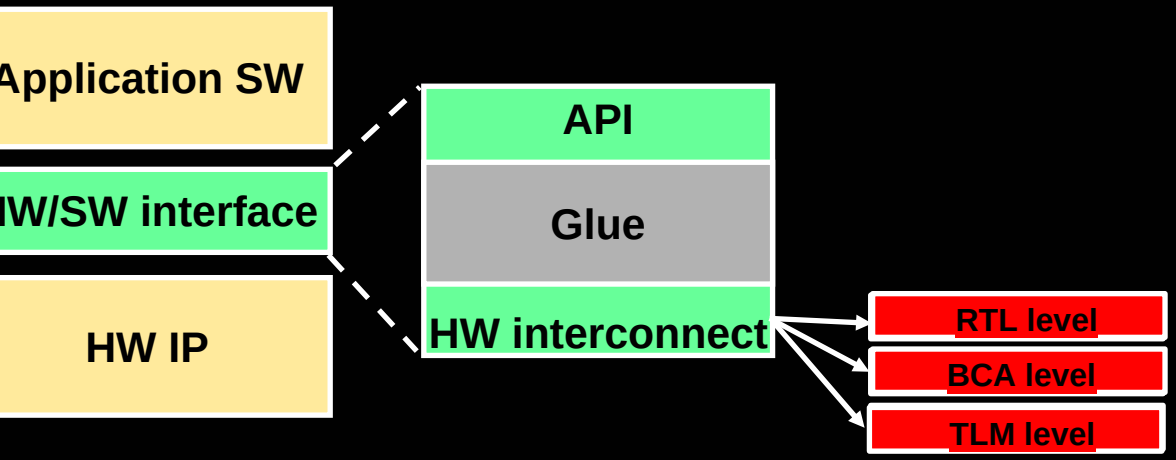
1. The challenges of HW-SW interfaces
2. HW-SW Interfaces for MPSoC
3. HW-SW abstraction for MPSoC: the concept of virtual architecture
4. ROSES: Automatic generation of HW-SW interfaces for MPSoC
5. HW-SW Interfaces in the design flow
6. Summary

# HW/SW Interface in the design Process




- Different Abstraction levels for both HW and SW
- Partitioning and communication Architecture may be abstracted
- Executable model makes global validation possible
- May require different implementation models for simulation
- Standards ease automation
- Multiple abstraction layers ease inter disciplines communication

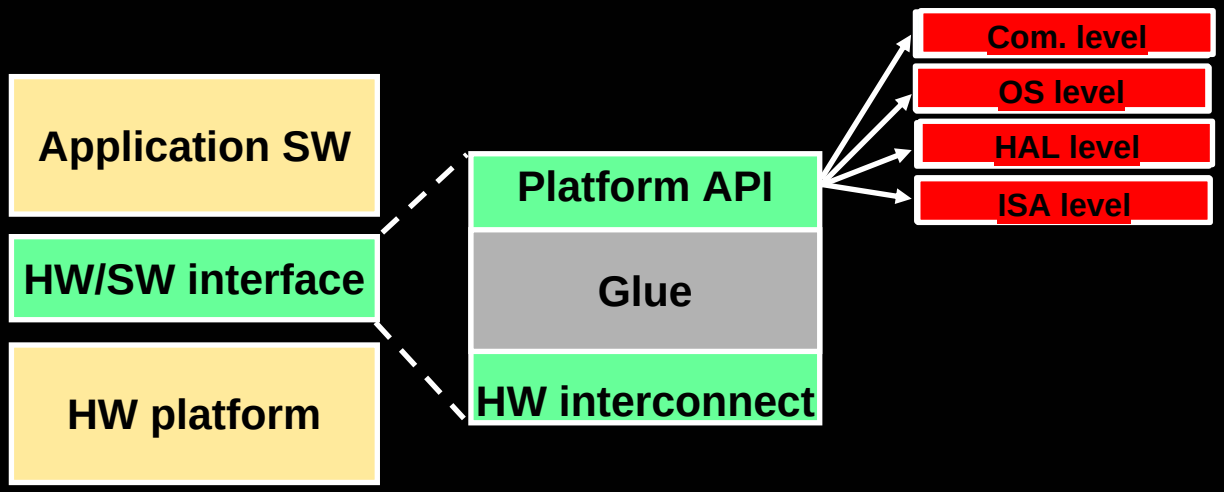
# HW Abstraction Levels





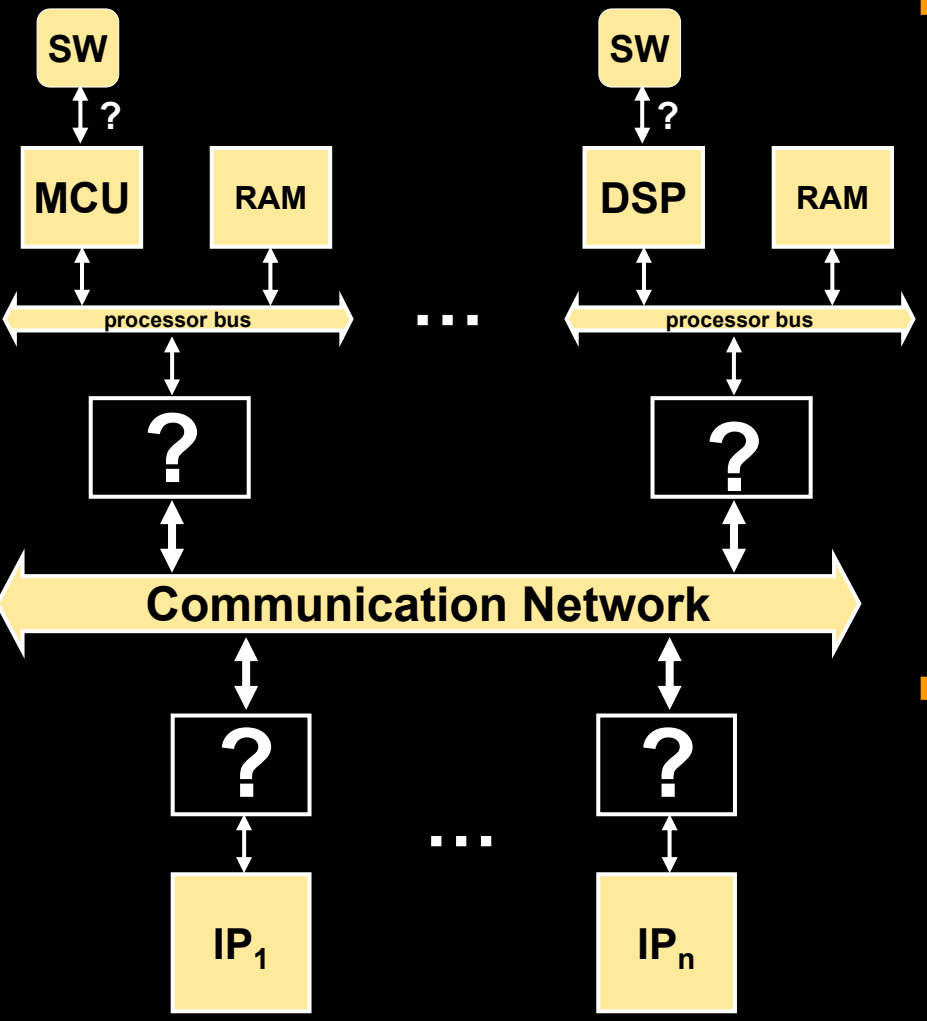
- Many standardisation initiatives to abstract wires
- Physical Hardware interface = Data+ Control + Ck
  - RTL: No abstraction
  - BCA(Bus Cycle accurate): Abstract Data structures
  - TLM (Transaction Level Model)
    - Bus Transaction: Abstract Clock
    - Message: Abstract Control

# Software Abstraction Levels



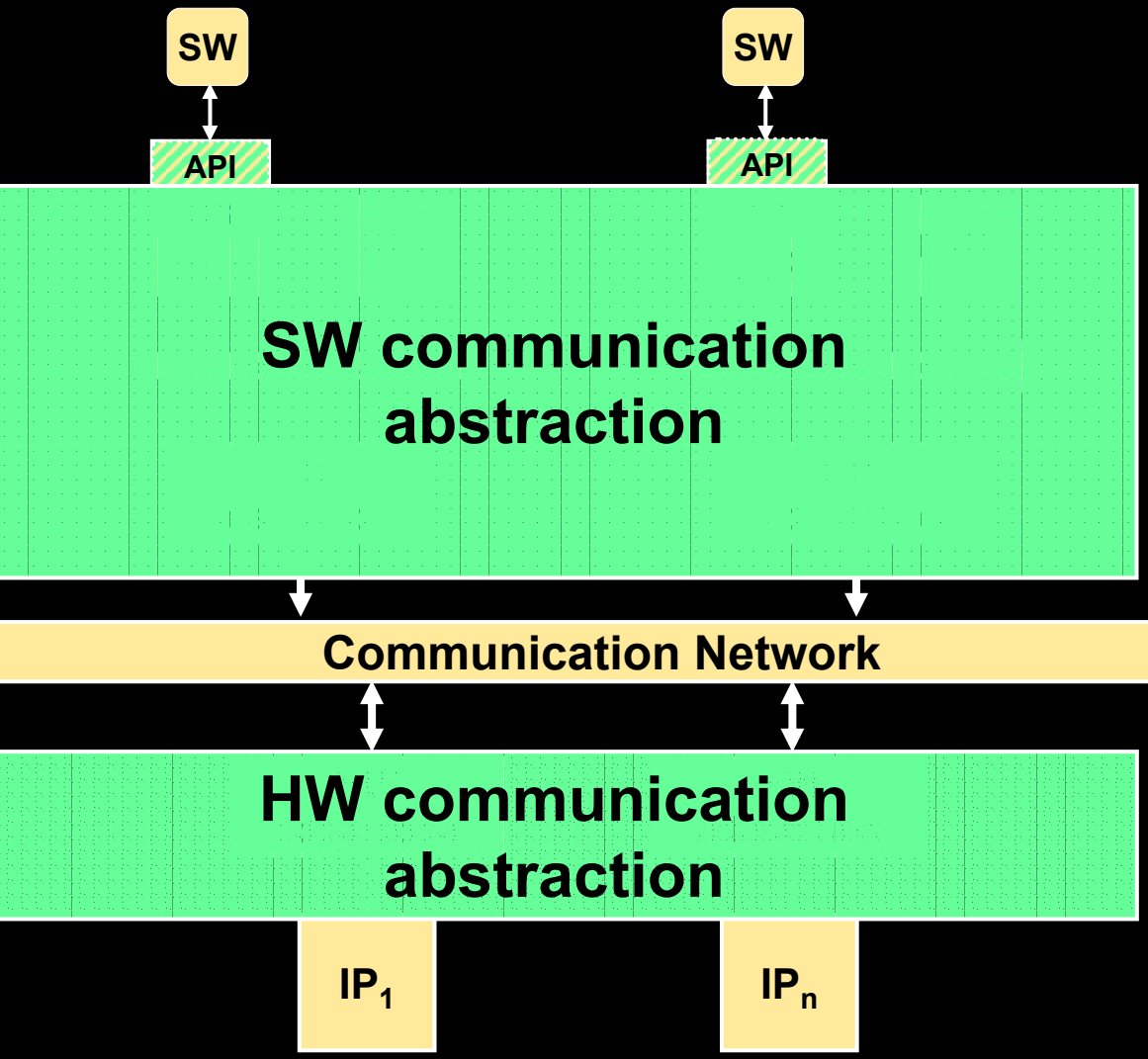

- Communication level: Set of tasks running on a not yet fixed communication architecture (e.g. MPI).
- OS level: Application SW = Set of tasks running on abstract operating system using OS API
- HAL level: OS is actually implemented using a HAL API that abstracts the underlying HW architecture.
- ISA level: All software code is fixed.

# HW/SW communication design Heterogeneous MPSoC




- My assumptions
  - Distributed SW executed on local architectures
  - Complex Local Architecture for SW sub-system ( CPU, local memory, Timer, PIC, DMA …)
  - On-chip communication network
  - HW IP
- HW-SW interfaces
  - Adapt SW Components to local architectures
  - Adapt SW sub-system to network
  - Adapt HW IP to network

# HW/SW communication Abstraction



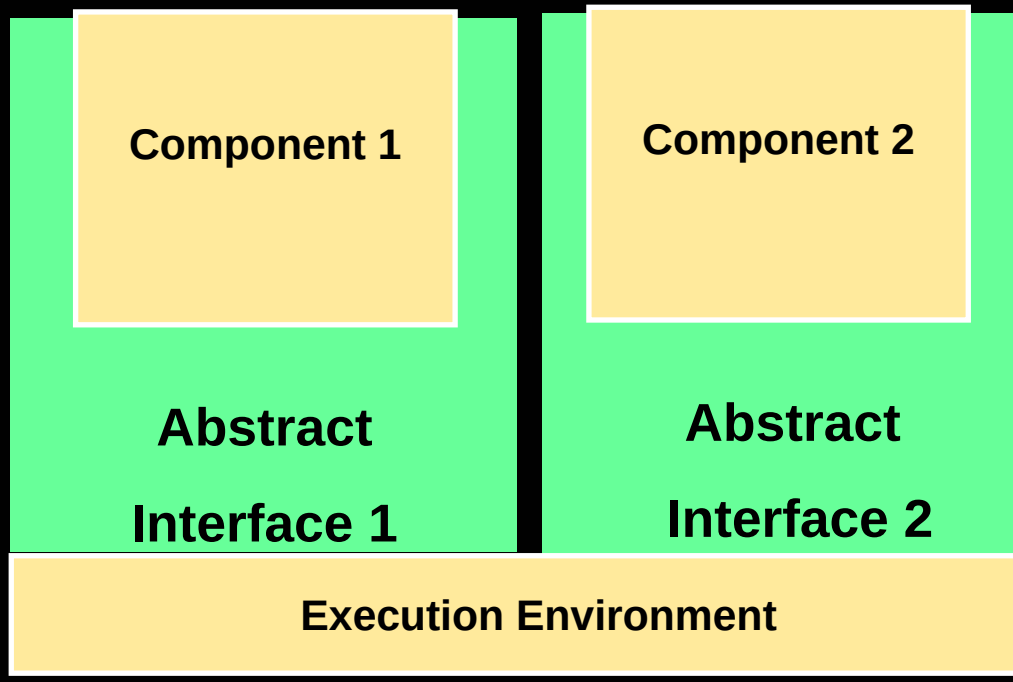

- SW communication abstraction
  - API hides rest of system for SW modules
  - HW-SW layers to adapt SW module to rest of systems
- HW communication abstraction
  - HL interface hides interconnect
  - HW-adaptation layer to link HW module to rest of system

# Outline

1. The challenges of HW-SW interfaces
2. HW-SW Interfaces for MPSoC
3. HW-SW abstraction for MPSoC: the concept of virtual architecture
4. ROSES: Automatic generation of HW-SW interfaces for MPSoC
5. HW-SW Interfaces in the design flow
6. Summary

# The Virtual Component Model

- **Virtual component**
  - Component
    - Hardware
    - Software
    - Functional
  - Abstract Interfaces
    - Required Services
    - Provided Services
    - Control Services
    - Synchronization
    - Parameters, ….



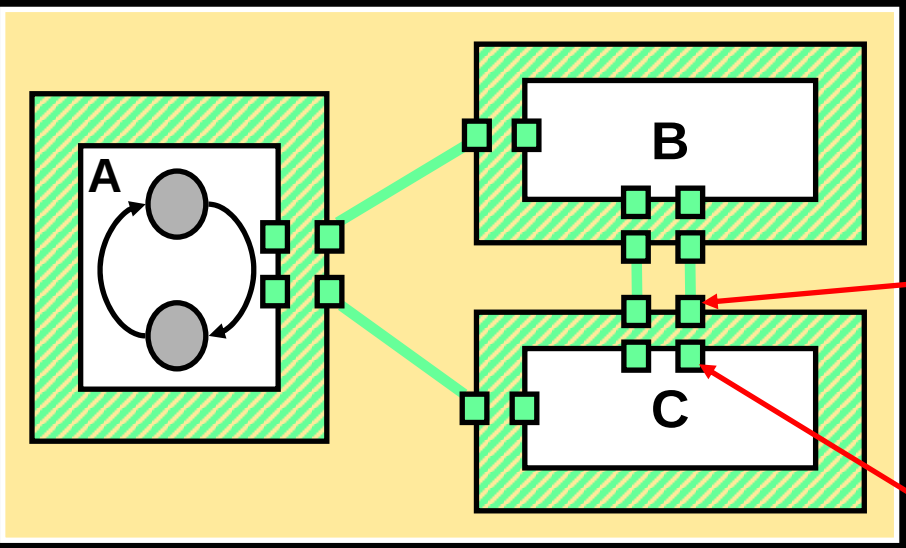

- **Execution Environment**
  - Abstract Platform (e.g. NoC, Cosimulation backplane, …)
- **Heterogeneous components thanks to adaptation**

# The Virtual Component Model

- A very popular SW Object models: CCM, Active objects, Containers, DCOM….
- Adopted by SoC communities: OCCN, StepNP, VCC, OCP, VCI, TLM, Coware, SystemC …
- Handles Heterogeneous Objects
- Hides details and allow delay decisions through the use of generic models
- Allows different and sophisticated adaptation schemes
- Allows automation for specific interfaces and/or target architectures
- Handles different abstraction levels

# Heterogeneous System Specification

- Basic model: a set of hierarchically interconnected modules
- Basic concepts:
  - Virtual Module
    - Interface, set of virtual ports (internal, external, SAP)
    - Content (Tasks / Instances + Communication channels)




System Specification

External port

Internal port

Abs. level

TLM

RT level

Protoc

FIFO

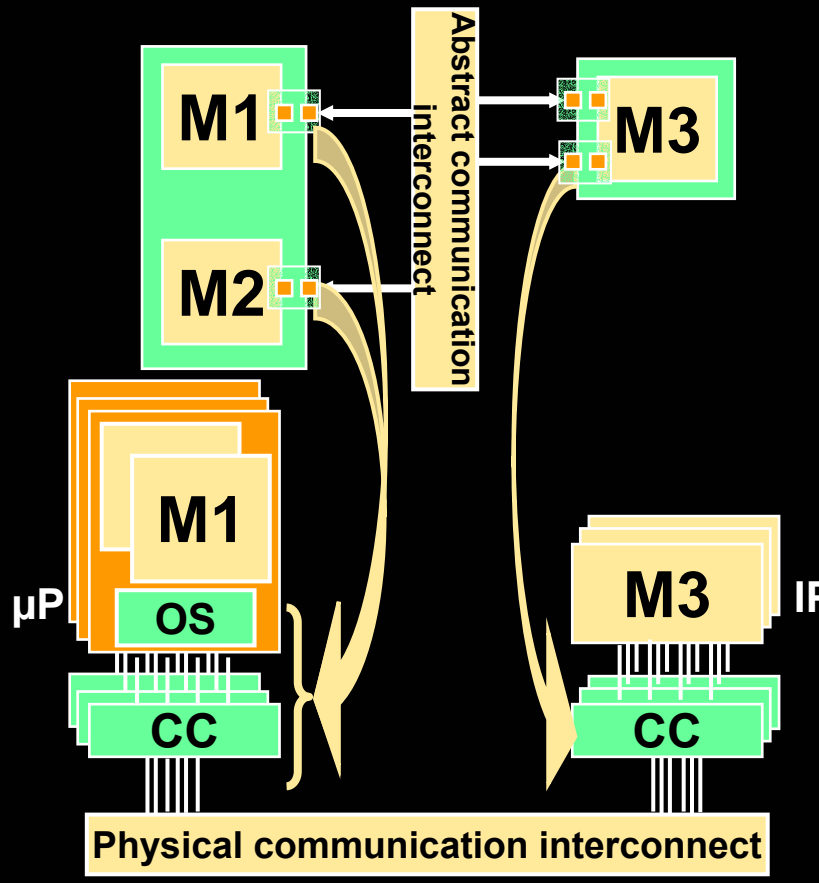
AMBA

# Outline

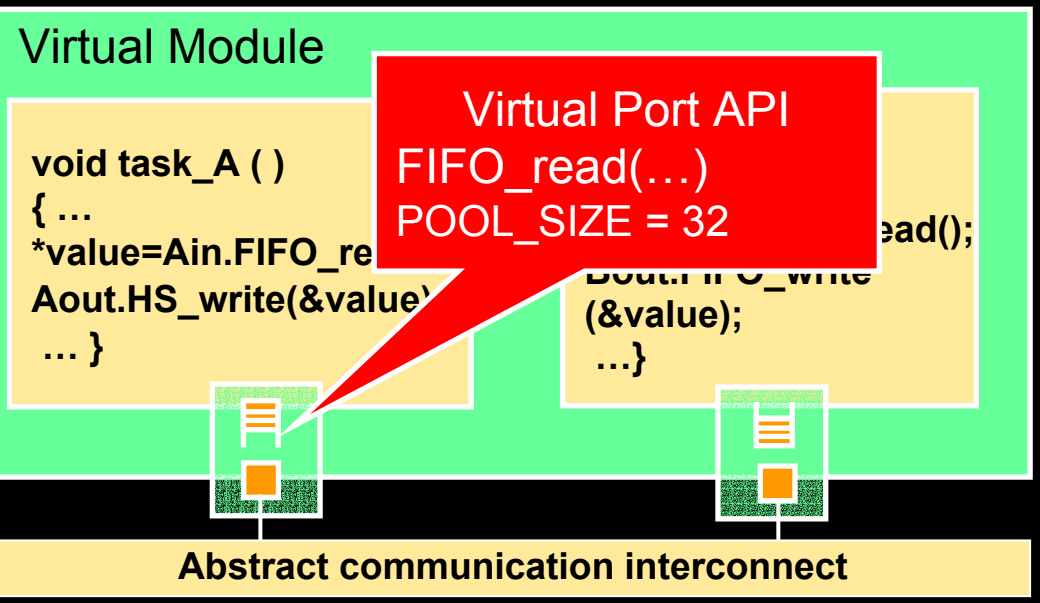
1. The challenges of HW-SW interfaces
2. HW-SW Interfaces for MPSoC
3. HW-SW abstraction for MPSoC: the concept of virtual architecture
4. ROSES: Automatic generation of HW-SW interfaces for MPSoC
5. HW-SW Interfaces in the design flow
6. Summary

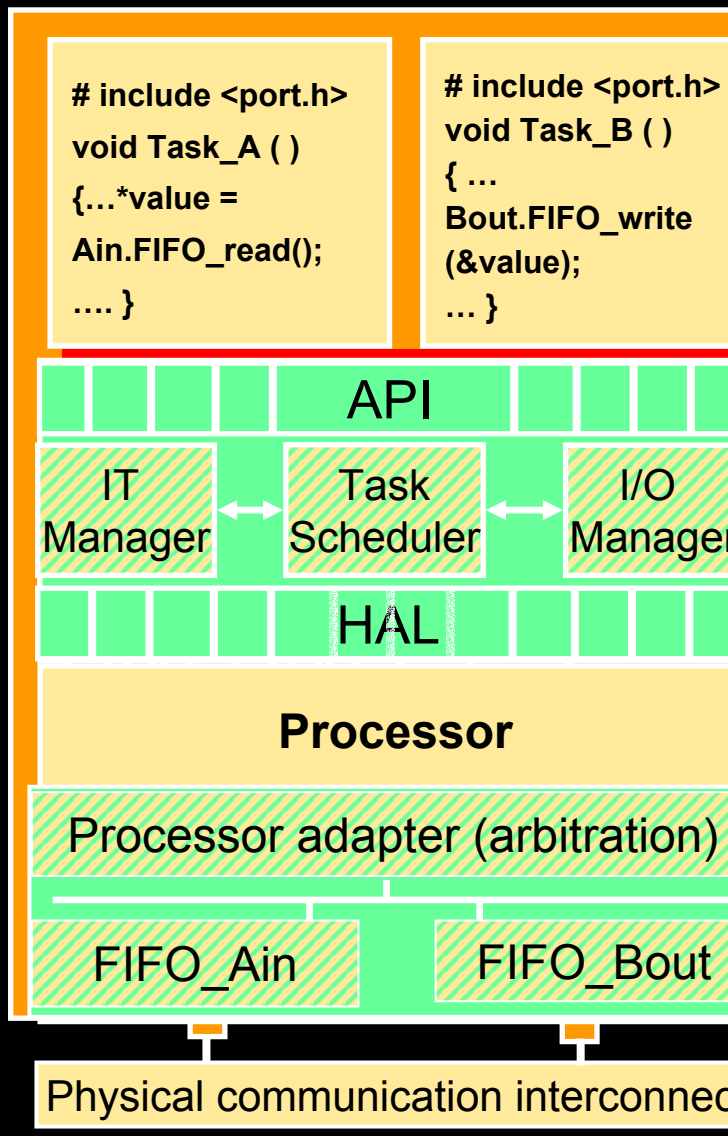
# ROSES: HW-SW Interfaces DA Flow

- System Specification as virtual architecture: Virtual modules (Components and NoC) use wrappers to abstract HW/SW communication e.g **Standard SW C++/SystemC Built on top of API**
- Architecture implementation as: heterogeneous components and sophisticated on-chip communication Network linked through HW and SW wrappers e.g **Same SW code, runs on implementation on top of OS**
- Automatic generation of application-specific on-chip HW/SW interfaces: **SW implementation of SW & HW adaptation**



# Communication services API example

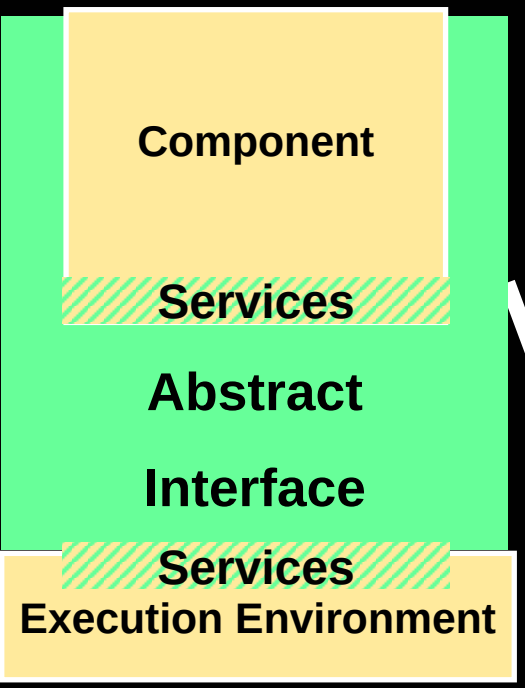


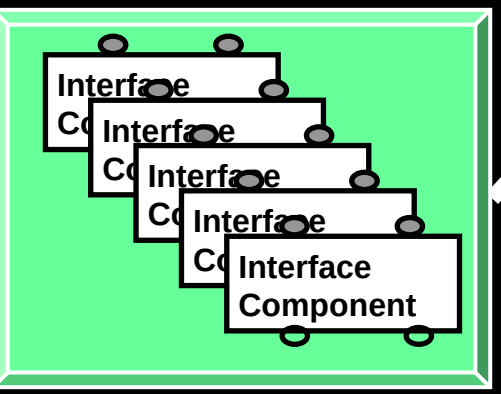

- **Virtual Architecture**
  - HW/SW wrapper: virtual ports (API, parameters)
  - SW, standard C++/SystemC built on top of API
- **RTL architecture**
  - Same SW code, runs on implementation
  - SW wrapper (implements API, task control, interrupts, I/O)
  - HW wrapper (bridge to communication interconnect)

# Key Technology: Building Interfaces

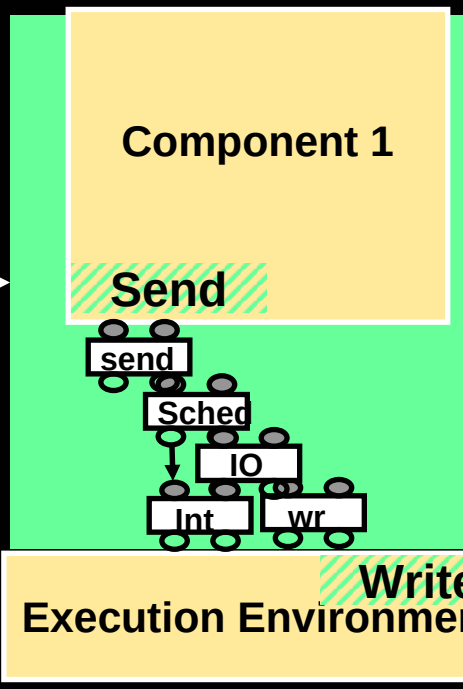
- Interface Component
  - Required/Provided Services
  - Control and Synch Signals
  - User Extendable Library
- Adaptation Layer Builder
  - Services Matching
  - Code Specialisation




Interface Component Library

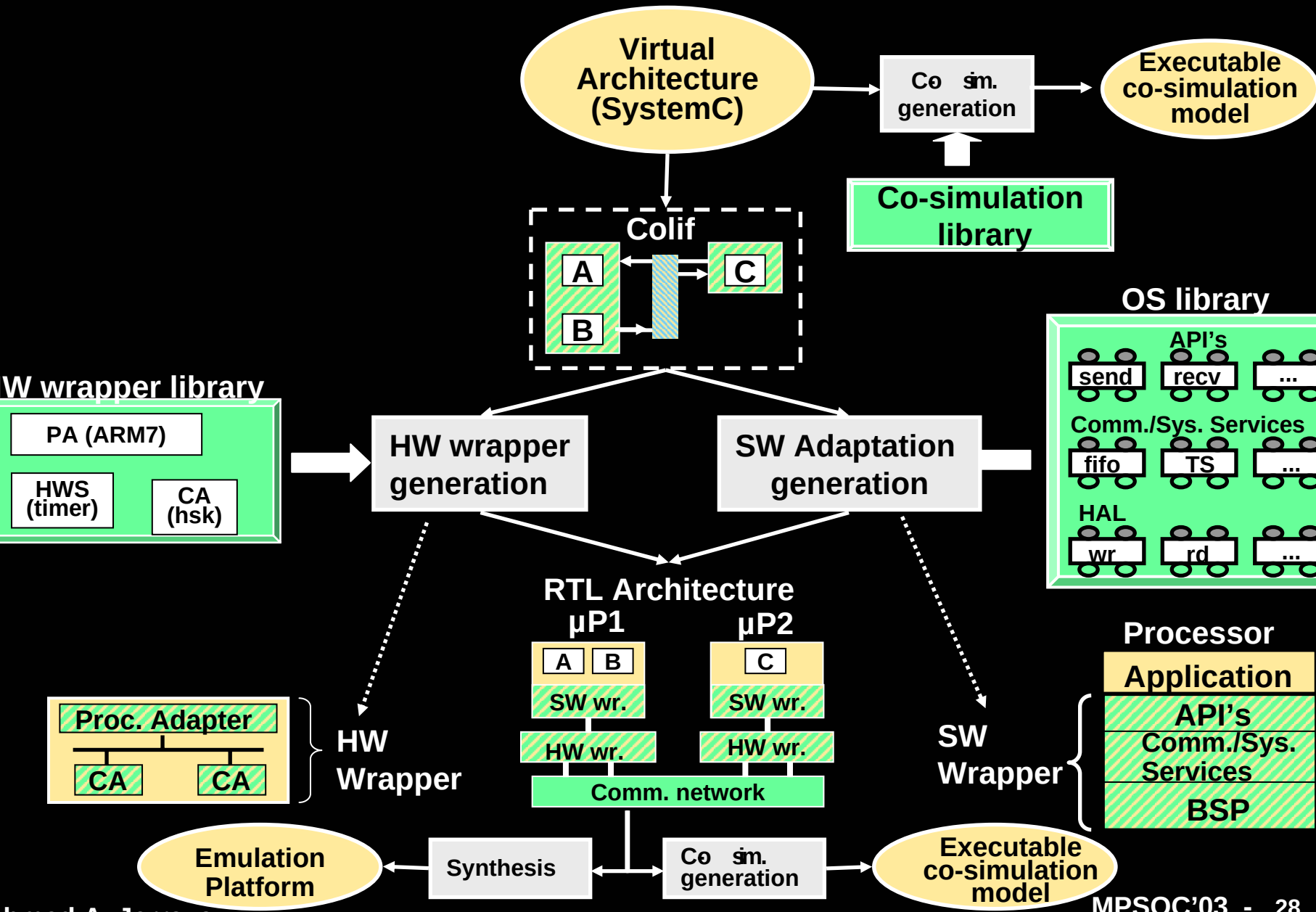



Adaptation Layer Builder




**Works for building SW, HW, Functional Wrappers**

# HW-SW Interfaces Generation Flow



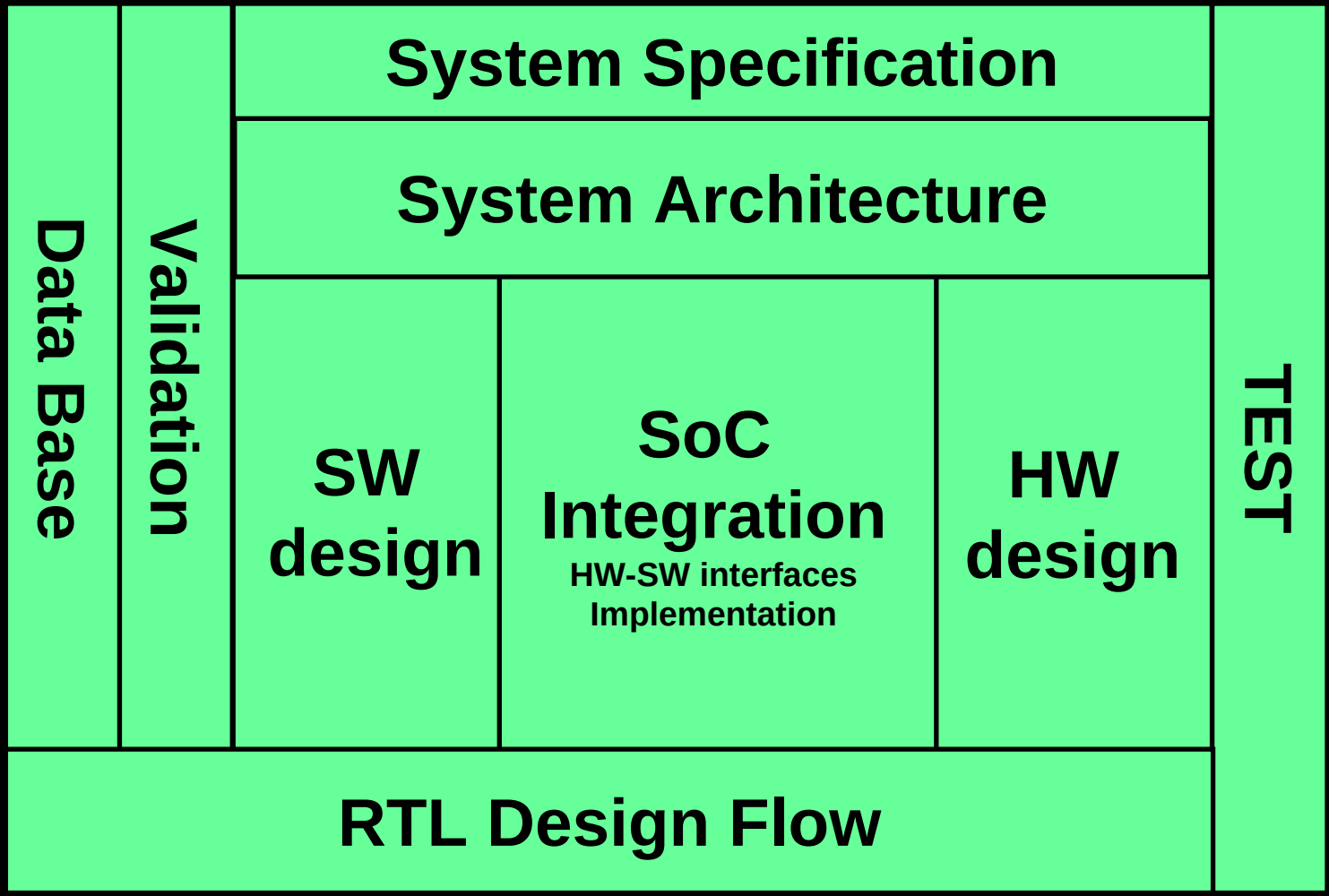
# ROSES on going activities

- Specification: Virtual architecture Model
  - HW SW interfaces at different Abstraction Levels
- Architecture exploration
  - Timed HW SW Interfaces Simulation at Different Abstraction levels
  - Library based simulation models of OS and HAL
  - Global system simulation
- Application specific HW SW interfaces implementation
  - Custom OS and Communication architecture generation
  - HW adaptation architectures
  - Partitioning HW SW interfaces
- SoC Integration
  - Targeting on Emulator
  - Implementation model, Syntetizable RTL
  - Cycle true Cosimulation Model
- Debug of hardware/software interfaces at different abstraction levels
  - Use OS generation flow to refine High Level test programs
  - Use same test programs at different abstraction levels

# Outline

1. The challenges of HW-SW interfaces
2. HW-SW Interfaces for MPSoC
3. HW-SW abstraction for MPSoC: the concept of virtual architecture
4. ROSES: Automatic generation of HW-SW interfaces for MPSoC
5. HW-SW Interfaces in the design flow
6. Summary

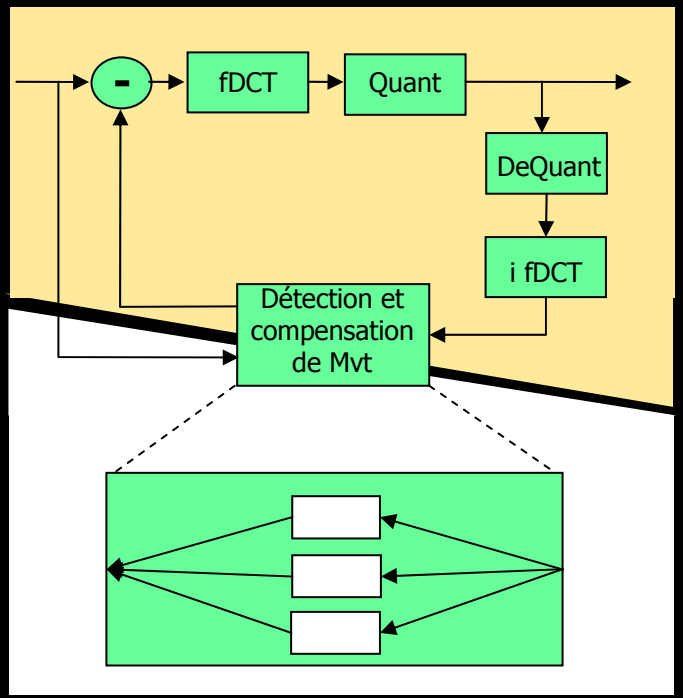
# System Design Flow Components




- HW-SW Interfaces are required by all parts.

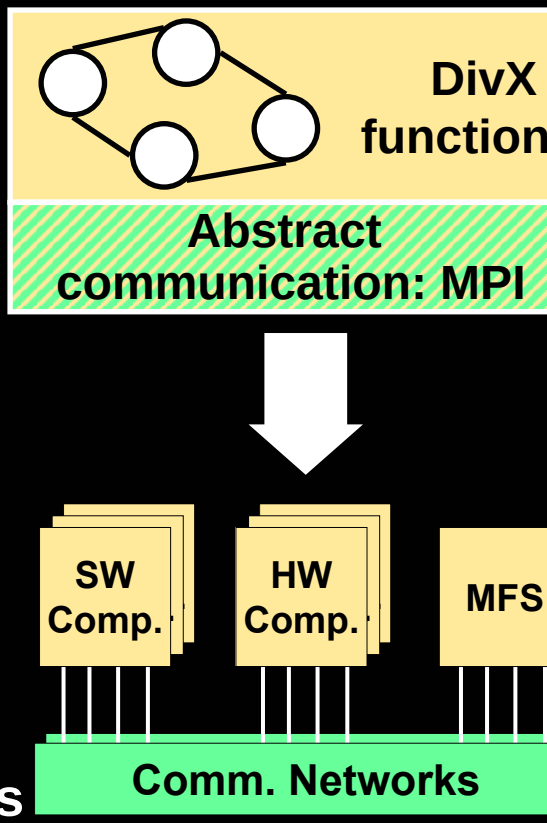
# MPSOC Design of an OpenDivX Encoder

- OpenDivX: free Mpeg4 encoder/decoder DivX
- Encoder OpenDivX: codes source video into DivX video
- Goal: Rapid design of DivX on MPSOC



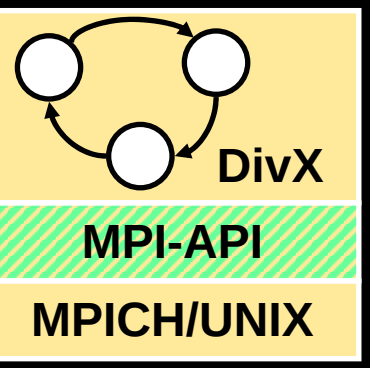
# OpenDivX Design Step

- Specification
  - C++/MPI
  - Validation 1: MPI/MPICH/LINUX
  - Validation 2: MPI/SystemC
- Architecture exploration
  - Manual partitioning
  - HL simulation
- SW design: reuse of HL C++ code
- HW design: use high performances architecture model
- HW-SW interfaces
  - Multilevel co-simulation
  - Automatic generation of HW-SW interfaces
- Implementation
  - Prototype, 4 processor ARM integrator platform

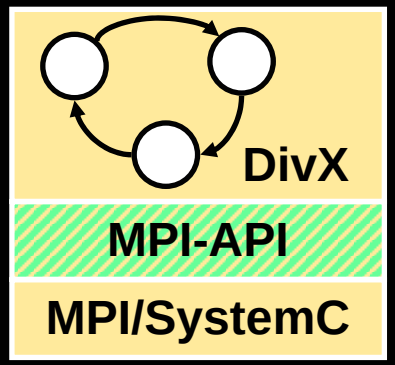


# DivX Main Design Steps

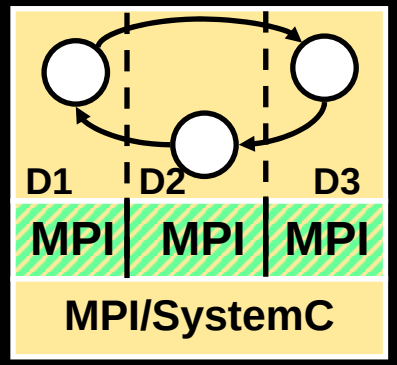
Initial algorithm

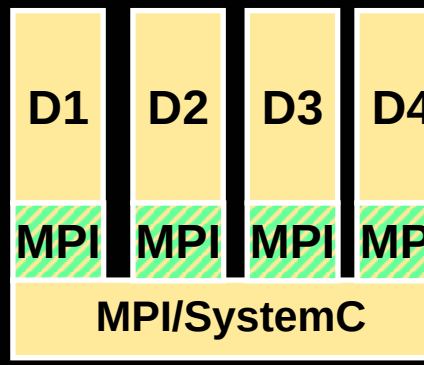



2. Same code on SystemC

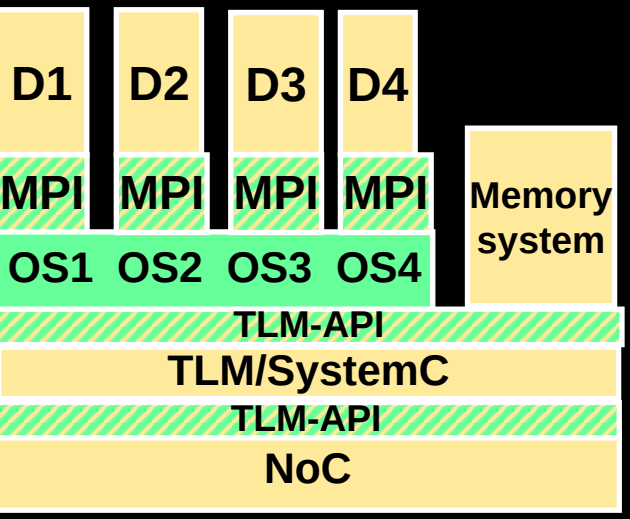



3. Same functions with different partitionings

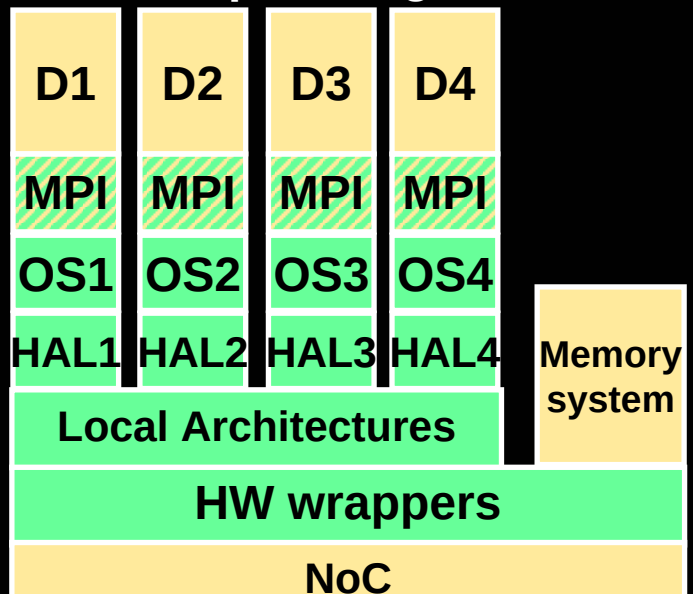



Fixing Communication
etwork/OS Generation



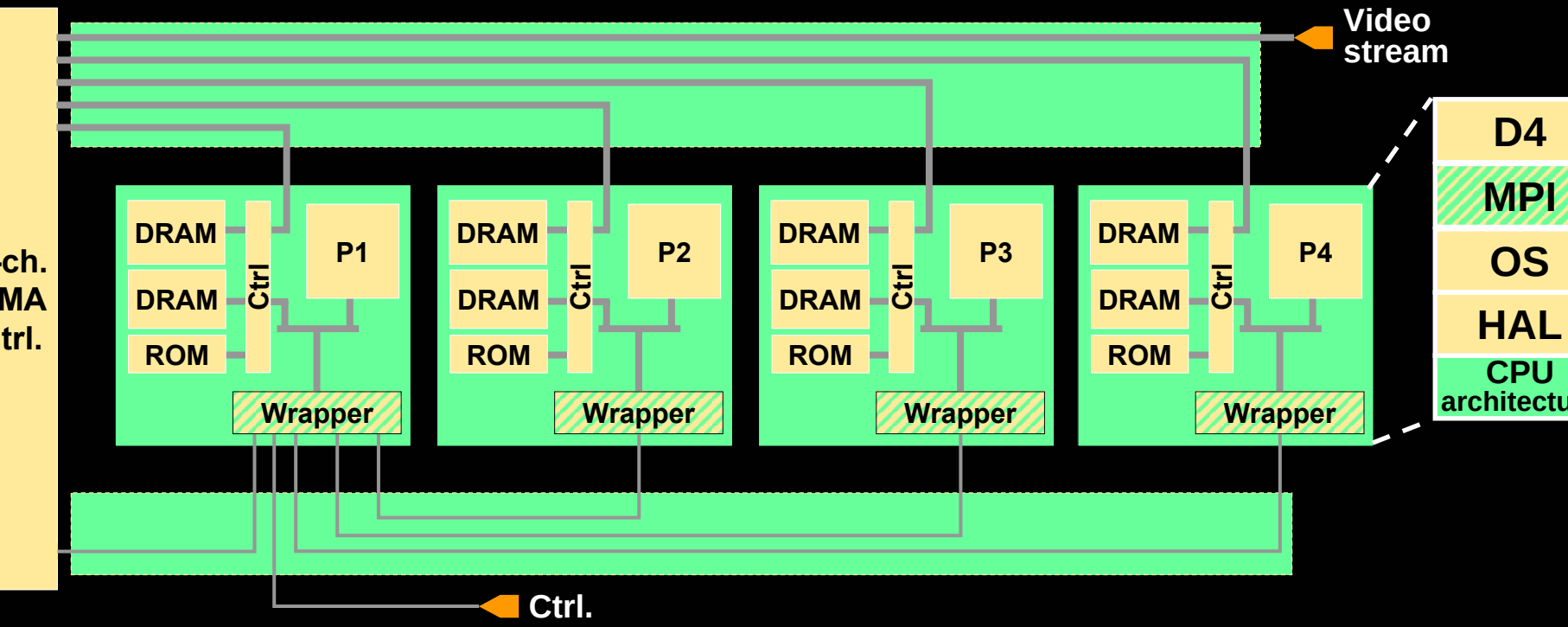

5. HW Adaptation generation




6. Implementation
- RTL design flo
- ARM integrator

# Architecture for OpenDivX Encoder

- Nexperia-like architecture
- 4-processor architecture : parallel execution of one master processor and 3 slave processors
- Point-to-Point communication networks
- Application-specific DMA controller
- Use of a double banc DRAM for local memories



# DivX Summary

- Initial specification uses HL communication interfaces to accommodate different partitioning & different communication networks.
- Custom OS generation allows different implementation of I/O, interrupts and resources management (different local architectures).
- HAL allows to accommodate different implementations and different CPU
  - RTL design flow
  - ARM integrator platform

**Key issues: multiple level validation and debug**

# Conclusion (1/2)

- SoC design requires a heterogeneous components interconnect scheme (CPU(s) + IP(s) + NoC)
- HW/SW interfaces design is the bottleneck for MPSoC
  - hard to master (Difficult to calibrate, Diversity, Complexity)
  - Application specific
  - Non rewarding part of the design flow
- HW-SW interfaces design may be automated:
  - Virtual Architecture Model
  - Custom OS/HAL to adapt SW
  - Custom Bridge to Adapt Hardware

# Conclusion (2/2)

- HW-SW Interfaces in the case of DivX design
  - Initial specification Parallel functions / MPI
  - Abstract partitioning: architecture exploration
  - Abstract OS/CPU/Bridges: performances tuning
  - Separate the design of HW, SW and NoC
  - Automatic Integration
  - Ease SoC Debug and Validation.
- Perspectives:
  - HW-SW interfaces Debug and Test (2004)
  - Partitioning HW/SW interfaces (2007)
  - Computation/Communication Partitioning (2010)

# Acknowledgement


- **Design modeling and Validation:**
  ***A.Dziri, L. Kriaa,, A. Moares W.Cesario***
- **Custom OS Modeling Generation and Optimization:**
  ***A.Bouchhima, Y.Paviot, W.Youssef, I. Bacivarov, S.Yoo***
- **Processor and memory interfaces generation:**
  **F. Gharsalli, A. Agrasset, *F. Rousseau,***
- **System Prototyping on Multi-ARM Platform :**
  ***A.Sasongho,* F.Rousseau**
- **HW-SW Interfaces Debug:**
  ***F.Hunsinger,***
- ***DivX MPSoC:***
  ***W. Youssef, Marius Bonaciau, G. Majauskas, I. Petkrov, A.Baghdadi***

# Readings about MP_SoC


1. D.E. Culler, J. Pal Singh, “Parallel Computer Architecture,” Morgan Kaufmann Publishers, 1999.
2. Oka and Suzuoki, “Designing and Programming the Emotion Engine,” IEEE Micro, vol. 19:6 pp. 20-28, Nov/Dec 1999.
3. J. T. J. van Eijndhoven, al. "TriMedia CPU64 Architecture," ICCD, Austin, TX, 1999, pp. 586 592.
4. K. Keutzer, “A Disciplined Approach to the Development of Platform Architectures,” Synthesis and System Integration of Mixed Technologies, SASIMI, Nara, Japan, October 18 19, 2001. .
5. M. Sgroi, et al., “Addressing the System-on-Chip Interconnect Woes Through Communication-Based Design,” Proc. of 38th Design Automation Conference, Las Vegas, June 2001.
6. IBM Inc., Blue Logic Technology, http://www.chips.ibm.com/bluelogic/
7. D. Wingard, “MicroNetwork-Based Integration for SOCs,” Proc. of DAC, Las Vegas, June 2001.
8. J. A. J. Leijten et al., “PROPHID : A Heterogeneous Multi-Processor Architecture for Multimedia,” Proc. of ICCD, 1997.
9. L. Benini, G. De Micheli, “Networks on chips: A New SoC Paradigm”, Computer, Vol. 35 No 1, pp. 70-78, January 2002.
10. K. Goossens, E. Rijpkema, P. Wielage, A. Peeters and J. van Meerbergen, “Networks on Silicon: The next Design paradigm for systems on Silicon”, DATE 2002, Paris, France, March 2002.
11. David E. Patterson, John L. Hennessy, “Computer Organization & Design - The Hardware/Software Interface”, Morgan Kaufmann Publishers, 1998.

# Readings about Roses


W. CESARIO, al., "Component-Based Design Approach for Multicore SoCs", DAC'02, New Orleans, USA June 10-14 2002.

S. Yoo, al. « A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design », *CODES*, 2001.

W.O. Cesario, al. "Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design", IEEE Design & Test, Sept, 2001.

A. Baghdadi, al. « An Efficient Architecture Model for Systematic Design of Application-Specific Multi-processor SoC », *Design Automation and Test in Europe*, March, 2001.

L. GAUTHIER, al., "Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software", TCAD, IEEE Transactions on Computer-Aided Design, Vol. 20 Nr. 11, November 2001.

D. Lyonnard, al. "Automatic Generation of Application-Specific Architecture for Heterogeneous Multiprocessor System-on-Chip", DAC, June, 2001.

S. YOO, G. NICOLESCU, L. GAUTHIER, A.A. JERRAYA, "Automatic Generation Including Fast Timed Simulation Models of Operating Systems in Multiprocessor SoC Communication Design", DATE 2002, Paris, France, March 2002.

P. Gerin, al. « Scalable and Flexible Co-simulation of SoC Design with heterogeneous Multi-processor Target Architecture », *Asia South Pacific Design Automation conference*, January, 2001.

F. GHARSALLI, al., "Embedded Memory Wrapper Generation for Multiprocessor SoC", DAC'02, June 10-14 2002, New Orleans, USA.

# Reading about System Specification


1. A. Lee and A. Sangiovanni-vicentelli, A Denotational Framework for Comparing Models of Computation, ERL Memorandum UCB/ERL-M97/11, University of California, Berkley, CA 94720, January 1997.
2. A. Jantsch, S. Kumar, A. Hemani, « The Rugby Model: A Metamodel for Studying Concept in Electronic System Design », *IEEE Design & Test of Computers*, 2000, p. 78-85.
3. D. D. Gajski, J. Zhu, R. Zömer, A. Gerstlauer, S. Zhao, *SpecC Specification Language and Methodology,* Kluwer Academic Publishers, Boston, MA, ISBN 0-7923-7822-9, March 2000
4. M. Sgroi, L. Lavagno, A.S. Vicentelli, « Formal Models for Embeded System Design », IEEE Design & Test of Computers, vol. 17, no. 12, April-June 2000.
5. SystemC, available at http://www.systemc.org
6. R. Ernst, D. Ziegenbein, K. Richter, L. Teich, "Hardware/Software Co-Design of Embedded Systems - The SPI Workbench," Proc. IEEE Workshop on VLSI'99, pp. 9-17, Orlando, 1999
7. W.O. Cesario, al. "Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design", IEEE Design & Test, Sept, 2001.

# Reading about Heterogeneous Systems


J.A. Rowson « Hardware/Software Co-simulation », proceeding Design Automation Conference, 1994.

C.A. Valderrama, A. Changuel, P.V. Vijaya-Raghavan, M. Abid, T. Ben Ismail, A.A. Jerraya, "A unified model for co-simulation and co-synthesis of mixed hardware/software systems", European Design and Test Conference (EDAC-ETC-EUROASIC'95), Paris, France, March 1995.

L. Séméria and A. Ghosh, “Methodology for Hardware/Software Co-verification in C/ C++” , Proceeding of ASPDAC, 2001.

Seamless CVE, available at http://www.mentorg.com

C. Passerone, L. Lavagno, M. Chiodo, A. Sangiovanni-Vincentelli “Fast hardware/software co-simulation for virtual prototyping and trade-off analysis”, in Proceedings of Design Automation Conference, June, 1997

Coware, Inc. “N2C” , available at http : // coware.com/cowareN2C.html

S. Yoo, al. « A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design », *CODES*, 2001.

P. Gerin, al. « Scalable and Flexible Co-simulation of SoC Design with heterogeneous Multi-processor Target Architecture », *Asia South Pacific Design Automation conference*, Januar 2001.

G. Nicolescu, S. Yoo, A.A. Jerraya, « Mixed-Level Cosimulation for Fine Gradual Refinement of Communication in SoC Design », *Design Automation and Test in Europe*, mars, 2001.

# *Thank You*