A Model Based System-On-Chip Design Methodology: SystemC in practice

Yves Vanderperren, Wim Dehaene

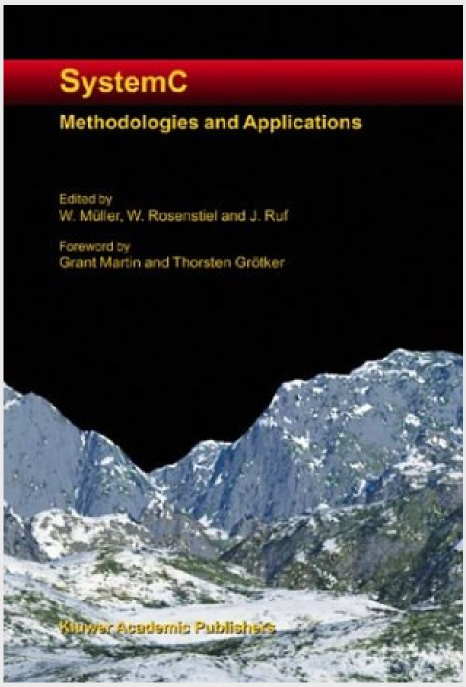
yves.vanderperren@esat.kuleuven.ac.be

Katholieke Universiteit Leuven
Department of Electrical Engineering (ESAT)

Note

This presentation summarizes the work done at Alcatel Microelectronics in 2001, as published in:

"SystemC – Methodologies and Applications"
Kluwer Academic Publishers, July 2003
ISBN: 1402074794



see also:

EUROPEAN SYSTEMC™ USERS GROUP

Presentations at 5th and 6th European SystemC Users Group Meetings

Outline

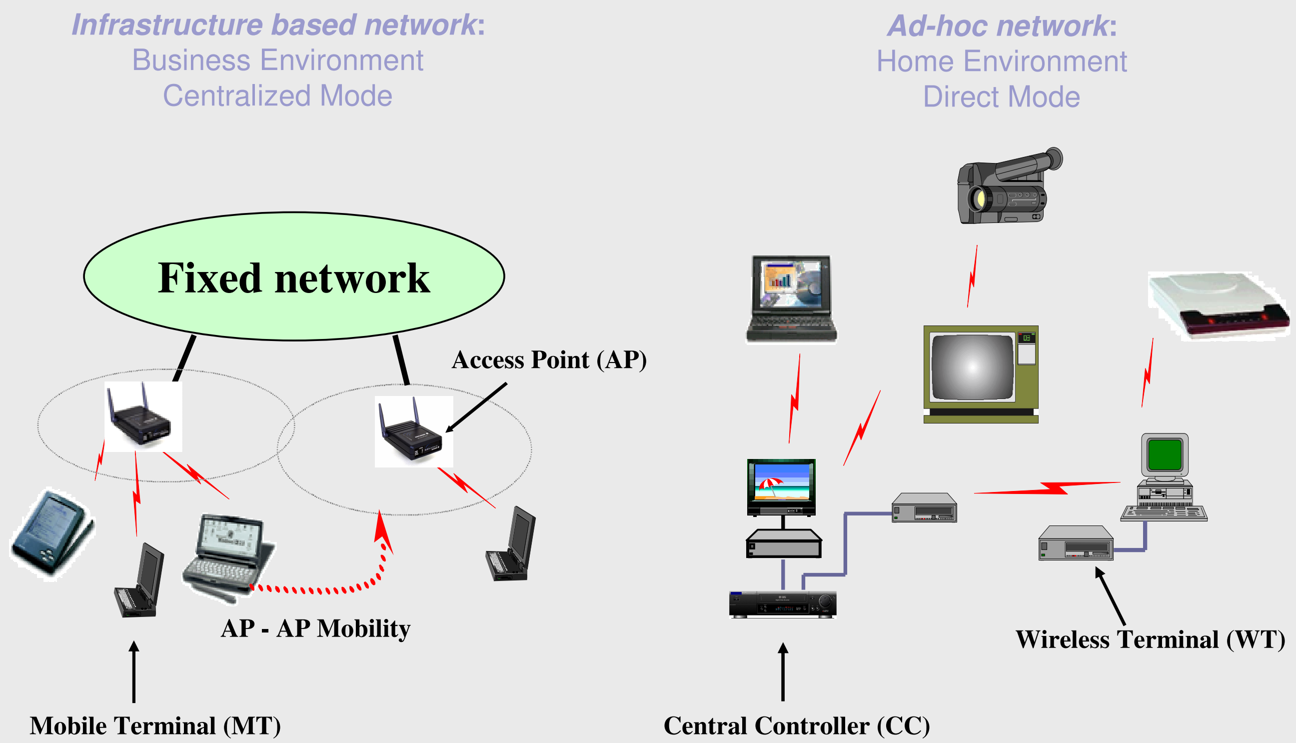
- Stating the case
  - Evolution towards SoC
  - Concrete example: the OFDM Wireless LAN Project at AmE
- Methodology overview
  - Process
  - Modelling Strategies
    - Role of SystemC
    - SystemC Modelling
    - Model Refinement and Verification
- Conclusions

Stating the case: *Evolution towards SoC*

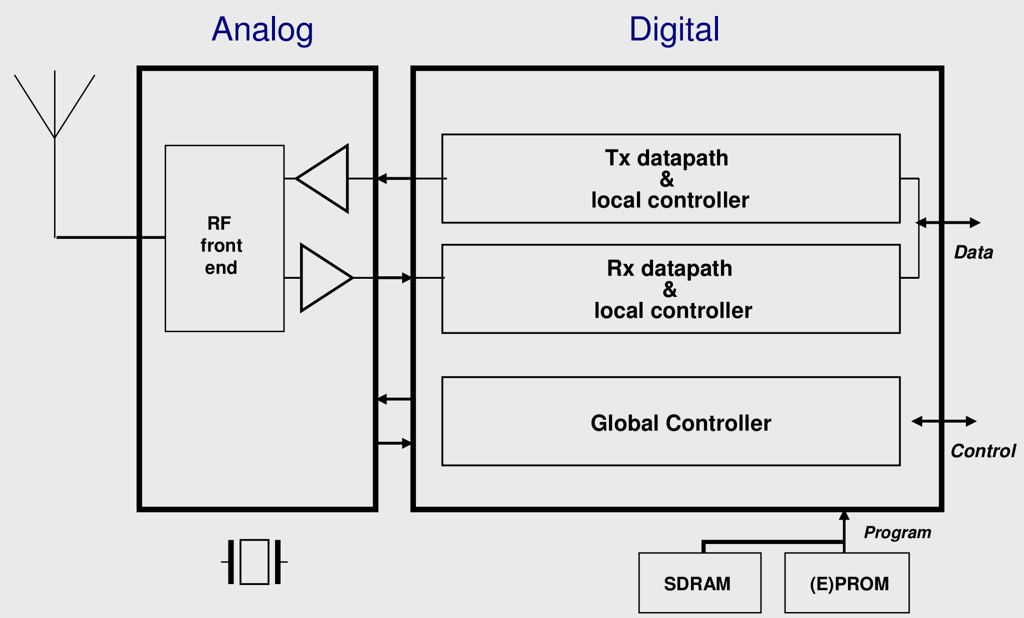
- Evolution of Manufacturing Technology
  - smaller feature size
  - with possibility of larger functions on the die
- Evolution of System Designs
  - get ultimately single chip (SoC) for reasons of
    - cost saving
    - compactibility
- SoC are complex and contain many components: Hw,Fw,Sw
  - all must inter-operate correctly and
  - must fit to Customer specifications
  - must be developed with faster-time-to-market

► Need for System-on-Chip (SoC) Design Methodology

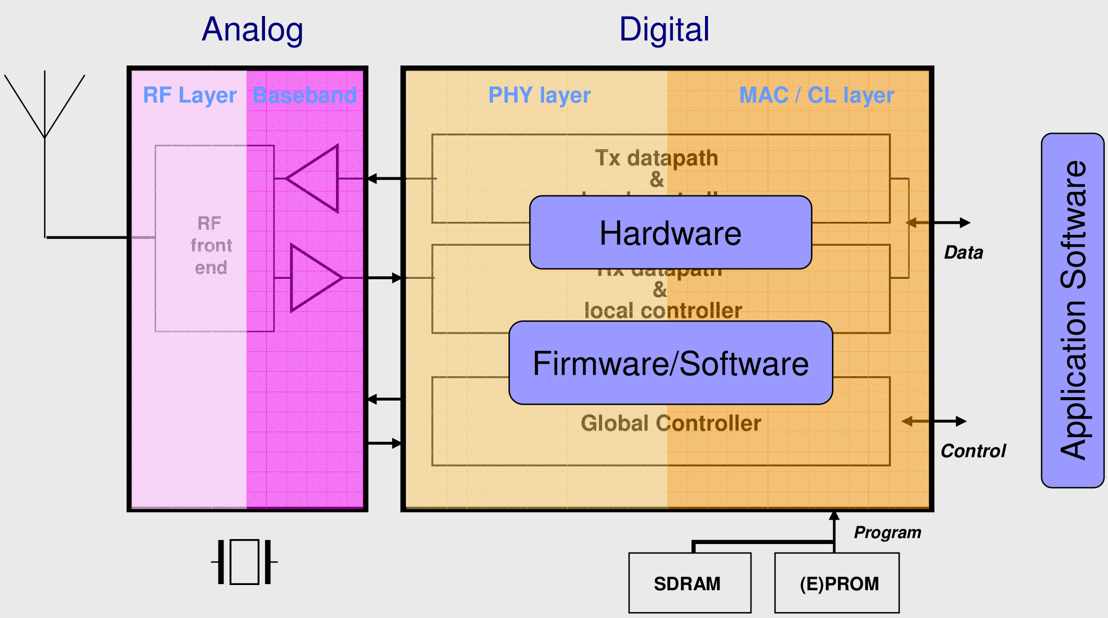
Stating the case: *the OFDM Wireless LAN Project at AmE*



Stating the case: *the OFDM Wireless LAN Project at AmE*



Stating the case: *the OFDM Wireless LAN Project at AmE*



Outline

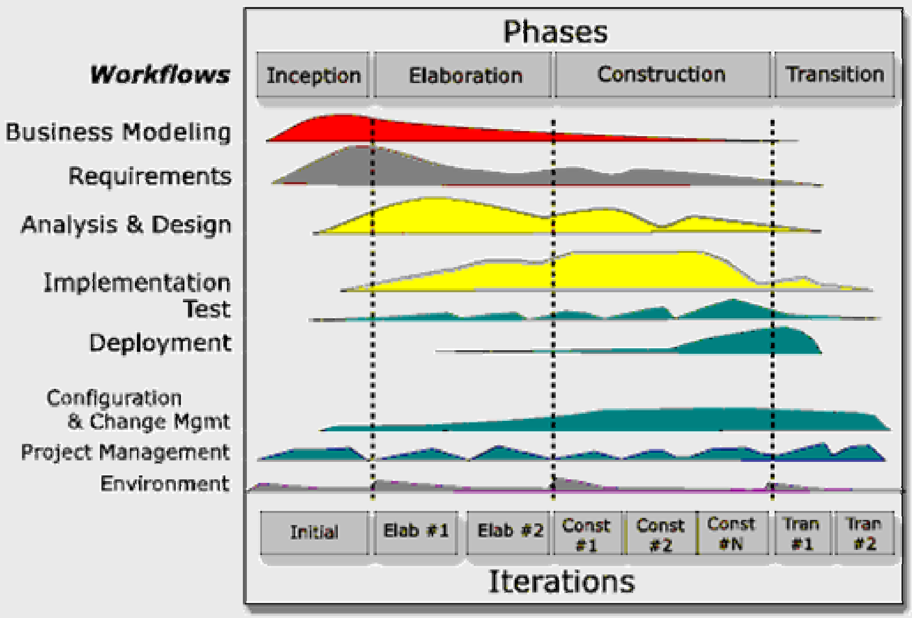
- Stating the case
  - Evolution towards SoC
  - Concrete example: the OFDM Wireless LAN Project at AmE
- Methodology overview
  - Process
  - Modelling Strategies
    - Role of SystemC
    - SystemC Modelling
    - Model Refinement and Verification
- Conclusions

Methodology Overview

- Iterative Development
  - Refine system implementation to address risks early
- Use Case Driven Architecture
  - Validate architectural design using Use Cases
  - Visual Modelling: UML
- Executable System Models & Model-Centric Development
  - Approach the design as a series of explorations and refinements
- Test
  - Evolves during each iteration in parallel with system
  - Allows efficient regression testing of each iteration
  - Models reused throughout iterations

Iterative Development

Rational Unified Process (RUP)



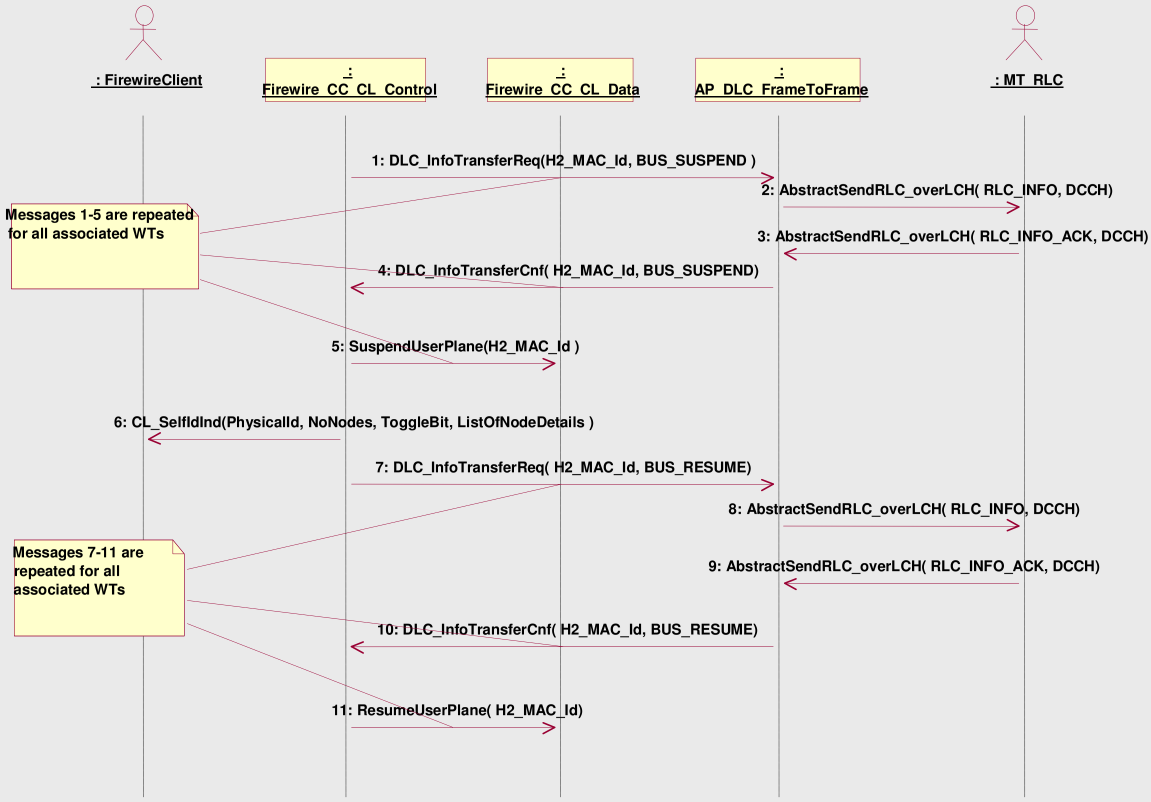
Iterative Development

	S Feasibility, Requirements and Algorithms	It 0 Architecture and High Level Modelling	It 1 Detailed Design and Cosimulation	It 2 Hardware Prototype (FPGA)	It 3 Silicon
Systems	Capture reqts; Agree Vision Doc; Create UC model; Develop key algo's	Specify Architecture; Create HL SystemC / UML model; Demonstrate architecture meets project requirements	Cosimulation of SystemC, VHDL and SW	Cosimulation of SW on target and HW on FPGA; Conduct system V&V	Silicon Verification; Product Qualification or Approvals
Hardware		Involvement with SystemC specification	Detailed Design of VHDL	Port design to FPGA	Back end design and silicon fab
Software		Specify SW Arch; Create Host-Based SW framework and basic functionality	Build further SW functionality on It0 framework; Port to target	Build further functionality on It0/It1 framework	Build further SW functionality
Model or Platform used	Matlab	SystemC: Matlab as reference for floating point operations	VHDL & C: SystemC as reference for cosimulation	FPGA	Product on silicon
Test target	Function	Timing (TF mostly CA for critical parts)	Check design vs. SystemC reference	Full system operation	Full real-time system operation

Use Case Analysis

- Mechanism to specify functional requirements related to top-level view
  - UC = a service providing value to an external entity
- UC's are basically textual – a behavioural sequence
  - UML diagrams help to organise and document
- For each UC we document a.o.:
  - Primary (expected) system responses → ✓ scope of the system, user goals
  - Secondary system responses (e.g. error conditions) → ✓ failure handling
  - Links to other UC's

Use Case Analysis

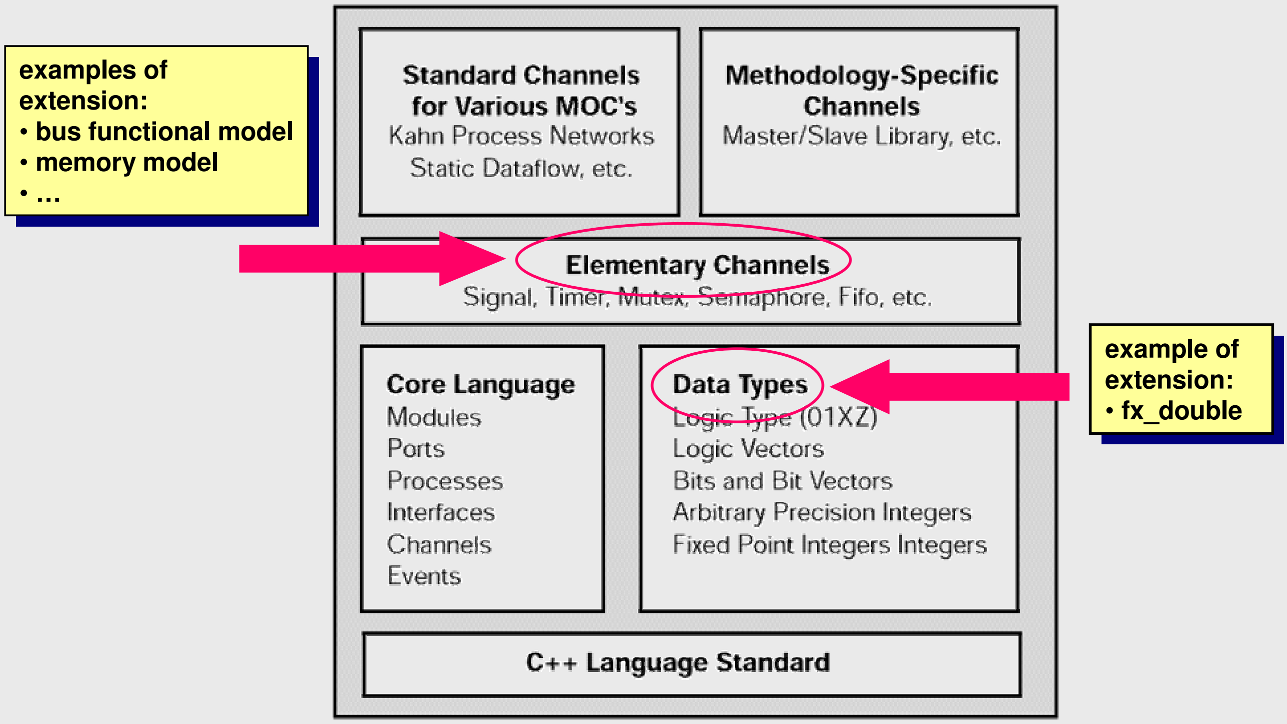


Executable System Models: *introduction to SystemC*

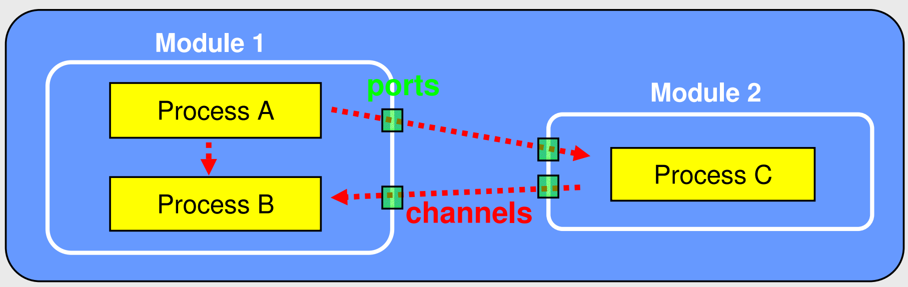
- System design language for HW/SW co-design
- A library of C++ classes
  - Processes
  - Events
  - Hardware data types
  - Modules, ports
  - Channels
- A light-weight simulation kernel
- Allows to make an *"Executable Specification"*

✓ concurrency
✓ time, reactivity
✓ finite precision
✓ hierarchy
✓ communication

Executable System Models: *introduction to SystemC*



Executable System Models: *introduction to SystemC*



port direction

port name

port type

```
SC_MODULE(accumulator) {
   // ports
   sc_in<int>   In;
   sc_in<bool> Clk;
   sc_out<int> Out;

   // process
   void accumulate();

   // local member data
   int Acc_;

   // constructor
   SC_CTOR(accumulator) {
      SC_METHOD(accumulate);
      sensitive_pos << Clk;
      Acc_ = 0 ;
   }
};
```

.h file

registration of the process with the SystemC kernel

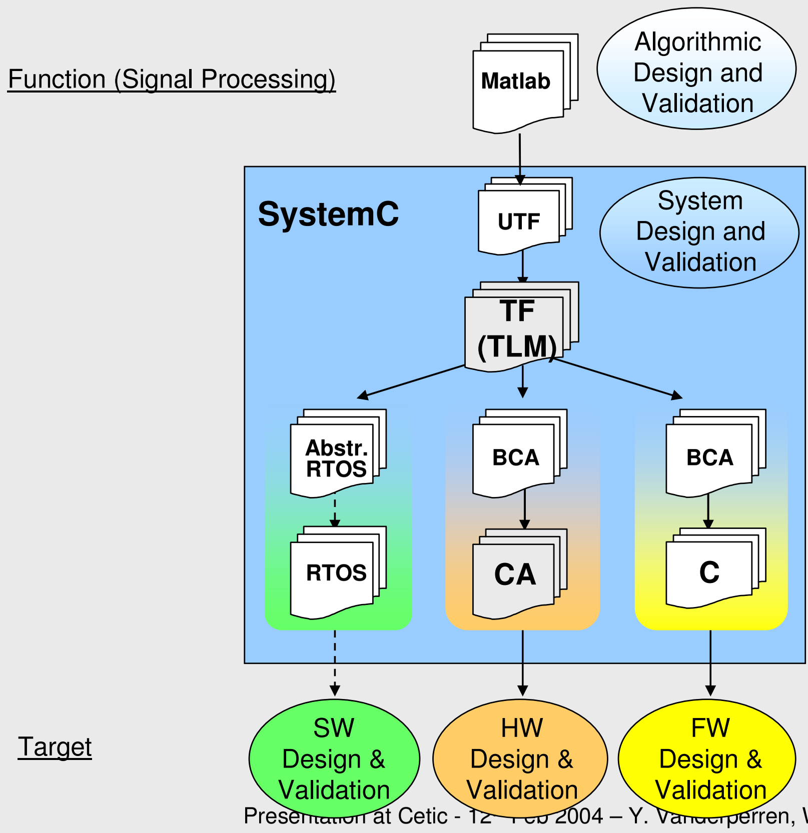
```
#include "… .h"

void accumulator::accumulate()
{
   Acc_ += In.read();

   Out.write(Acc_);
}
```

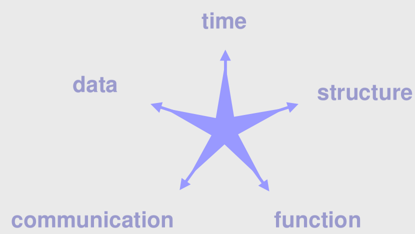
.cc file

Executable System Models: *role of SystemC*

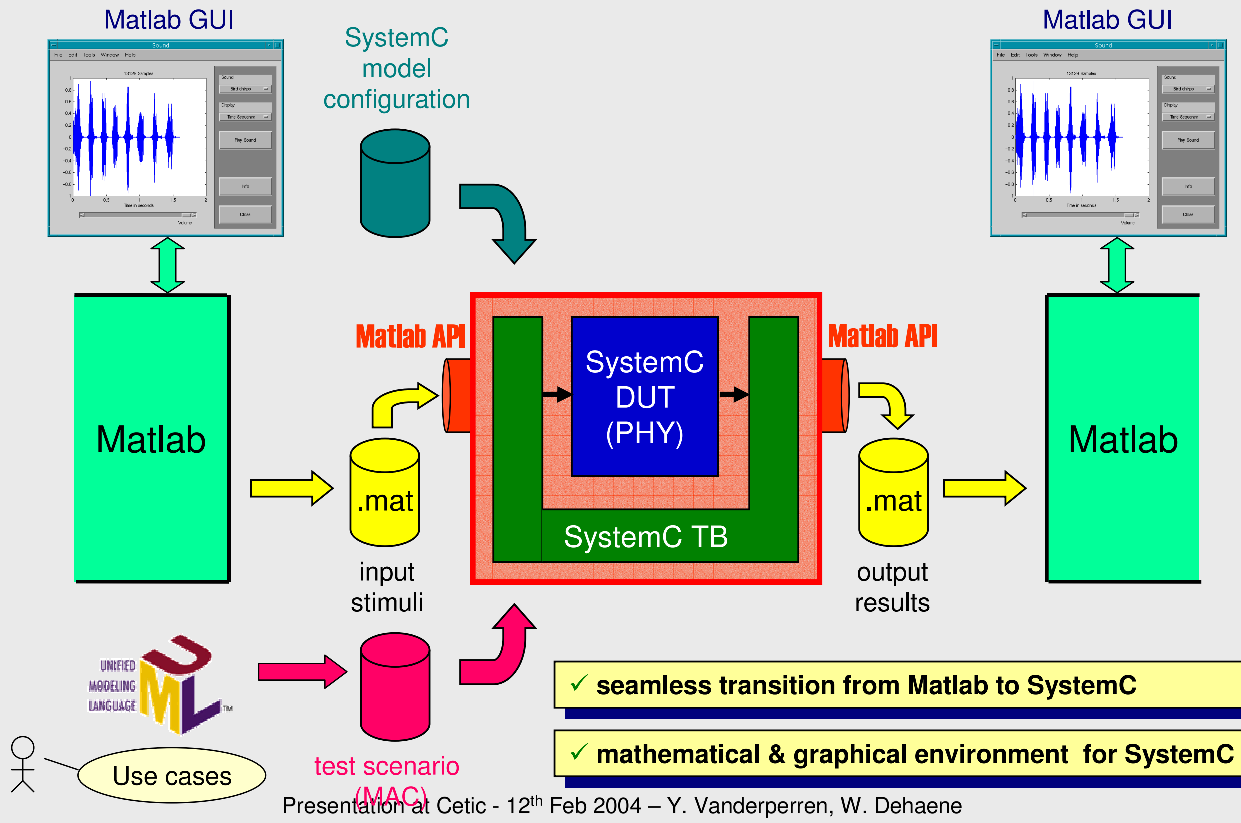
- ***An Executable Specification*** for communication with designers
  - Designers participate in writing the specification – knowledge transfer
- ***A Tool*** providing
  - a starting point for HW implementation and verification
    - detail the architecture for critical blocks
    - do finite-precision design
  - a platform serving as a HW model for SW development
  - an early verification of the overall system behavior and architecture, by running test scenarios
    - design space exploration
- ***A Reference***
  - testbenches are the starting point for all other testbenches and system test plans (Lab qualification)

Executable System Models: *role of SystemC*



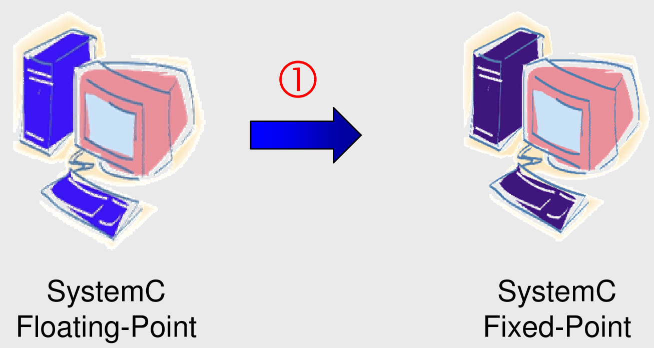


Executable System Models: *Matlab ► SystemC*



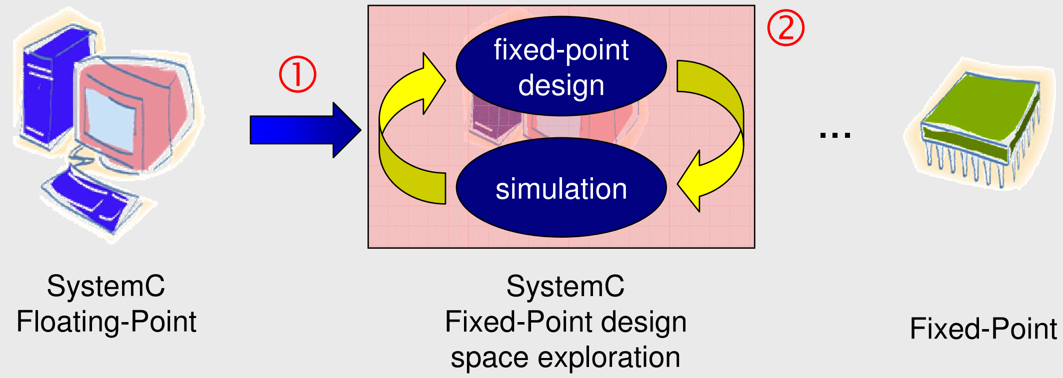
- ✓ **seamless transition from Matlab to SystemC**
- ✓ **mathematical & graphical environment for SystemC**

Executable System Models: *fixed-point design*



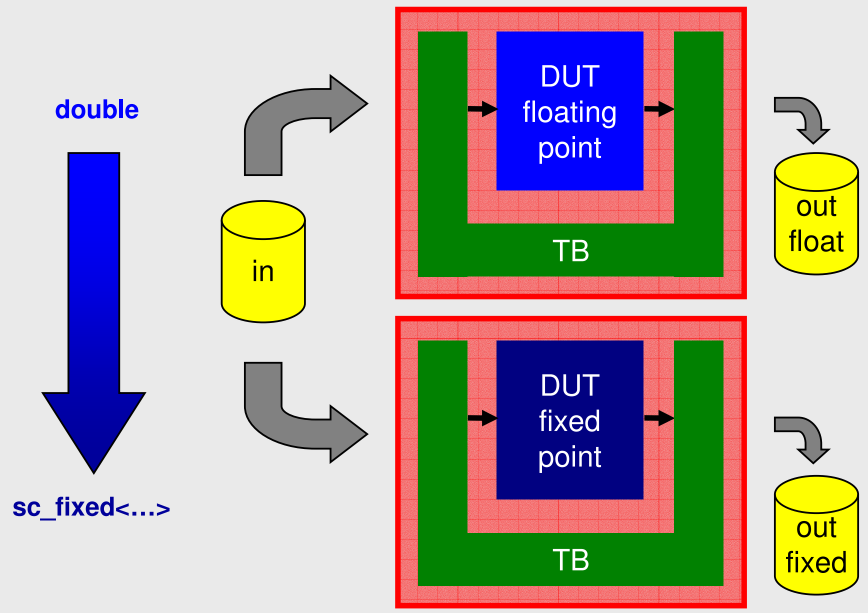
- Conversion of the SystemC model to fixed-point

Executable System Models: *fixed-point design*



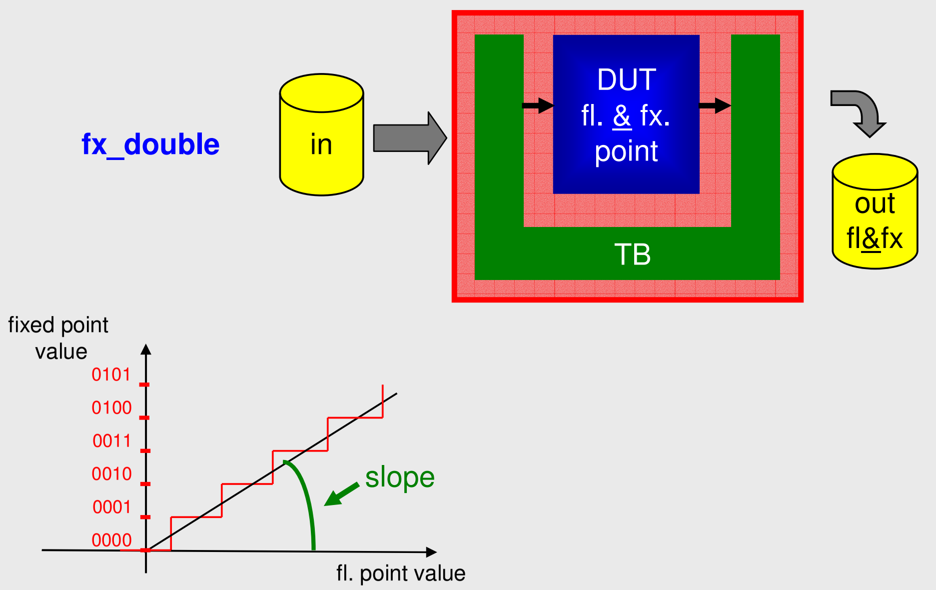
- Conversion of the SystemC model to fixed-point
- Fixed-point design space exploration
  - scale properly
    - avoid overflow
    - avoid quantization error
  - minimize cost & power with precision constraint
    - use as few bits as possible

Executable System Models: *fixed-point design*



- × specific operations
- × conversion needed
- × 2 models
- × maintenance
- × no scale
- × fixed-point design space exploration
- × ...

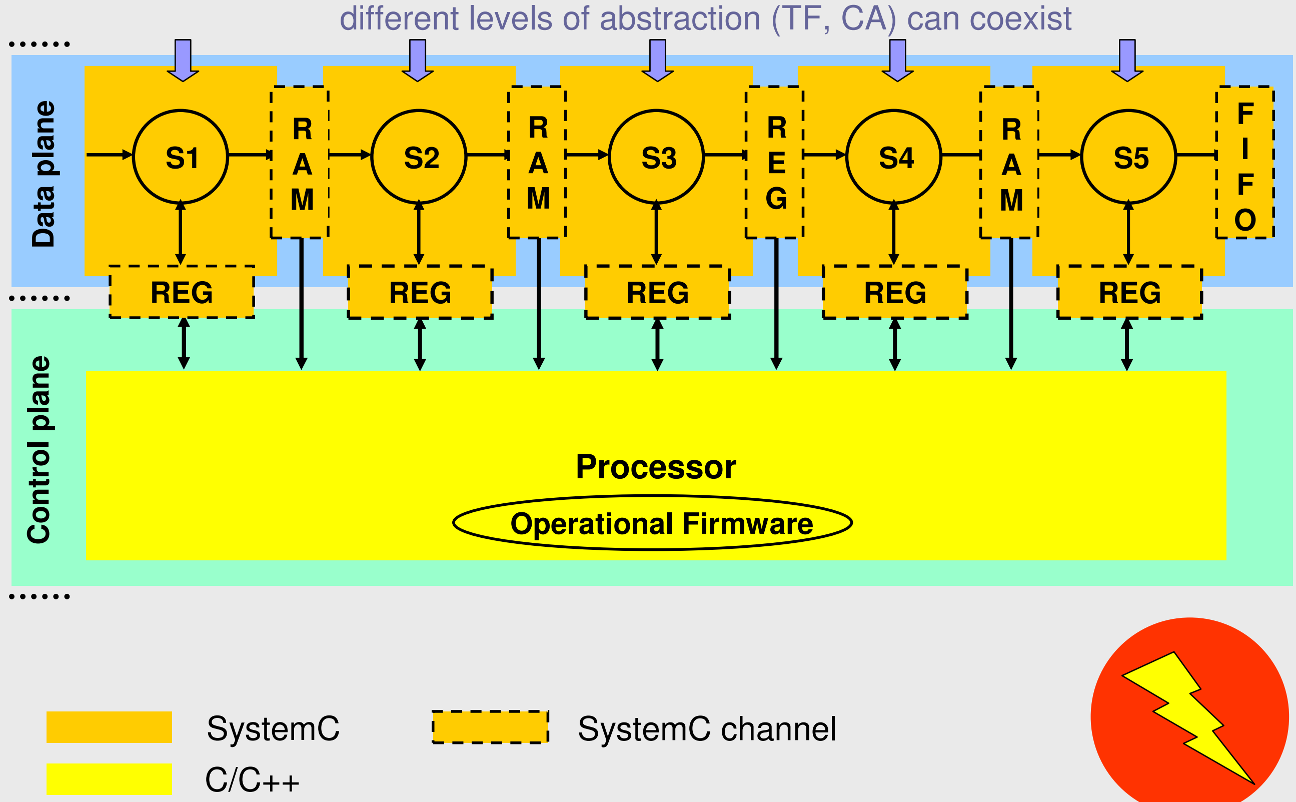
Executable System Models: *fixed-point design*



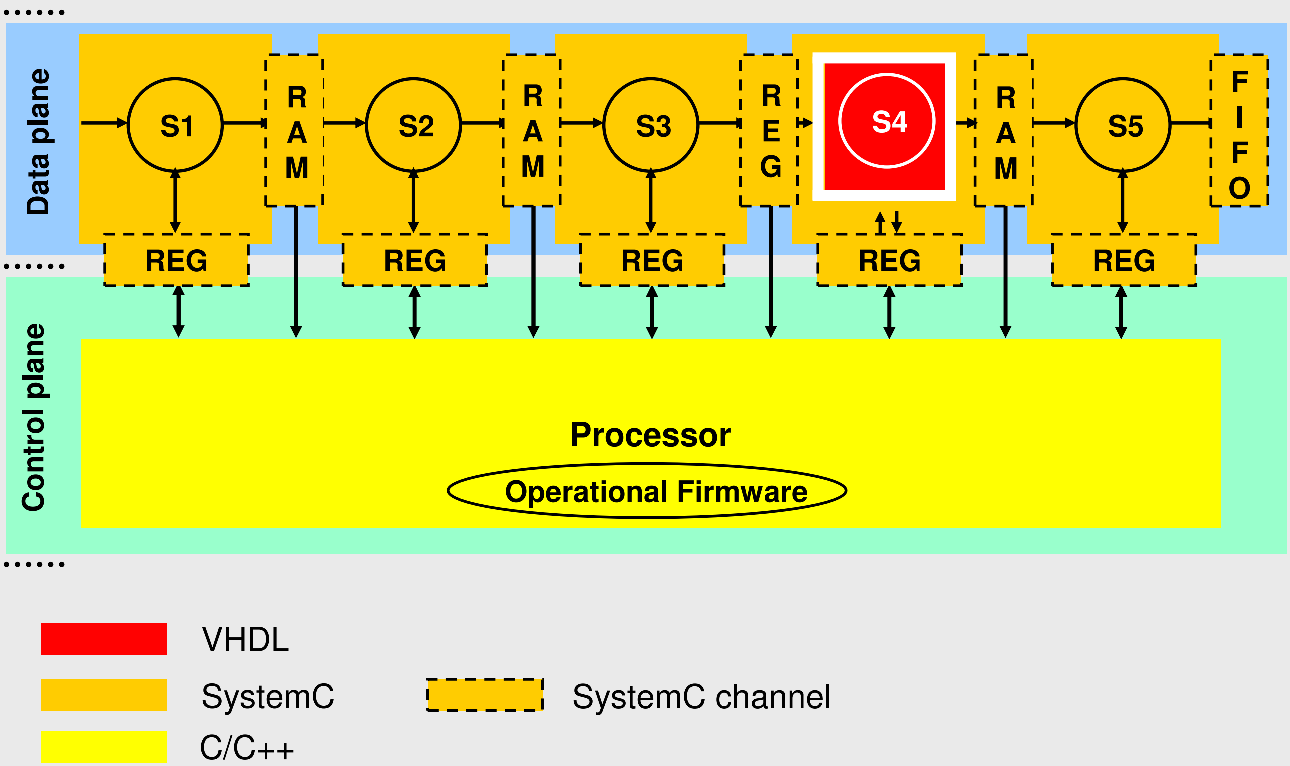
- ✓ common operations
- ✓ no conversion
- ✓ 1 model
- ✓ maintenance
- ✓ scale propagation
- ✓ dynamic scale
- ✓ strong type-checking
- ✓ fixed-point design space exploration
- ✓ common interfaces, enhanced reuse
- ✗ simulation speed

Executable System Models: *SystemC ► Implementation*

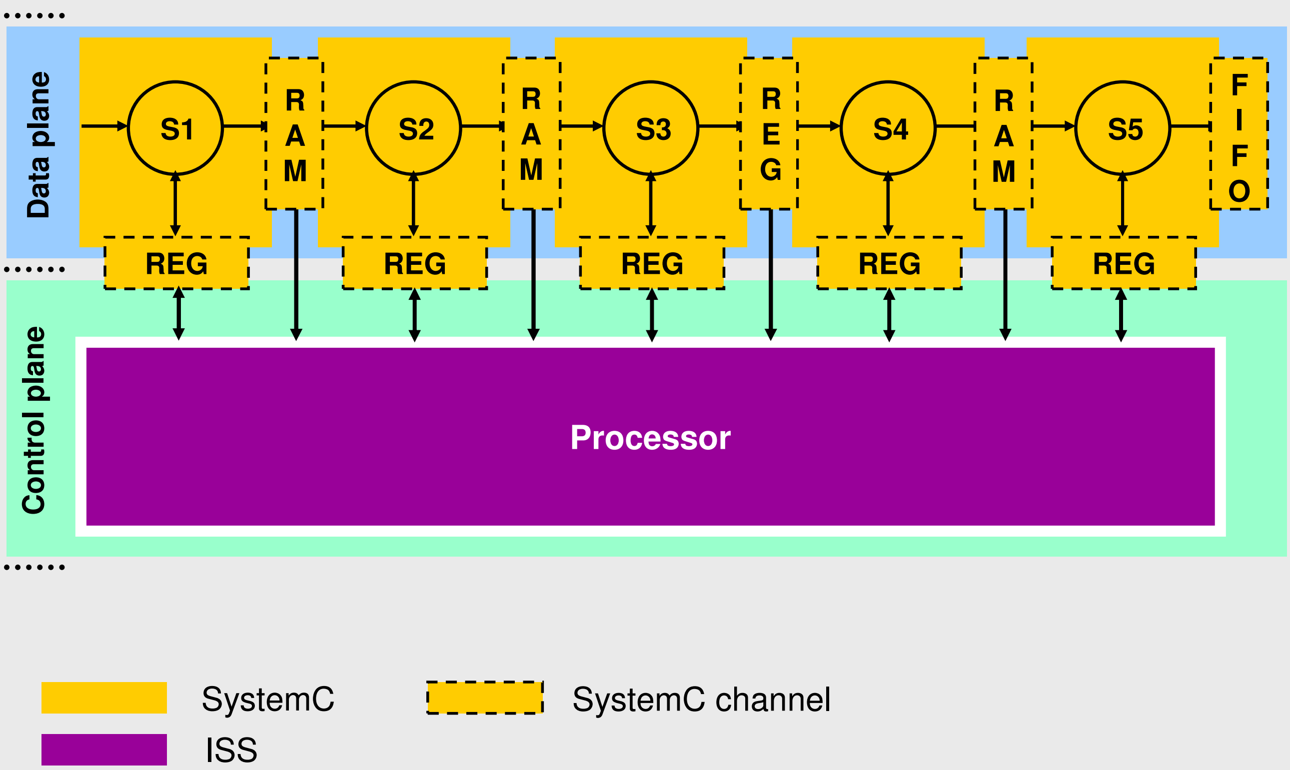
different levels of abstraction (TF, CA) can coexist



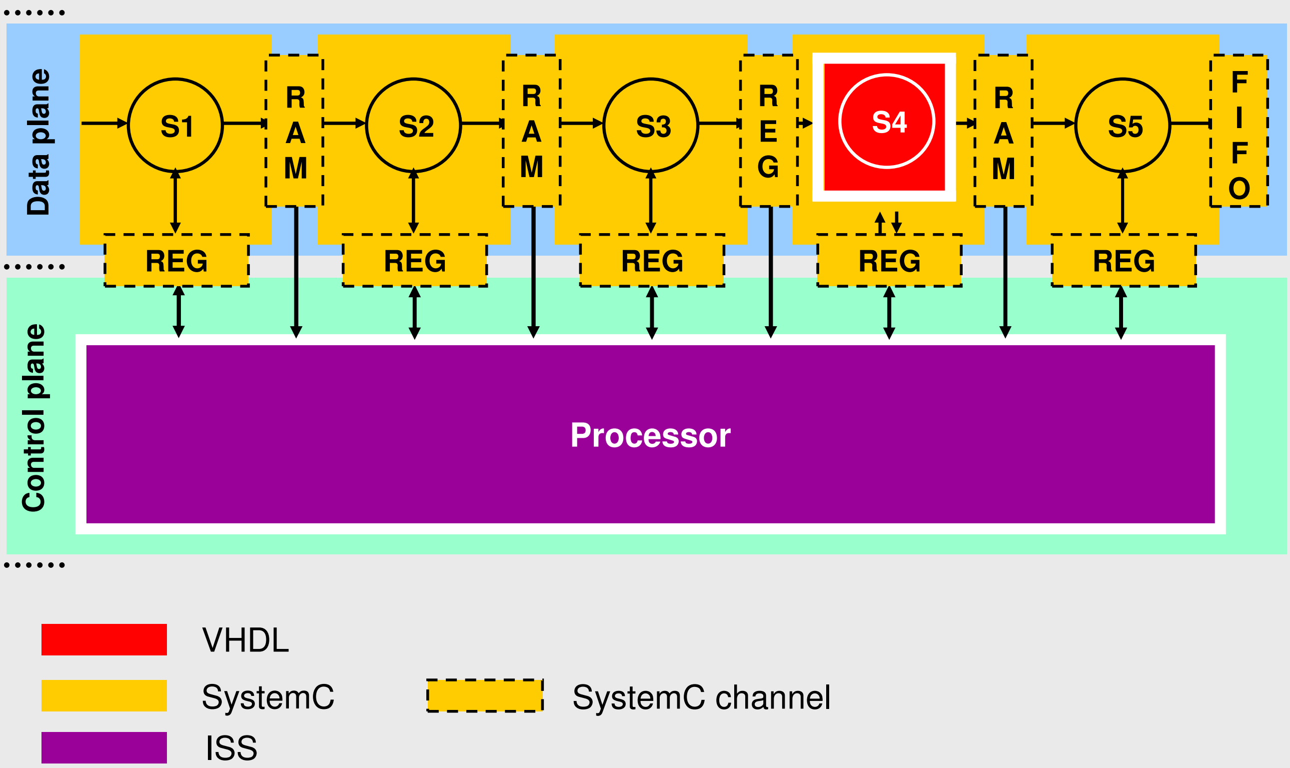
Executable System Models: *SystemC* ► *Implementation*



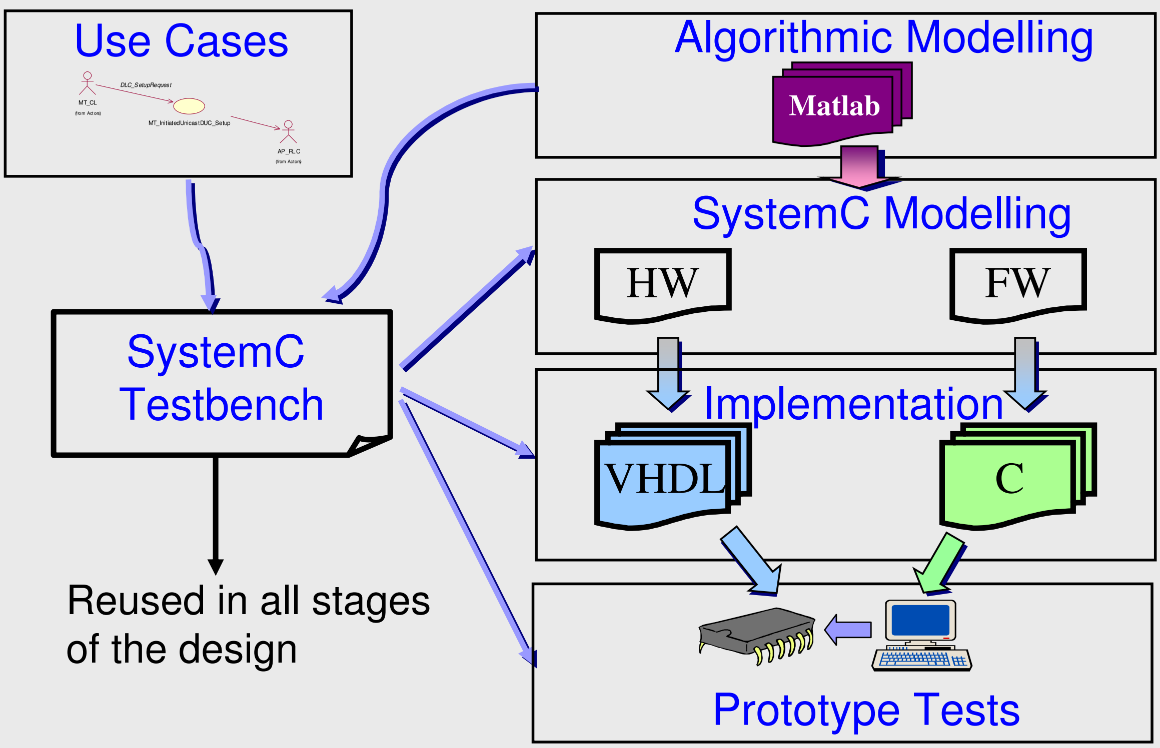
Executable System Models: *SystemC* ► *Implementation*



Executable System Models: *SystemC ► Implementation*



Test: *vertical testbench reuse*



Outline

- Stating the case
  - Evolution towards SoC
  - Concrete example: the OFDM Wireless LAN Project at AmE
- Methodology overview
  - Process
  - Modelling Strategies
    - Role of SystemC
    - SystemC Modelling
    - Model Refinement and Verification
- Conclusions

Conclusions

- Besides the language, one needs also a Methodology
  - Iterative based design: reducing risks, model based
  - Iteration 0: SystemC model development
    - executable specification as golden reference
    - tool for early (designers/customers) feedback and design decisions
  - Next iterations: design implementation, test and qualification
    - SystemC model as reference & testbench reuse
- Experience from using SystemC and executable models: a.o.
  - SystemC easy to learn and to use, extendable
  - Executable spec. ⇏ no need for documentation
  - SystemC 2x bridges the world of SW and HW
    - same language base for both
    - simulation speed
      (abstract communication, TF level)

Conclusions

- Bottom line,
  - **Improved product quality**
    - meeting requirements
    - early bug-detection
  - **Improved scheduling accuracy**
    - much sounder basis for confident prediction of product release date than the traditional waterfall development
  - **Improved inter-disciplinary (system, hw, sw) cooperation**
    - cross-fertilisation of ideas
    - better communications
  - **Improved confidence in the team**
    - early and regular demonstrations of progress in tangible ways