



**Public review
AMS Draft 1 Standard**

**Martin Barnasconi
AMS Working Group Chair
March 23, 2009**

Overview

- Motivation
- AMS Working Group
- AMS Draft 1 Standard kit contents
- Use cases
- Modeling formalisms and language constructs
- Code examples

Motivation

Why having AMS extensions for SystemC?

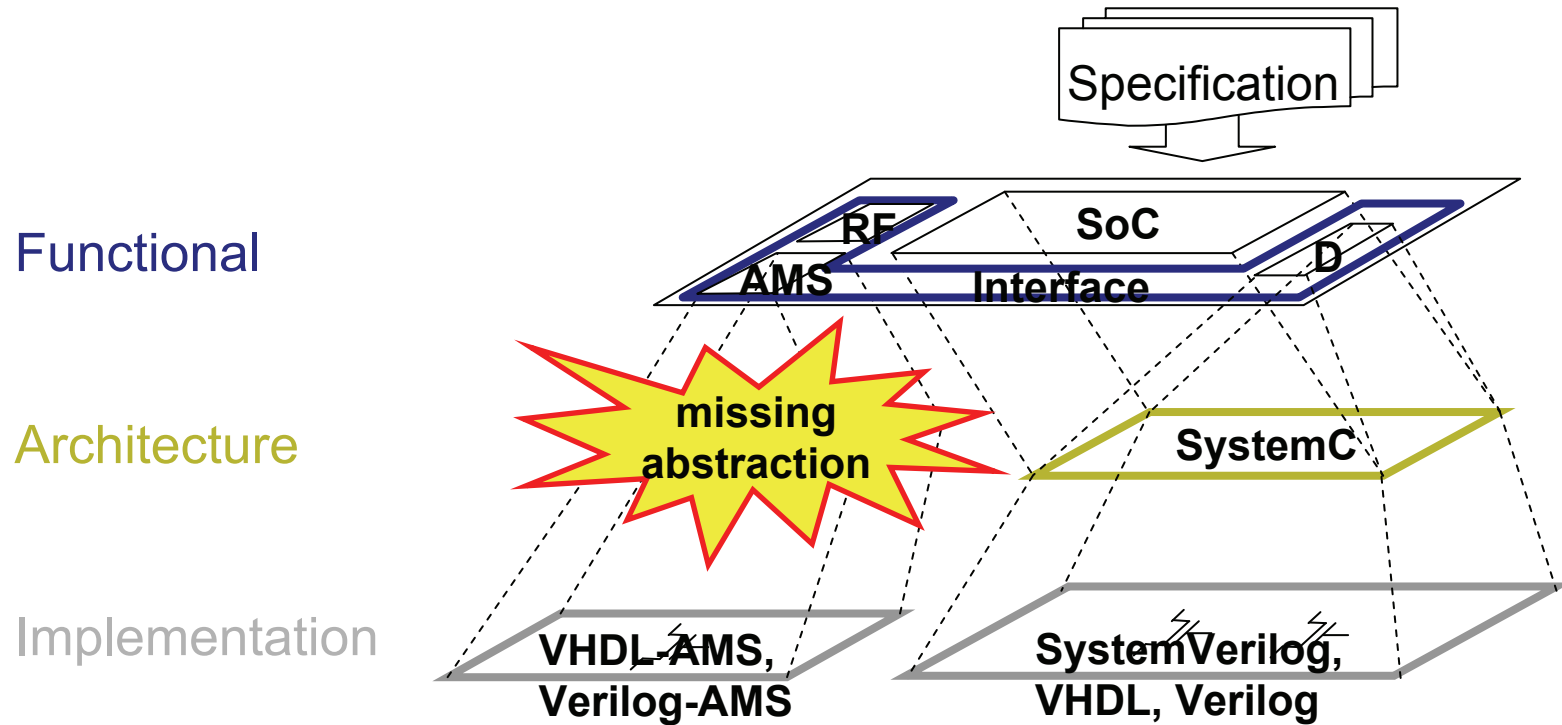
- **Missing is**

- An agreed system modeling language and methodology to design embedded AMS systems
- An open modeling and programming interface between AMS and digital HW/SW system descriptions
- A platform that facilitates AMS model exchange and reuse of intellectual property (IP)
- An architecture design tool for AMS system-level design and verification

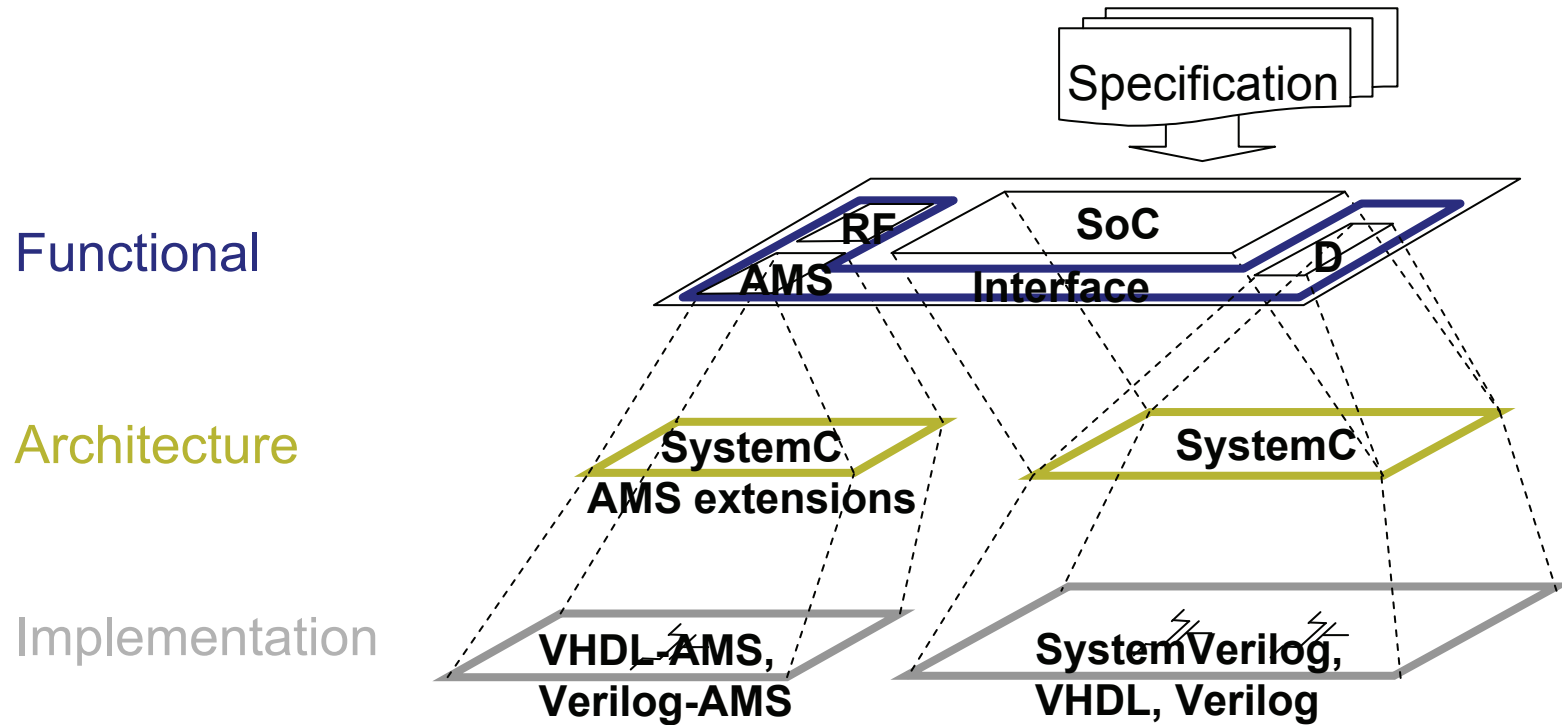
- **It's time to standardize *AMS extensions* for SystemC !**

- Open SystemC Initiative will drive standardization, deployment and support of the SystemC AMS extensions
- Targeting an open source standard for system-level design for Embedded AMS systems

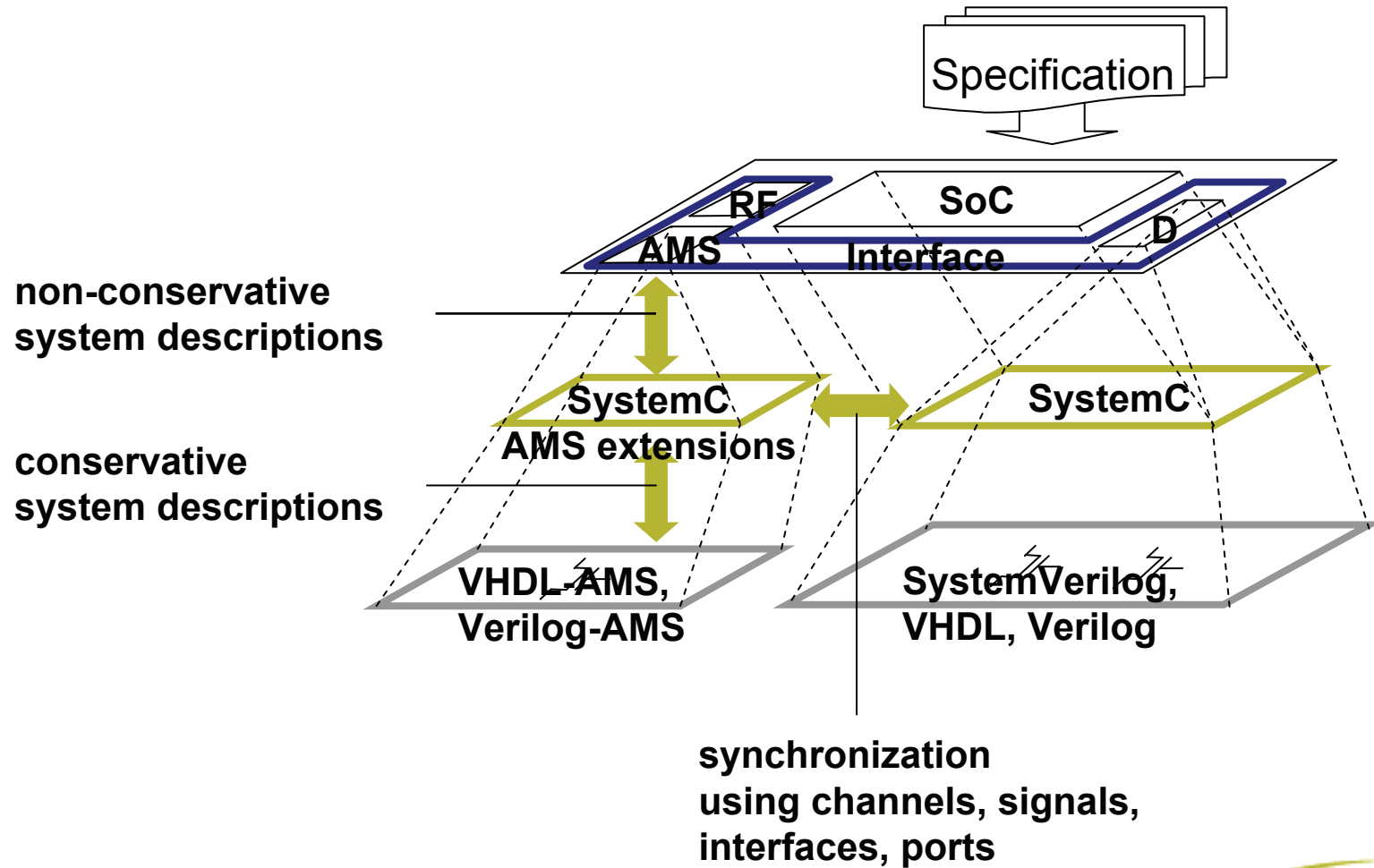
Positioning SystemC AMS Extensions



Positioning SystemC AMS Extensions



Positioning SystemC AMS Extensions



The SystemC AMS extensions

■ Objectives

- Unified and standardized modeling approach to design Embedded AMS systems
- AMS model descriptions supporting a design refinement methodology, from functional specification to implementation
- AMS constructs and semantics in a SystemC compatible class library implemented in C++
- Providing an interoperable modeling platform for development and exchange of AMS intellectual property
- Creating a robust foundation for development of system-level tools

AMS Working Group

OSCI AMS Working Group Roster



- **Steady growth in AMS WG: 53 individuals from 19 organizations**
 - strong drive from semiconductor industry
 - full support of universities and research institutes
 - growing interest and participation of EDA/ESL vendors
- **Chair: Martin Barnasconi, NXP Semiconductors**
Vice chair: Christoph Grimm, Vienna University of Technology

AMS Working Group scope

- **Embedded analog/
mixed-signal systems**
 - Heterogeneous systems including analog, mixed-signal, RF and digital HW/SW

- **Application domains**
 - **Wireless**
 - **Wired**
 - **Automotive**
 - **Imaging sensors**

- **Use cases**

- Virtual prototyping for SW development
- Creating reference models for functional verification
- Architecture exploration, definition and algorithm validation

End Product Markets	2003	2004	2005	2006	2007
Microprocessor/DSP	18.9%	16.0%	13.1%	10.5%	14.7%
Computer, Peripheral	22.9%	21.6%	18.5%	24.2%	19.0%
Wired Network	11.2%	5.2%	5.8%	4.8%	5.2%
Wireless Network	13.1%	10.4%	13.1%	7.3%	6.9%
Multimedia	25.6%	34.2%	33.8%	37.9%	31.9%
Automotive	1.9%	3.0%	3.8%	4.0%	4.3%
Others	6.4%	9.7%	11.9%	11.3%	18.1%

source: SystemC Trends report, April 2007

focus of AMS WG



Planning and timing

- **Phase 1: Requirements study (2006-2007)**
 - Agreement of functional requirement specification
 - Architecture and code review existing solutions
- **Phase 2: Definition and Proposal (2007-2008)**
 - Whitepaper introducing SystemC AMS Extensions
 - SystemC AMS draft 1 Standard
- **Phase 3: Feedback and Standardization (2008-2009)**
 - **Public review of SystemC AMS Language Reference Manual**
 - Promote SystemC AMS extensions as OSCI standard
- **AMS WG status and drafts will be announced via www.systemc.org**



AMS Draft 1 Standard

AMS Draft 1 standard – kit contents

- **Draft Standard SystemC AMS extensions Language Reference Manual**
- **Requirements specification for SystemC AMS extensions**
- **Whitepaper “An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS extensions”**
- **Code example SystemC AMS extensions**

AMS Language Reference Manual

- **This document defines the Draft 1 Standard for the SystemC AMS extensions**
- **Contents**
 - Overview
 - Terminology and conventions
 - Core language definitions
 - Predefined models of computation
 - Predefined analyses
 - Utility definitions
 - Introduction to the SystemC AMS extensions (Informative)
 - Glossary (Informative)

Requirements specification

- Document describing the motivation, AMS applications and use cases for analog/mixed-signal modeling and defines the detailed implementation requirements for the AMS Standard
- Contents
 - Motivation, applications and use cases
 - Implementation requirements
 - Comparison table requirements specification and AMS Draft 1 Standard

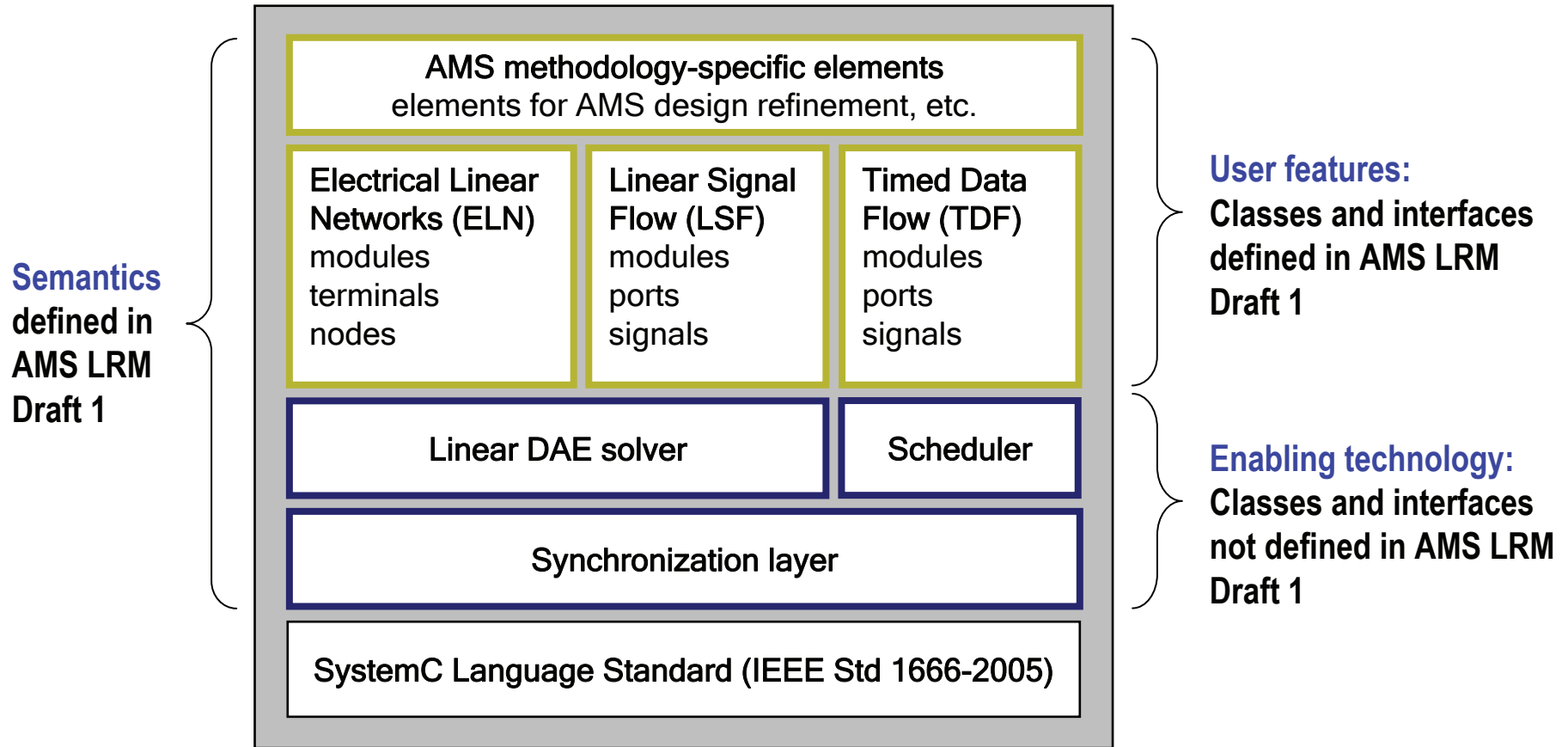
Whitepaper

- **An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS Extensions**
- **Contents**
 - Introduction and motivation
 - Use cases and requirements
 - Language constructs
 - Modeling and methodology example

Code example

- **Example showing the basic capabilities of the SystemC AMS extensions**
- **Constructs shown**
 - User-defined TDF primitive modules
 - Usage of TDF signals and ports
 - Usage of LSF primitive modules, signals and ports
 - Usage ELN primitive modules, nodes and terminals
 - Usage of converter modules and ports between different models of computation
 - Embedded linear dynamic equations
- **Illustrative example only – no proof-of-concept simulator available yet**

SystemC AMS extensions LRM Draft 1

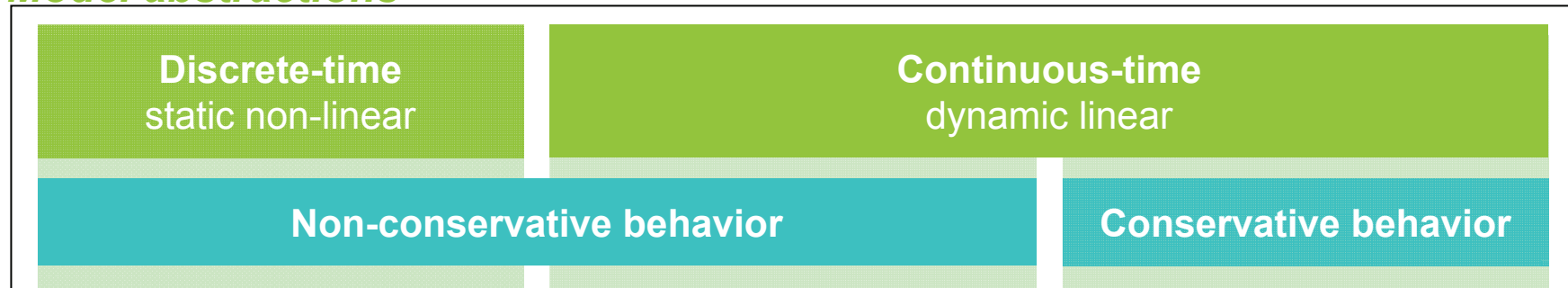


Model abstraction and formalisms

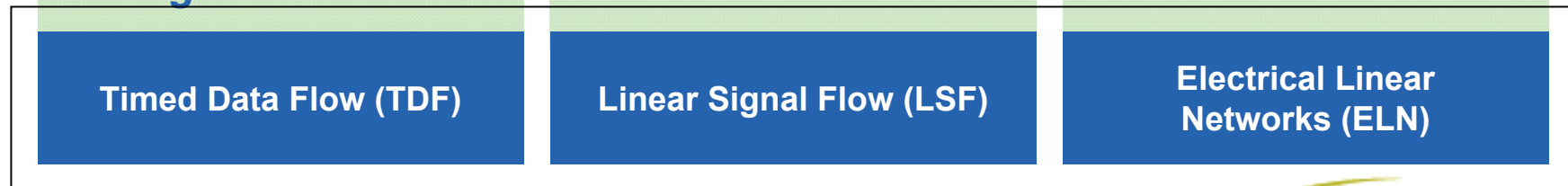
Use cases



Model abstractions

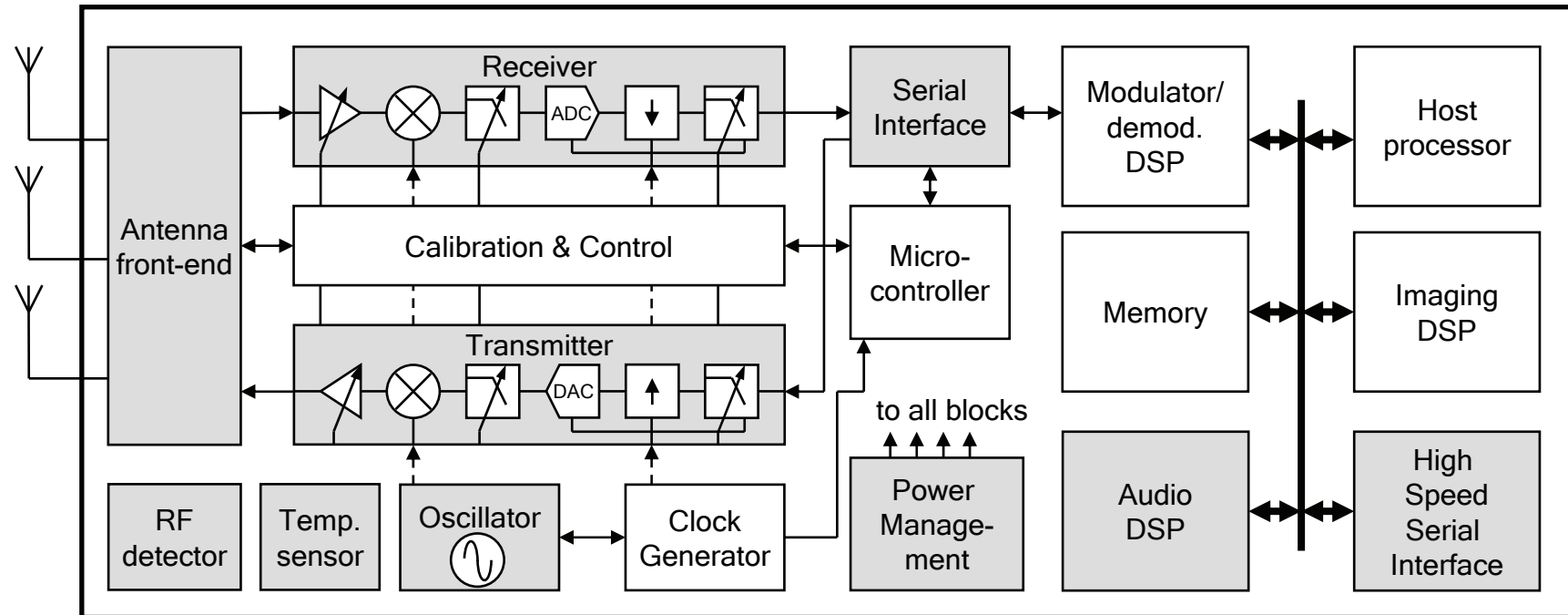


Modeling formalism



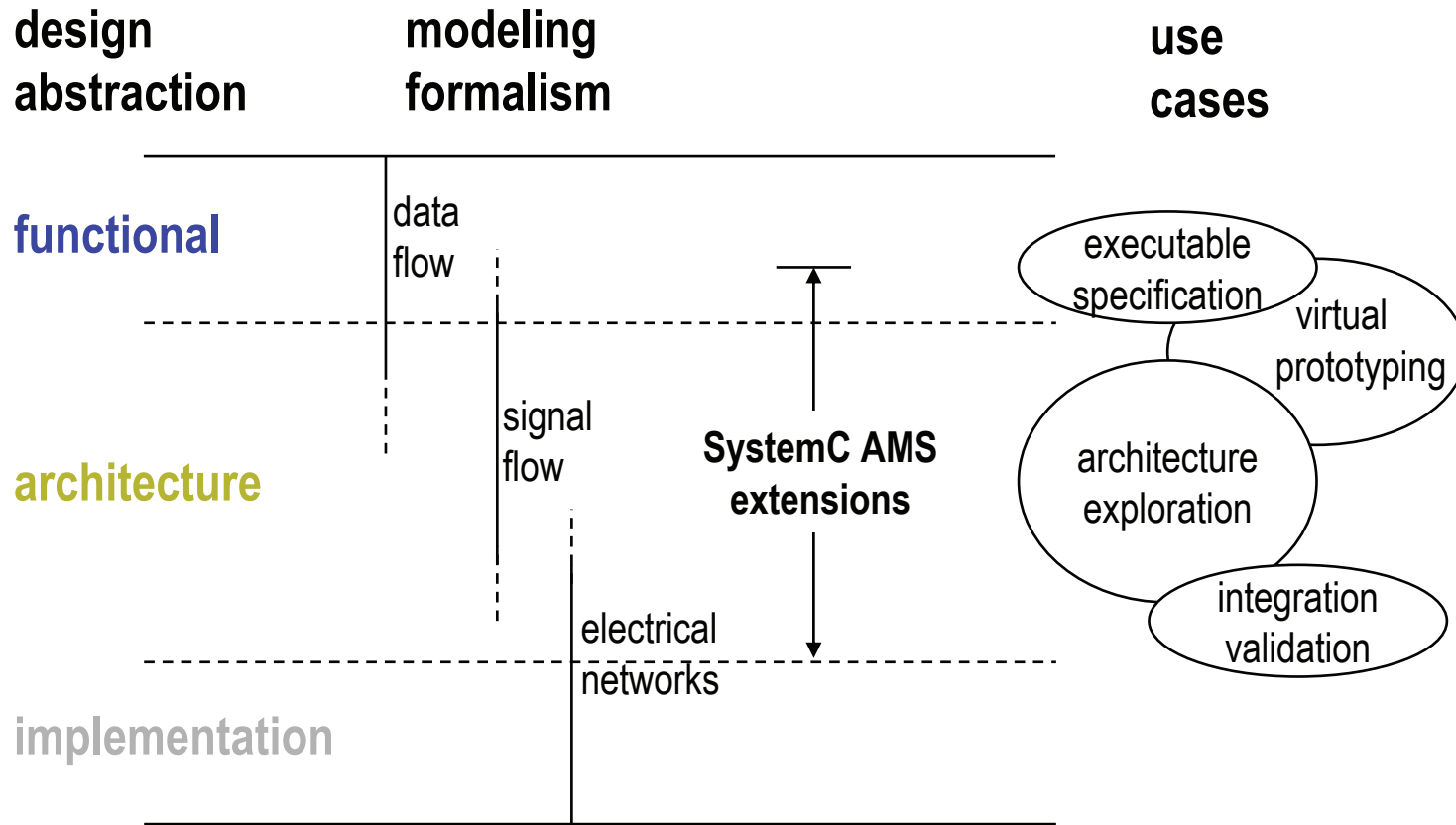
Use cases

Embedded AMS systems – a closer look...



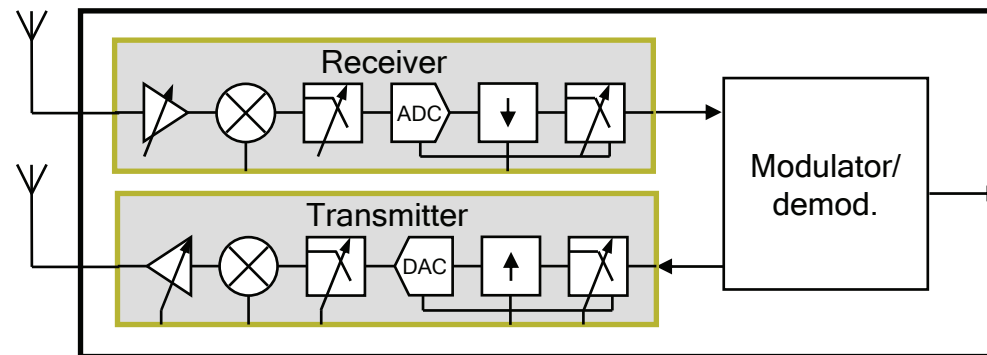
- **Tight interaction between digital HW/SW and AMS sub-systems**
 - Control path: more and more HW/SW calibration and control of analog blocks
 - Signal path: ISO/OSI protocol (SW) stack – modeling including PHY layer

Modeling formalisms and use cases



Use case: executable specification

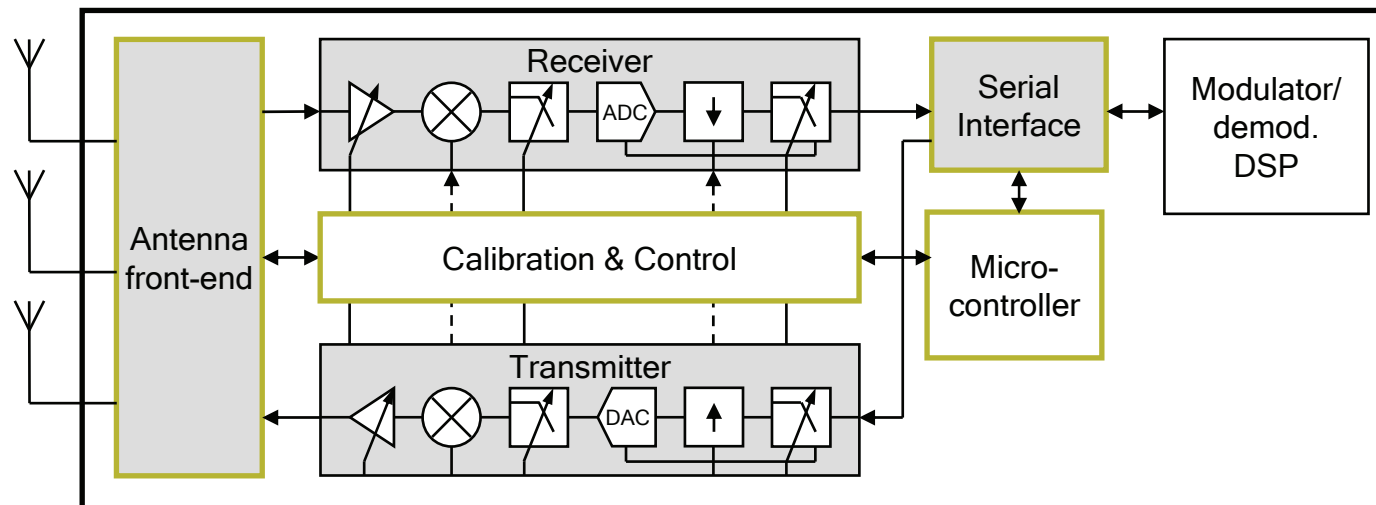
- **Objective: verify the system specification by creating an executable description of the system using simulation**
- **Highest level of design abstraction applied, using functional models and signal processing algorithms**
- **Timed Data Flow and Linear Signal Flow modeling formalisms used**



Transceiver based on functional models

Use case: architecture exploration

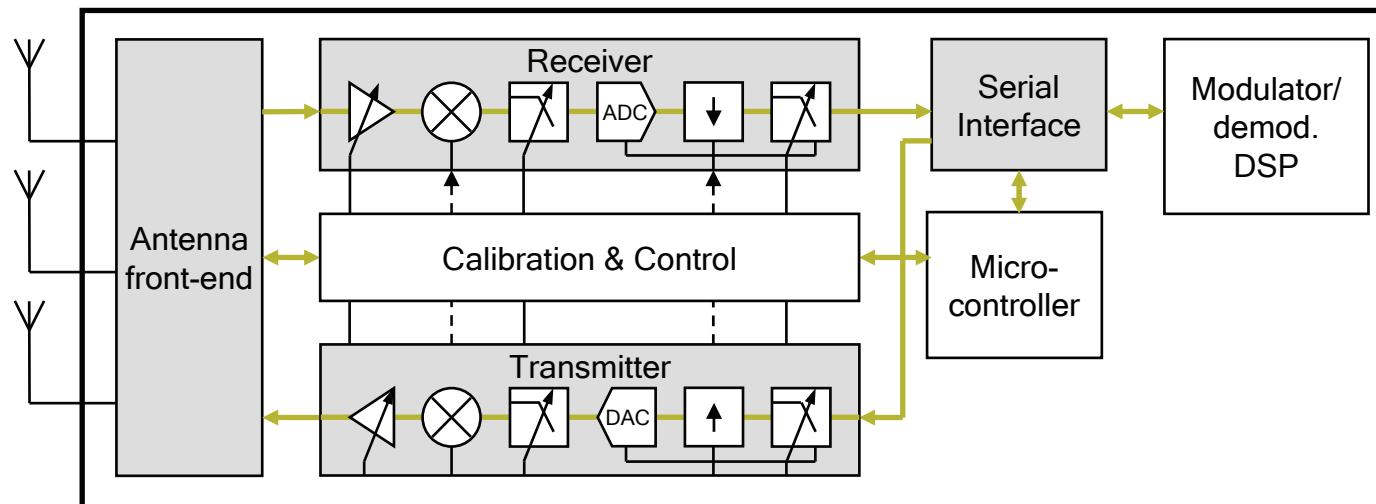
- **Objective: determine, evaluate and dimension the key properties of the system architecture**
- **Two phases**
 - Refinement of executable specification by adding non-ideal properties
 - Introduce interfaces components and architectural elements to match the final implementation



Transceiver including architecture elements (MCU, serial interface, ...)

Use case: integration validation

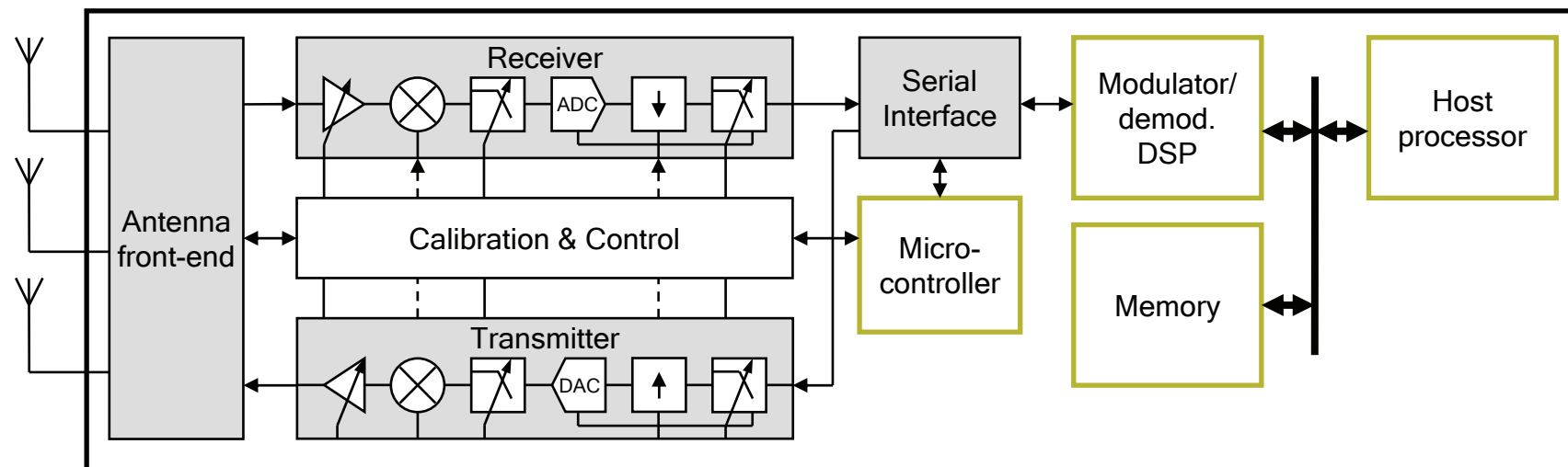
- **Objective: accurate modeling of interfaces of all subsystems**
 - analog circuits/subsystems: introduce electrical nodes
 - digital circuits/subsystems: bit/cycle accurate signals and busses
 - HW/SW subsystem (CPU, MCU): TLM interfaces



Transceiver subsystems modeled with accurate interfaces

Use case: virtual prototyping

- **Objective: SW development using a high-level untimed or timed model that represents the AMS sub-system**
- **Interoperability with SystemC TLM extensions expected**
- **Modeling formalism used:
Timed Data Flow, incorporating untimed models if needed**



Transceiver subsystem connected to digital virtual platform

Modeling formalisms and language constructs

SystemC AMS extensions - concept

- **AMS modeling formalisms based on known *models of computation* (MoC)**
 - Data flow
 - Signal flow
 - Electrical networks
- **AMS models of computation are not based on communication / synchronization of processes**
 - instead, AMS descriptions represent an *equation system*
- **An AMS primitive module represents a set of equation, which has to be contributed to the overall equation system**
- **An AMS interface / channel represents a node in a conservative system or a variable in a non-conservative system**

SystemC AMS extensions – elements ^{1/2}

- **Timed Data Flow - efficient simulation of discrete-time behavior**
 - Data flow simulation accelerated using static scheduling
 - Schedule is activated in discrete time steps, introducing timed semantics
 - Support of static non-linear behavior
- **Linear Signal Flow - simulation of continuous-time behavior**
 - Differential and Algebraic Equations solved numerically at appropriate time steps
 - Primitive modules defined for adders, integrators, differentiators, transfer functions, etc.
- **Electrical Linear Networks - simulation of network primitives**
 - Network topology results in equation system which is solved numerically
 - Primitive modules defined for linear components (e.g. resistors, capacitors) and switches

SystemC AMS extensions – elements ^{2/2}

- **AMS methodology-specific elements**
 - Unified design refinement methodology to support different use cases
 - Time domain simulation and Small-signal frequency-domain AC and noise analysis
- **User-defined AMS extensions**
 - Additional simulators and solvers can be linked in a C++ manner
 - Using the synchronization layer for the communication with SystemC
- **Synchronization with SystemC**
 - Fixed time-step synchronization with SystemC
 - Predefined converter ports and converter modules/primitives to synchronize between TDF, LSF and/or ELN and SystemC
- **Each model of computation has its own namespace**
 - Timed Data Flow: sca_tdf
 - Linear Signal Flow: sca_lsf
 - Electrical Linear Networks: sca_eln

SystemC AMS extensions - module types

- **AMS modules are derived from `sc_module`**
 - note: not all `sc_module` member functions can be used
- **AMS modules are always primitive modules**
 - an AMS module can not contain other modules and/or channels
- **Hierarchical descriptions still use `sc_module` (or `SC_MODULE` macro)**
- **Depending on the MoC, AMS modules are pre-defined or user-defined**
- **Language constructs**
 - `sca_MoC::sca_module` (or `SCA_MoC_MODULE` macro)
 - e.g. `sca_tdf::sca_module` (or `SCA_TDF_MODULE` macro)

SystemC AMS extensions - channel types

- **AMS channels are derived from `sc_interface`**
- **AMS channels for Time Data Flow and Linear Signal Flow**
 - based on directed connection
 - used for non-conservative AMS model of computation
 - Language constructs
 - ◆ `sca_MoC::sca_signal`
 - ◆ e.g. `sca_lsf::sca_signal`, `sca_tdf::sca_signal<T>`
- **AMS channels for Electrical Linear Networks**
 - conservative, non-directed connection
 - characterized by an across (voltage) and through (current) value
 - Language constructs
 - ◆ `sca_MoC::sca_node` / `sca_MoC::sca_node_ref`
 - ◆ e.g. `sca_eln::sca_node`, `sca_eln::sca_node_ref`

TDF language constructs

■ Predefined classes

- sca_tdf::sca_module
- sca_tdf::sca_signal_in_if
- sca_tdf::sca_signal_out_if
- sca_tdf::sca_signal
- sca_tdf::sca_in
- sca_tdf::sca_out
- sca_tdf::sc_core::sc_in
(sca_tdf::sc_in)
- sca_tdf::sc_core::sc_out
(sca_tdf::sc_out)

■ (some) member functions

- set_delay, get_delay
- set_rate, get, rate
- set_timestep, get_timestep
- set_timeoffset
- read, write
- kind
- set_attributes
- initialize
- processing
- ac_processing
- ...

Example: TDF language constructs

```
SCA_TDF_MODULE(mytdfmodel)           // create your own TDF primitive module
{
  sca_tdf::sca_in<double> in1, in2; // TDF input ports
  sca_tdf::sca_out<double> out;     // TDF output port

  void set_attributes()
  {
    // placeholder for simulation attributes
    // e.g. time step between module activations
  }

  void initialize()
  {
    // put your initial values here
  }

  void processing()
  {
    // put your signal processing or algorithm here
  }

  SCA_CTOR(mytdfmodel) {}
};
```

LSF language constructs

■ Predefined classes

- sca_lsf::sca_in
- sca_lsf::sca_out
- sca_lsf::sca_signal
- sca_lsf::sca_add
- sca_lsf::sca_sub
- sca_lsf::sca_gain
- sca_lsf::sca_dot
- sca_lsf::sca_integ
- sca_lsf::sca_delay
- sca_lsf::sca_source
- sca_lsf::sca_ltf_nd
- sca_lsf::sca_ltf_zp

■ Predefined classes (cont.)

- sca_lsf::sca_ss
- sca_lsf::sca_tdf::sca_source
- sca_lsf::sca_tdf::sca_gain
- sca_lsf::sca_tdf::sca_mux
- sca_lsf::sca_tdf::sca_demux
- sca_lsf::sca_tdf::sca_sink
- sca_lsf::sc_core::sca_source
- sca_lsf::sc_core::sca_gain
- sca_lsf::sc_core::sca_mux
- sca_lsf::sc_core::sca_demux
- sca_lsf::sc_core::sca_sink
- ...

Example: LSF language constructs

```
SC_MODULE(mylsfmodel)          // create a model using LSF primitive modules
{
  sca_lsf::sca_in in;          // LSF input port
  sca_lsf::sca_out out;       // LSF output port

  sca_lsf::sca_signal sig;    // LSF signal

  lp_filter_lsf(sc_module_name, double fc=1.0e3) // Constructor with
  {                                     // parameters

    sub1 = new sca_lsf::sca_sub("sub1");      // instantiate predefined
    sub1->x1(in);                             // primitives here
    sub1->x2(sig);
    sub1->y(out);

    dot1 = new sca_lsf::sca_dot("dot1", 1.0/(2.0*M_PI*fc) );
    dot1->x(out);
    dot1->y(sig);
  }
};
```

ELN language constructs

■ Predefined classes

- sca_eIn::sca_terminal
- sca_eIn::sca_node
- sca_eIn::sca_node_ref
- sca_eIn::sca_r
- sca_eIn::sca_l
- sca_eIn::sca_c
- sca_eIn::sca_vcvs
- sca_eIn::sca_vccs
- sca_eIn::sca_ccvs
- sca_eIn::sca_cccs
- sca_eIn::sca_nullor
- sca_eIn::sca_gyrator
- ...

■ Predefined classes (cont.)

- sca_eIn::sca_vsource
- sca_eIn::sca_vsink
- sca_eIn::sc_tdf::sca_vsource
- sca_eIn::sca_tdf::sca_istource
- sca_eIn::sc_core::sca_vsource
- sca_eIn::sc_core::sca_istource
- sca_eIn::sca_tdf::sca_r
- sca_eIn::sca_tdf::sca_l
- sca_eIn::sca_tdf::sca_c
- sca_eIn::sc_core::sca_r
- sca_eIn::sc_core::sca_l
- sca_eIn::sc_core::sca_c
- ...

Example: ELN language constructs

```
SC_MODULE(myelnmodel)           // model using ELN primitive modules
{
  sca_e1n::sca_terminal in, out; // ELN terminal (input and output)

  sca_e1n::sca_node    n1;      // ELN node
  sca_e1n::sca_node_ref gnd;    // ELN reference node

  SC_CTOR(lp_filter_e1n)       // standard constructor
  {
    r1 = new sca_e1n::sca_r("r1", 10e3); // instantiate predefined
    r1->p(in);                          // primitive here (resistor)
    r1->n(out);

    c1 = new sca_e1n::sca_c("c1", 100e-6);
    c1->p(out);
    c1->n(gnd);
  }
};
```

Code example

Code example – top-level (RF front-end)

```
SC_MODULE(frontend)
{
  sca_tdf::sca_in<double> rf, loc_osc;
  sca_tdf::sca_out<double> if_out;
  sc_core::sc_in<sc_dt::sc_bv<3> > ctrl_config;
```

SC_MODULE used for hierarchical structure

```
sca_tdf::sca_signal<double> if_sig;
sc_core::sc_signal<double> ctrl_gain;
```

usage of different signals

```
mixer* mixer1;
lp_filter_eln* lpf1;
agc_ctrl* ctrl1;
```

```
SC_CTOR(frontend) {
```

```
  mixer1 = new mixer("mixer1"); // TDF module
  mixer1->rf_in(rf);
  mixer1->lo_in(loc_osc);
  mixer1->if_out(if_sig);
```

High-level mixer model
(TDF module)

```
  lpf1 = new lp_filter_eln("lpf1"); // ELN module
  lpf1->in(if_sig);
  lpf1->out(if_out);
```

LPF close to implementation level
(ELN module)

```
  ctrl1 = new agc_ctrl("ctrl1"); // SystemC module
  ctrl1->out(ctrl_gain);
  ctrl1->config(ctrl_config);
```

easy to combine with normal SystemC modules !

```
};
```

Code example – mixer function in TDF

```
SCA_TDF_MODULE(mixer) // TDF primitive module definition
{
  sca_tdf::sca_in<double> rf_in, lo_in; // TDF in ports
  sca_tdf::sca_out<double> if_out;      // TDF out ports
}
```

**TDF primitive
module:
no hierarchy**

```
void set_attributes()
{
  set_timestep(1.0, SC_US); // time between activations
}
```

**Attributes specify
timed semantics**

```
void processing()
{
  if_out.write( rf_in.read() * lo_in.read() );
}
```

**processing()
function executed at
each activation**

```
SCA_CTOR(mixer) {}
};
```

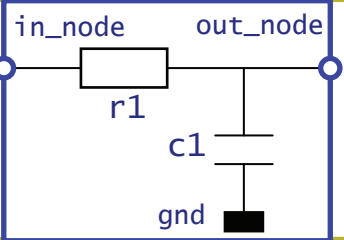
**AMS
constructor**

Code example – Lowpass filter in ELN 1/2

<pre>SC_MODULE(lp_filter_eln) { sca_tdf::sca_in<double> in; sca_tdf::sca_out<double> out;</pre>	SC_MODULE used for hierarchical structure
<pre>sca_eln::sca_node in_node, out_node; // node declarations sca_eln::sca_node_ref gnd; // reference node</pre>	nodes to connect components
<pre>sca_eln::sca_r *r1; // resistor sca_eln::sca_c *c1; // capacitor</pre>	network primitives (components)
<pre>sca_eln::sca_tdf_vsource *v_in; sca_eln::sca_tdf_vsink *v_out;</pre>	primitive converter modules from/to TDF

⋮

Code example – Lowpass filter in ELN 2/2

<pre>SC_CTOR(lp_filter_eln) {</pre>	normal constructor
<pre> v_in = new sca_eln::sca_tdf_vsource("v_in", 1.0); v_in->inp(in); v_in->p(in_node); v_in->n(gnd);</pre>	TDF input is converted to voltage
<pre> r1 = new sca_eln::sca_r("r1", 10e3); // 10kohm resistor r1->p(in_node); r1->n(out_node); c1 = new sca_eln::sca_c("c1", 100e-6); // 100uF capacitor c1->p(out_node); c1->n(gnd);</pre>	
<pre> v_out = new sca_eln::sca_tdf_vsink("v_out", 1.0); v_out->p(out_node); v_out->n(gnd); v_out->outp(out); } };</pre>	output voltage converted to TDF signal

Conclusions

- **Introducing AMS Draft 1 Standard**
 - AMS extensions built on top of SystemC
 - Building a foundation for new AMS system-level design methodologies

- **SystemC AMS extensions**
 - For modeling and simulation of Electrical Linear Networks, Linear Signal Flow, and Timed Data Flow behavior
 - Language constructs to create AMS models at higher levels of abstraction
 - Essential for executable specification, architecture exploration, integration validation and virtual prototyping use cases

- **Elements defined in the AMS Language Reference Manual (LRM) Draft 1, published December 3, 2008**



Thank You