# Introduction to the SystemC AMS DRAFT Standard


| | |
|---|---|
| Christoph Grimm | Vienna University of Technology |
| Martin Barnasconi | NXP Semiconductors |
| Alain Vachoux | Ecole Polytechnique Fédérale de Lausanne |
| **Karsten Einwich** | **Fraunhofer IIS/EAS Dresden** |




Fraunhofer IIS

# The Fraunhofer-Gesellschaft – at a Glance

The Fraunhofer-Gesellschaft undertakes **applied research** of direct utility to private and public enterprise and of wide benefit to society.


- Founded in 1949 and named by Joseph von Fraunhofer (1787-1826) – a scientist (Fraunhofer lines), inventor (methods of making lenses) and businessmen (managing of a glassworks)
- 57 institutes across Germany with a total staff of 15,000
- Total budget €1.4 billion with approx. 60% of income generated from contract research and government-sponsored projects

**Research areas:**

- **Information and Communication Technology**
- Life Sciences
- **Microelectronics**
- Surface Technology and Photonics
- Production
- Materials and Components
- Defense and Security





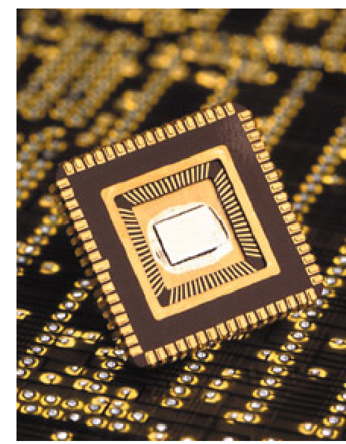
Fraunhofer
IIS

# The Fraunhofer Institute for Integrated Circuits IIS

- Founded in 1985
- Approx. 585 staff
- Budget: approx. € 72 million
- Revenue sources:
  > 80% income from projects
  < 20% public funding
- Locations in Erlangen (Headquarter), Fürth, Nuremberg, **Dresden**, Ilmenau

**Business areas:**

- Audio und multimedia technologies
- Imaging systems
- Digital broadcasting systems
- Embedded communication
- IC design and **design automation**
- Communication networks
- Navigation
- Logistics
- Medical technology
- Optical inspection systems
- X-ray technology





Fraunhofer IIS

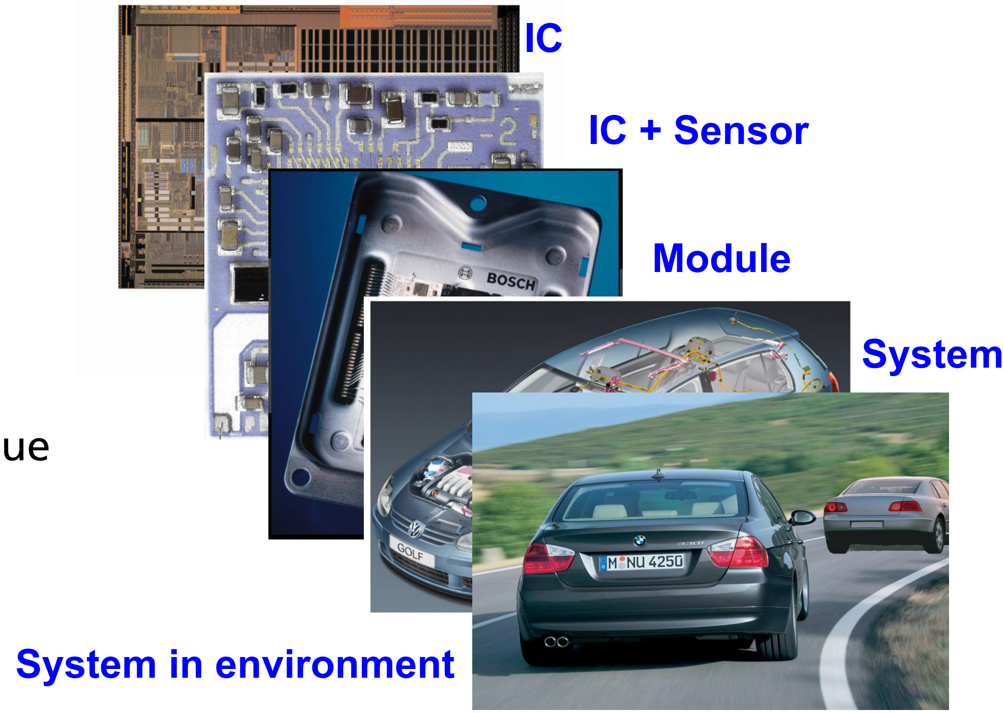
# Design Automation Division EAS, Dresden

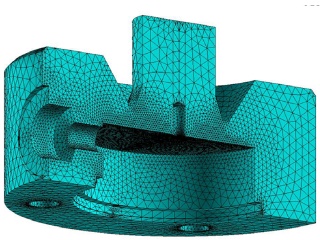
Founded in 1993

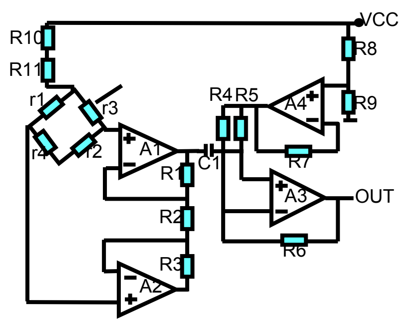
Approx. 80 staff

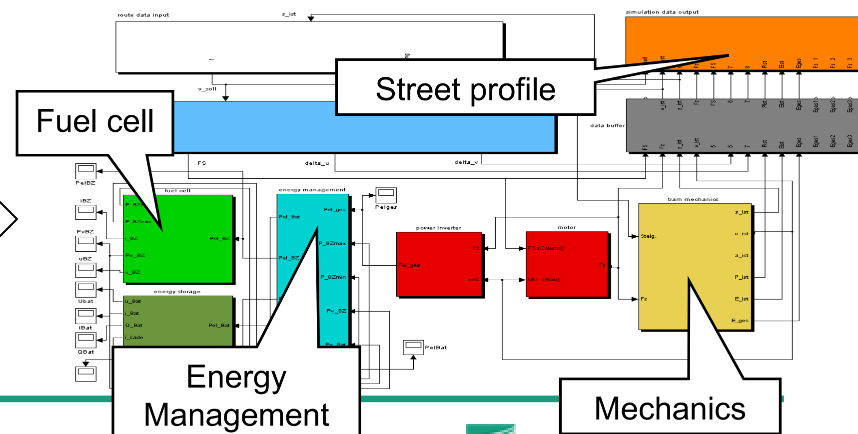
Budget: approx. € 6 million

- Development of methods and tools for computer-aided design of electronic circuits and systems for the complete value chain
- Main areas of work Modeling, simulation, synthesis, optimization, verification and testing










Fraunhofer IIS

# Outline

Introduction

SystemC AMS Basics

Introduction to SystemC AMS DRAFT 1

Application Examples

# SystemC AMS is …

- an extension of **SystemC** which permits modeling of analog mixed-signal behavior

# SystemC is …

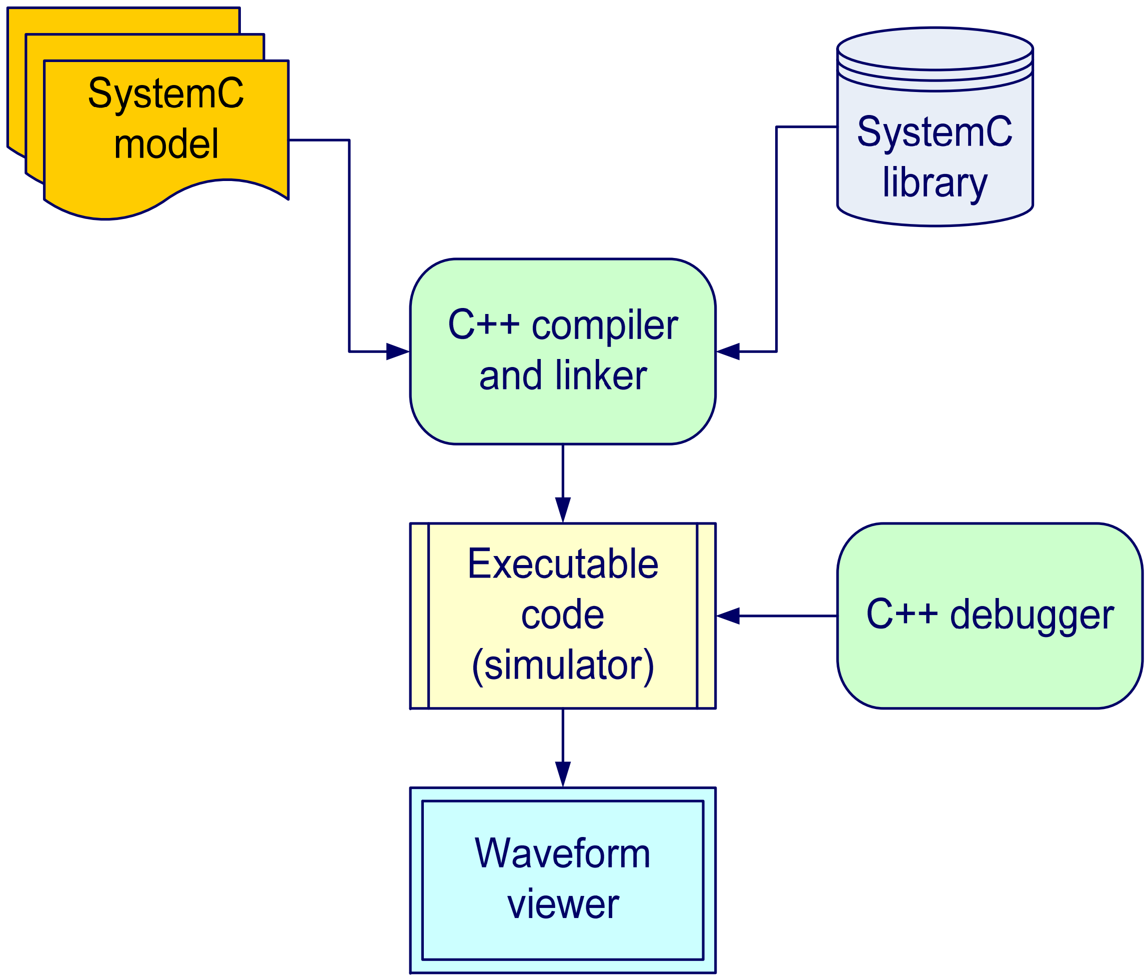
a **definition of C++ language constructs** for the description of complex digital hardware systems on different abstraction levels, using different Models of Computation (MoC)

Definition of classes for modeling:

- Time
- De-composition, Hierarchy
- Concurrency (processes)
- Reactivity
- Signals
- Generic communication channels
- Datatypes

SystemC – models can be simulated using a reference implementation of the C++ class library

# SystemC Use Flow



# SystemC – Yet another VHDL or Verilog ?

- Based on the programming language C++
- Generic Model of Computation
- Layered approach enables extensibility

- Best suited for higher abstraction levels like system level design and architectural level exploration
- Easy integration of software

- Support of special methodologies like TLM

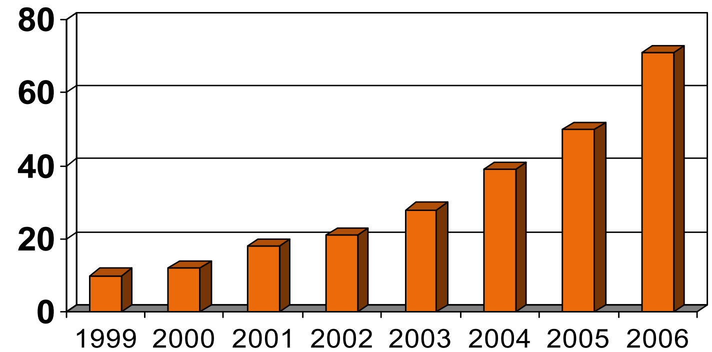
# Why AMS Extensions for SystemC ?

- It's an analog world – analog is how we interact with the real world
- Each digital system is embedded in the analogue world

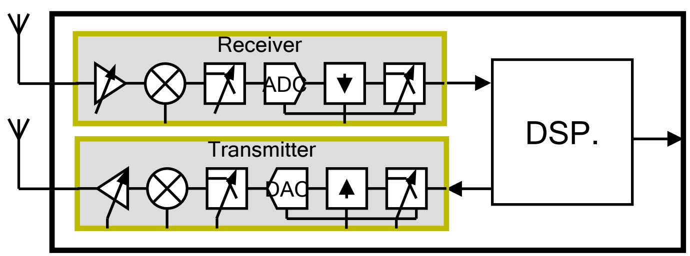
- Analog doesn't scale, the performance of the devices becomes worse, …
- -> Use digital gates to improve analog performance
- –> digital assisted analog
- -> Tight interaction (loops) between analog and digital hard- and software

**Analog's Presence in SoC**

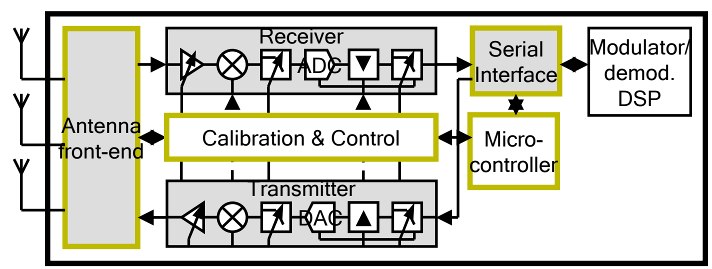
Percents of SoC with analog Elements




Source ITRS 2006




Traditional architecture



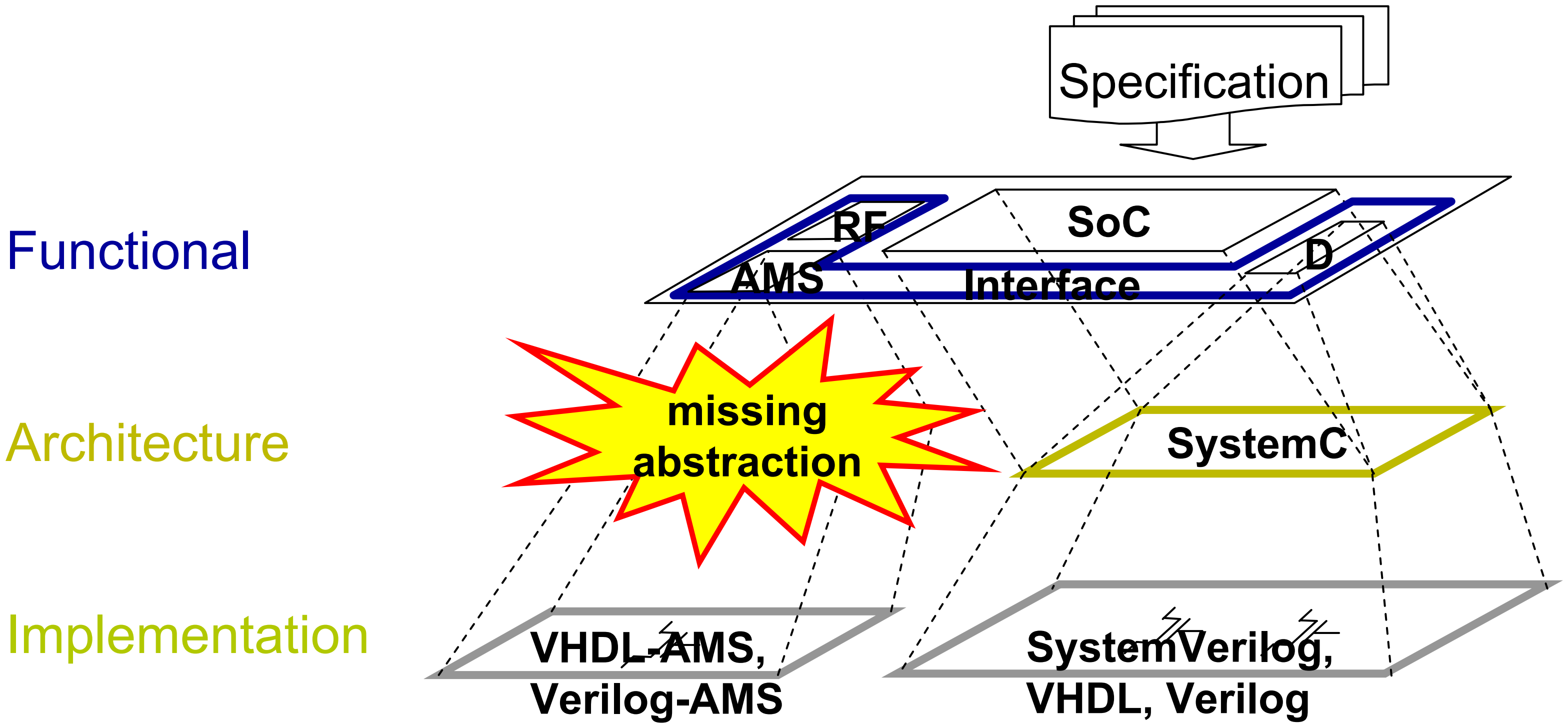

Digital assisted analog





Fraunhofer IIS

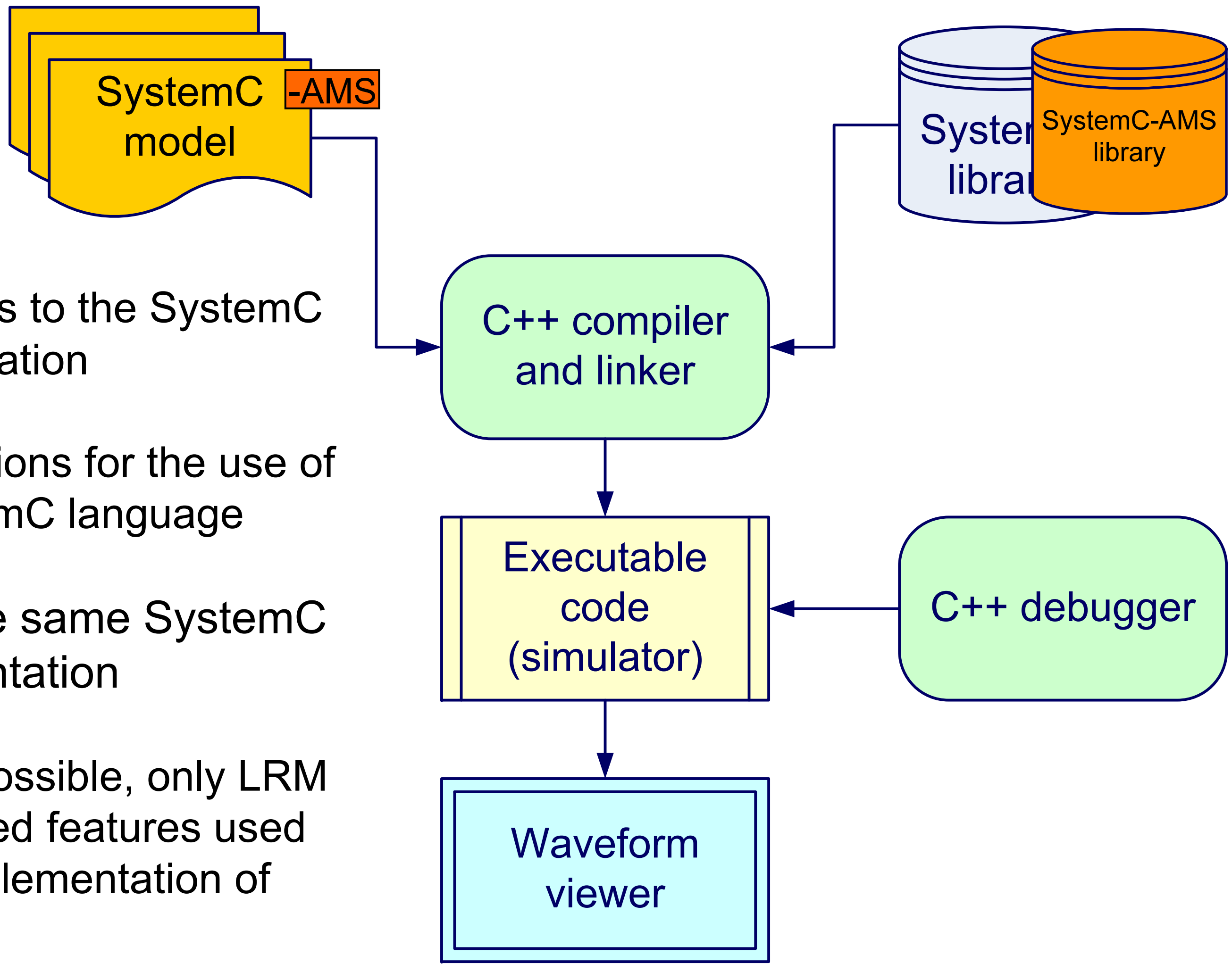
# Why AMS Extensions for SystemC ?




Analog and digital cannot further considered separately

End-to-End platform modeling becomes more and more essential

# SystemC-AMS is an extension of SystemC




- no changes to the SystemC implementation

➡ no restrictions for the use of the SystemC language

➡ use of the same SystemC implementation

- as far as possible, only LRM documented features used for the implementation of the library

# AMS Working Group

# SystemC AMS History

- **2000** Fraunhofer and Infineon developed SystemC extension library mixsigc
- 2000 University Frankfurt and Continental Teves developed AVSL
- 2001 SystemC-AMS study group founded
- Analog extensions from different universities (South Hampton, University Ancona), Dataflow implementation from Shukla
- Ca. 2002 reimplementation of mixsigc to systemc-ams -> publication as prototype of the study group
- **2006** official approval as OSCI AMSWG with Martin Barnasconi from NXP as chair
- 2008 White paper publication at DAC
- December 2008 DRAFT1 Language Reference Manual publicized





Fraunhofer IIS

# 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

we are here

AMS WG status and drafts will be announced via www.systemc.org





Fraunhofer IIS

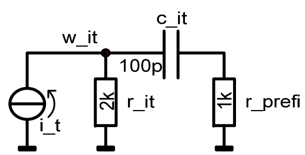
# SystemC AMS Extension Basics

# SystemC Model of Computation

- SystemC has a **generic** Model of Computation (**MoC**)
- This generic MoC is based on the communication and synchronization of parallel processes
  - -> the underlying **system behavior** is **solved "ad hoc"**
- Therefore methods and classes for process registration, events and triggering to events are existing
- Thus MoC's which assume that the system **state changes at discrete time points** can be easily mapped

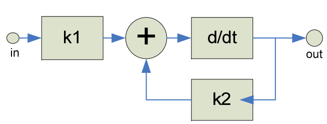
# What's different between analog and digital ?

- Analog equation cannot be solved by the communication and synchronization of processes




$$0 = i\_t + \frac{v(w\_it)}{r\_it} + c\_it \bullet \frac{d(v(w\_it) - v(w\_p))}{dt}$$

$$0 = \frac{v(w\_p)}{r\_prefi} - c\_it \bullet \frac{d(v(w\_it) - v(w\_p))}{dt}$$

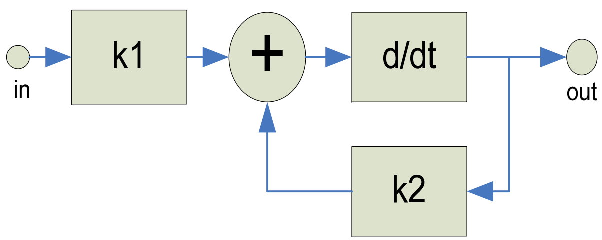


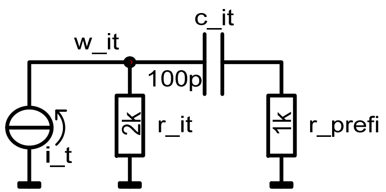

$$out = k1 \bullet \frac{din}{dt} + k2 \bullet out$$

->in general an **equation system must be setup**

- The analog system **state changes continuously**
  - the value between solution points is continuous (linear is a first order approximation only)
  - -> the value of a time point between two solution points can be estimated only after the second point has been calculated (otherwise instable extrapolation)

# Non conservative vs. Conservative




- Abstract representation of analog behavior
- The graph represents a continuous time (implicit) equation (system)

- Represents topological **structure** of the modeled system
- Nodes are characterized by two quantities – the **across** value (e.g. voltage) and the **through** value (e.g. current)
- For electrical systems, **Kirchhoff's laws** applied (KCL, KVL)
- For other **physical domains generalized** versions of Kirchhoffs's laws applied

# Why different analogue MoC's?

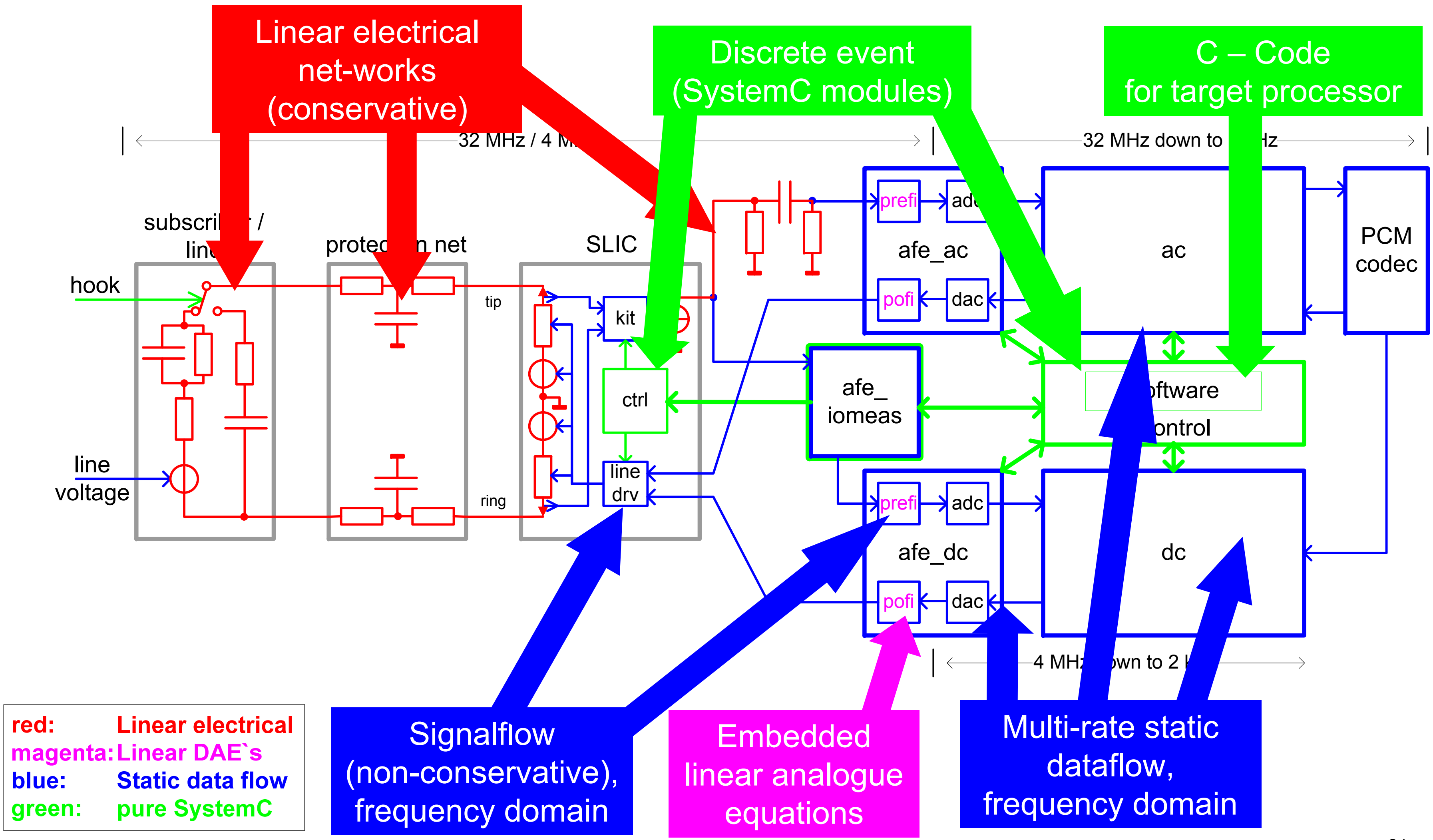
- Modeling on **different abstraction** / accuracy levels yields the possibility to apply specialized algorithms, which are **orders of magnitude faster** than a general approach.
- It is possible to **reduce** the **solvability problem** significantly.
- Due to the **encapsulation** of analogue MoC / solvers SystemC-AMS models are very well **scalable** – very large models can be handled.
- Examples for specialized analogue Models of Computations:
  - Dataflow solver for non-conservative / Signalflow Descriptions
  - Linear Networks / Differential-Algebraic Equation (DAE) systems
  - Non-linear Networks / DAE systems
  - Switched Capacitor Networks (leads to simple algebraic equation)





Fraunhofer
IIS

# Modeling with multiple MoC

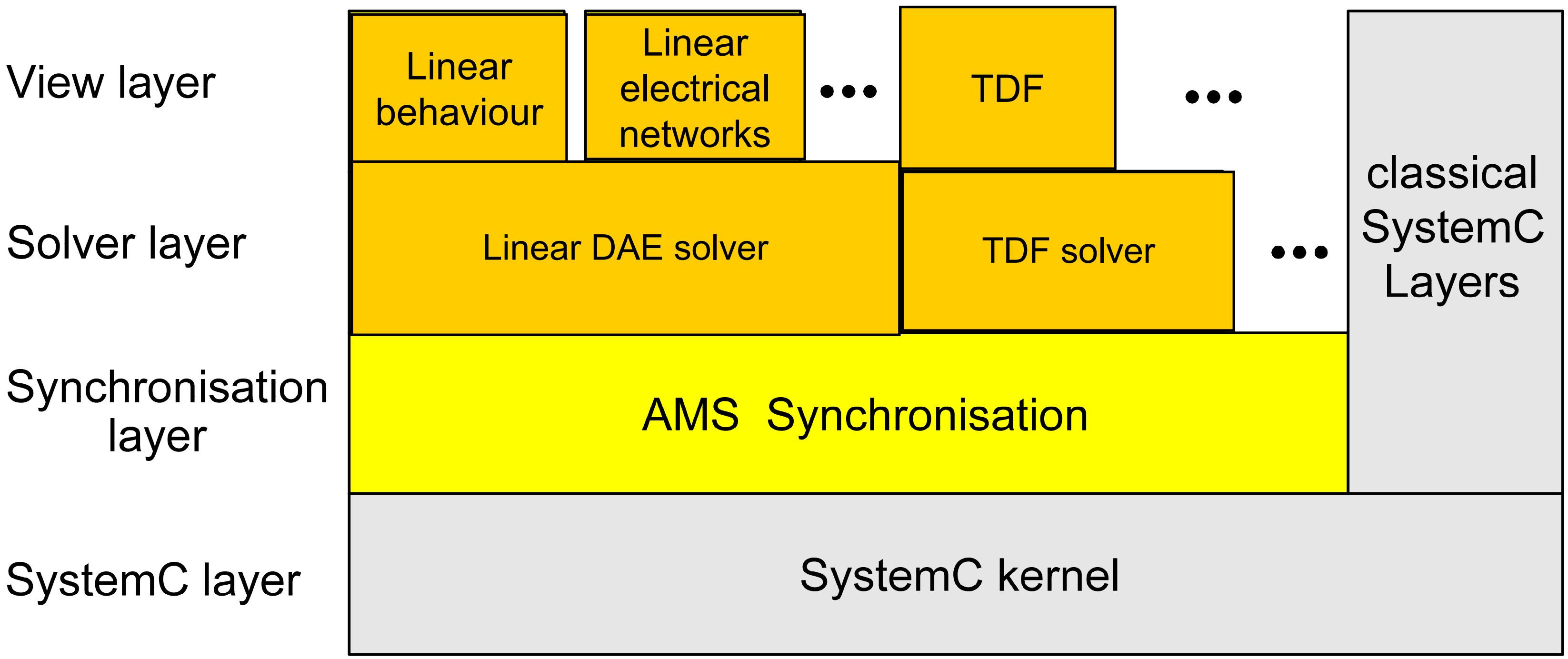







Fraunhofer IIS

# SystemC-AMS Generic Concept







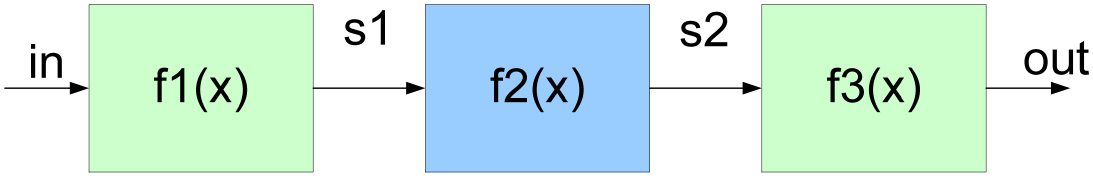

Fraunhofer IIS

# SystemC-AMS MoC

- Public version supports modeling of:
  - Non-conservative systems
  - Multi rate synchronous dataflow (SDF now called TDF)
  - Linear electrical networks
  - Linear behavioral functions (linear transfer function numerator/denumerator and pole zero, state space),
  - Frequency domain simulation
  - Powerful trace functionality

- Experimental extensions available at Fraunhofer: Switched Capacitor solver, Nonlinear DAE solver with de-synchronization

# Dataflow MoC: Modeling non-conservative behavior



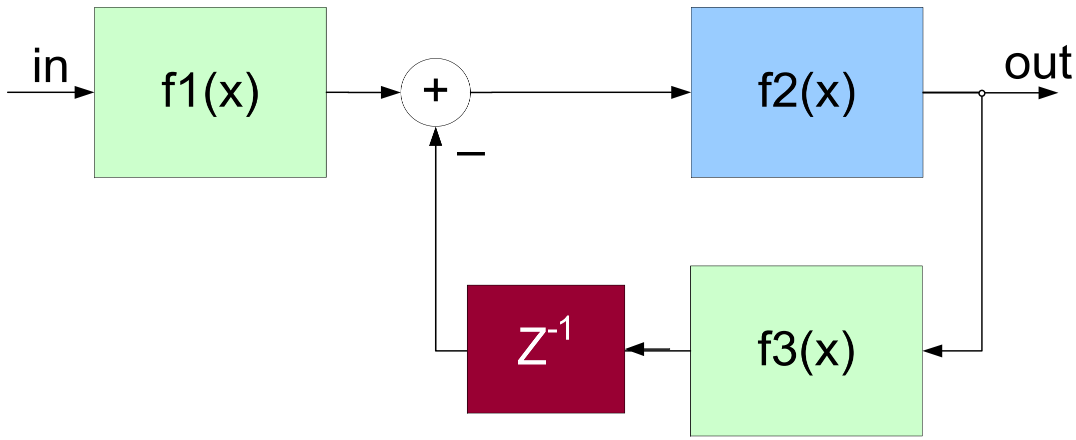

equation system:

s1 = f1(in)
s2 = f2(s1)
out = f3(s2)

out = f3( f2( f1(in) ) )

- Simple firing rule: A module is activated if enough samples are available at its input ports.
- The function of a module is performed by
  - reading from the input ports (thus consuming samples),
  - processing the calculations and
  - writing the results to the output ports.
- For synchronous dataflow (SDF) the numbers of read/written samples are constant for each module activation.
- The scheduling order follows the signalflow direction.
- One drawback is the need of having the equations in an explicit formulation. Thus, only explicit DAE systems can be described by means of the SDF.

# Loops in Synchronous Dataflow Graphs




- Simulating signalflow behaviour by synchronous dataflow MoC with algebraic loops is not possible.

- Thus, at least one delay in the loop is crucial!

$$out = f2(\ f1(in) - f3(out)\ ) \longrightarrow out = f2(\ f1(in) - f3(out)\ z^{-1}\ )$$

# Why Synchronous Dataflow?

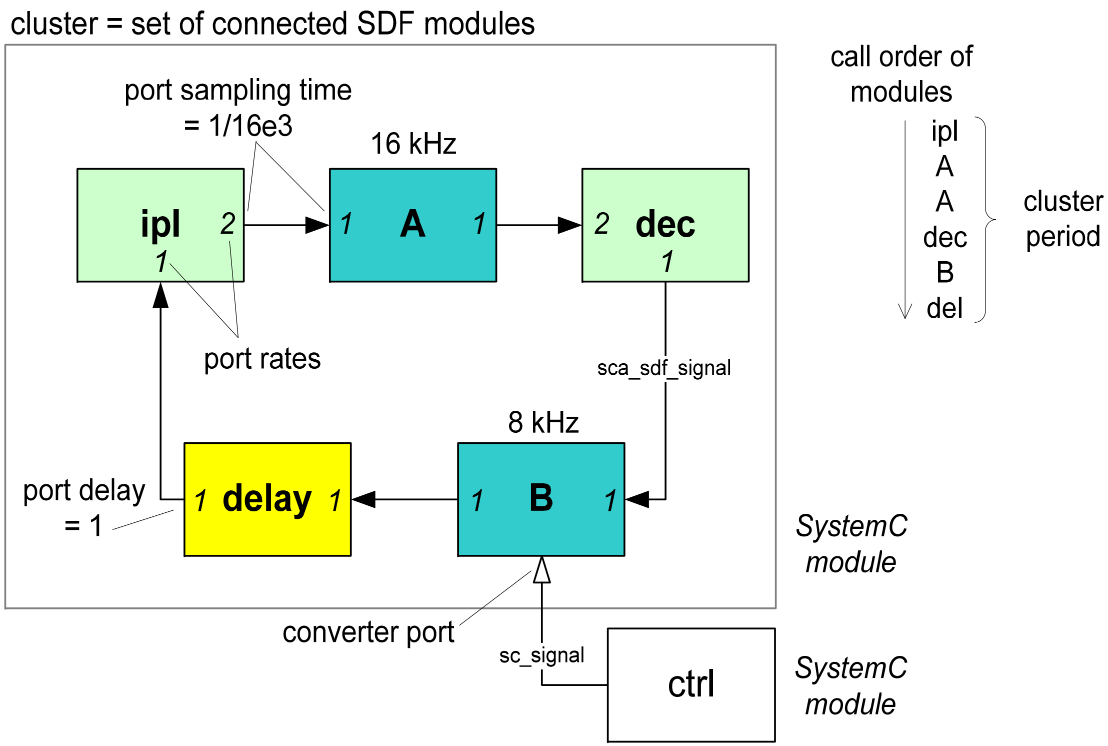
- Due pre-scheduling very **fast execution**
- Well encapsulation -> **no solvability problem**
- No iterations required
- Well adopted for signal processing systems
- Smooth crossover to digital processing domain
- The price: **no algebraic loops** is usually acceptable for system and architectural level modeling

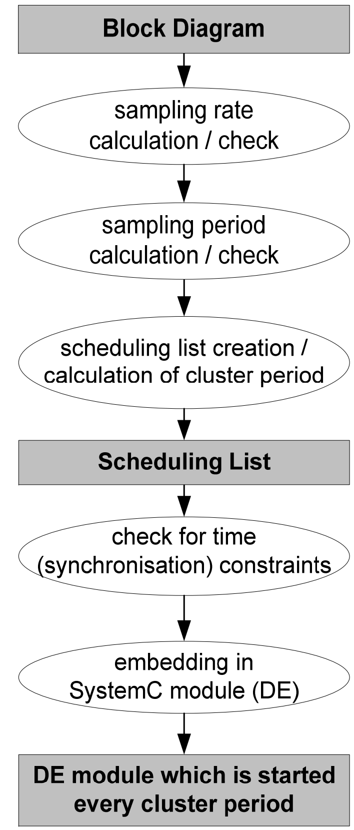




Fraunhofer IIS

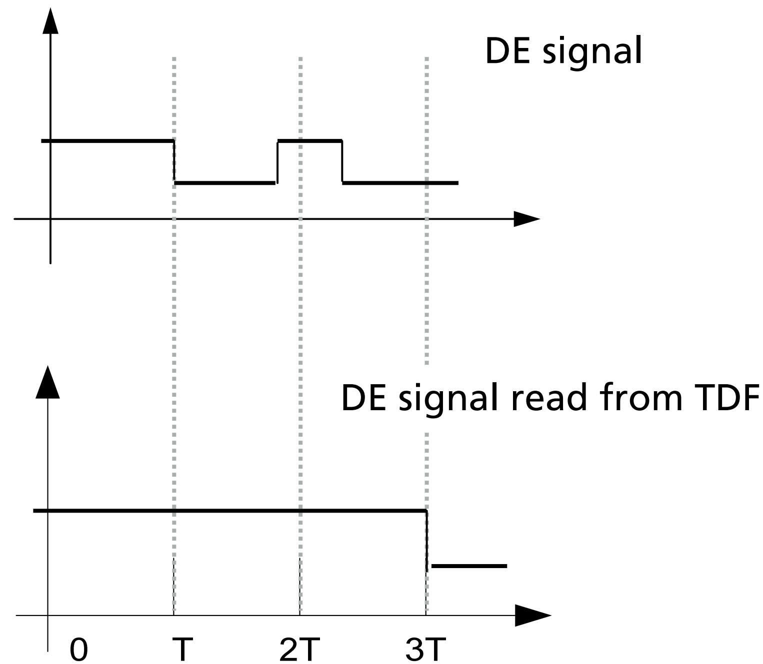
# Implementation of Multi-Rate SDF with Time (TDF) in SystemC-AMS



# Synchronization between TDF and DE

- TDF samples are mapped to sc_time.
- SystemC (DE) signals are sampled at Δ=0 of the specified sampling period. SDF samples are scheduled at Δ=0 as well (and thus valid at least at Δ=1).
- The sampling period T is specified as port attribute and propagated along the TDF signals of the cluster.
- That is why the sampling period must be specified at least for one port of a module in every TDF cluster – are ≥ 2 sampling periods given, the simulator performs a consistency check.



# AMS Draft 1 Standard



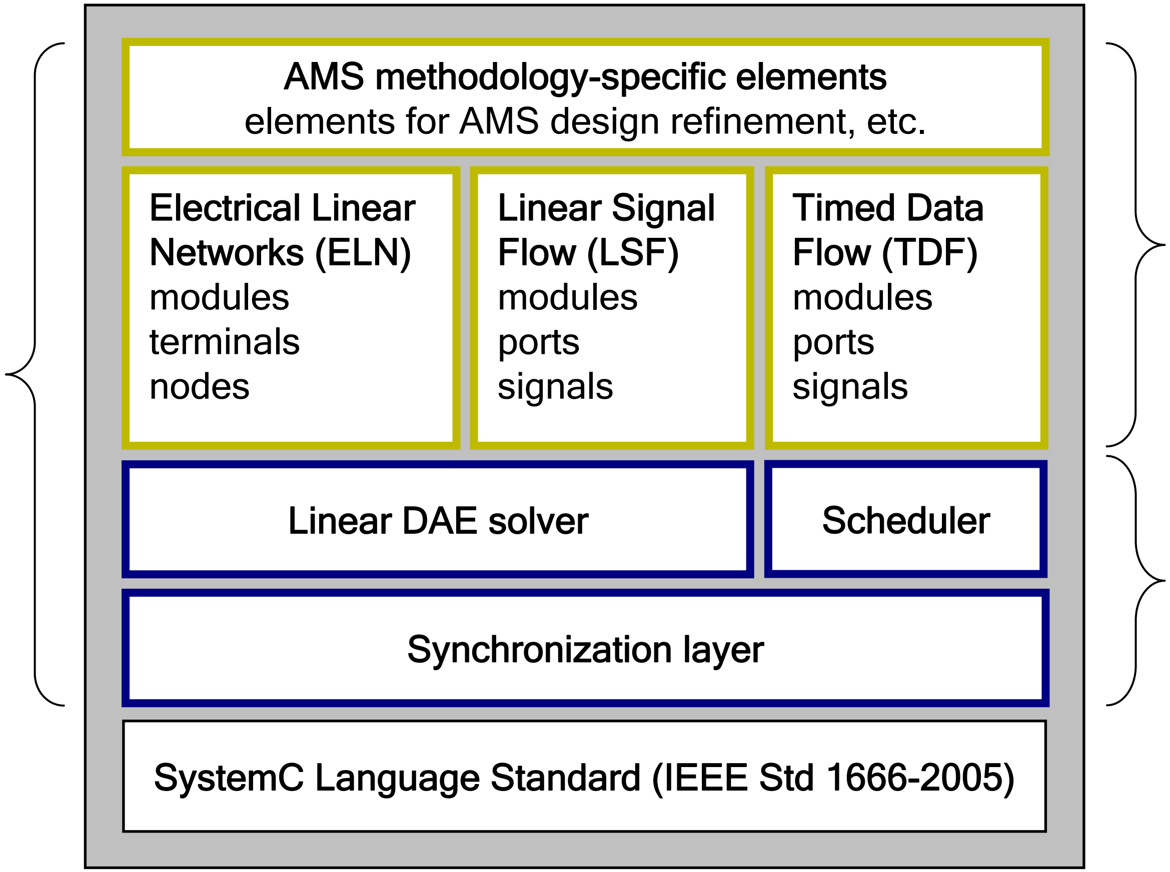
Fraunhofer
IIS

# 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

# SystemC AMS extensions LRM Draft 1

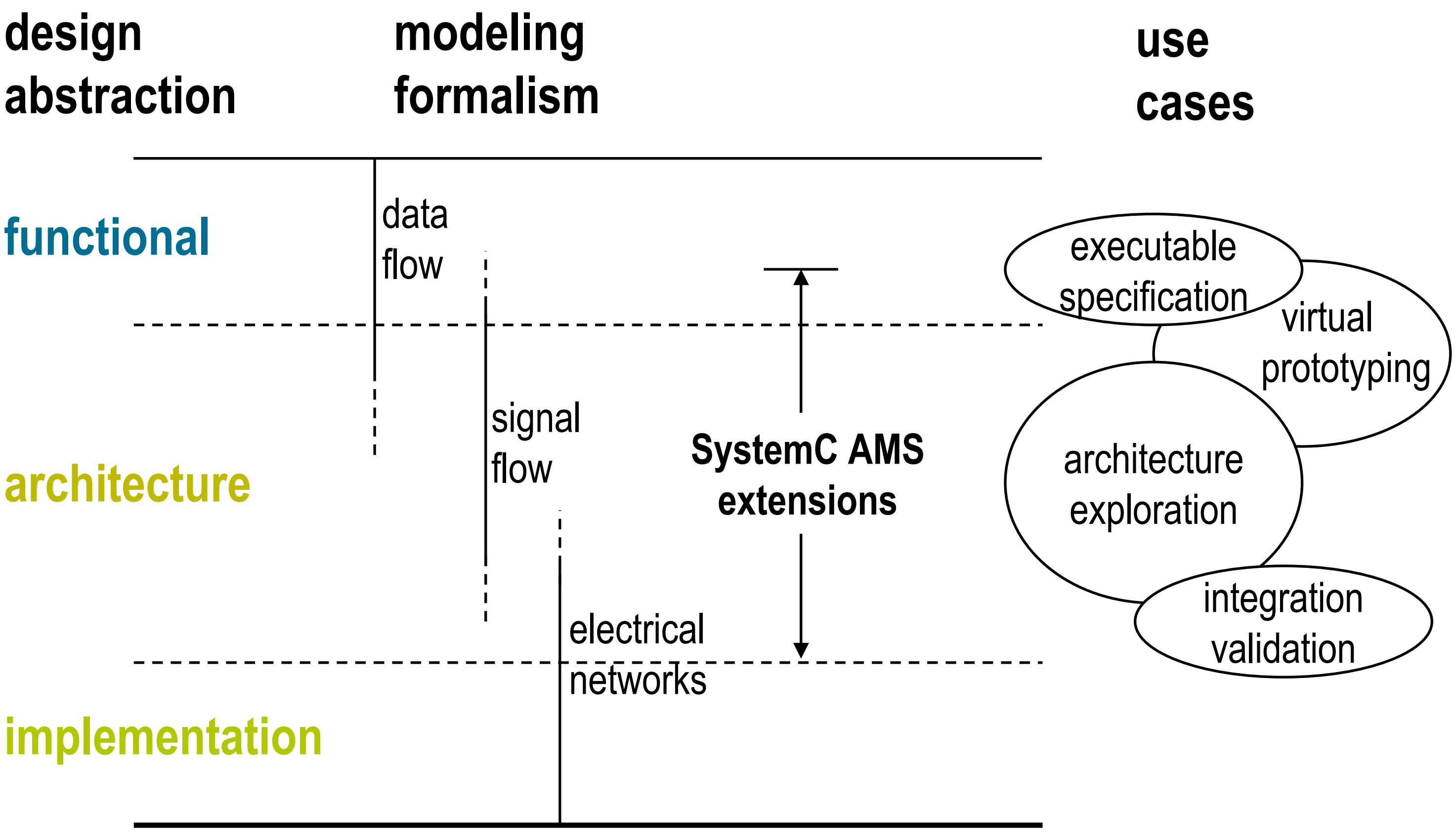
**Semantics** defined in AMS LRM Draft 1




**User features:** Classes and interfaces defined in AMS LRM Draft 1

**Enabling technology:** Classes and interfaces not defined in AMS LRM Draft 1

# Modeling formalisms and use cases



# Model abstraction and formalisms

*Use cases*

- Executable specification
- Virtual prototyping
- Architecture exploration
- Integration validation

*Model abstractions*

| Discrete-time<br>static non-linear | Continuous-time<br>dynamic linear | |
|---|---|---|
| Non-conservative behavior | | Conservative behavior |

*Modeling formalism*

| Timed Data Flow (TDF) | Linear Signal Flow (LSF) | Electrical Linear Networks (ELN) |
|---|---|---|

# SystemC AMS DRAFT 1

Modeling formalisms and language constructs

# SystemC AMS extensions - concept

- AMS modeling formalisms based on known *models of computation* (MoC)
  - Data flow - abstract modeling of non-conservative linear/non-linear
  - Signal flow - linear non-conservative modeling
  - Electrical networks – linear conservative modelling

- 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



Fraunhofer IIS

# 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 Language Definition Issues

- A Module represents a contribution of equations to a certain MoC
  - -> primitives of each MoC must be derived from a MoC specific base class

- The concept shall be extensible to an arbitrary number of MoC
  - -> how to handle the naming confusion / convention
  - -> Solution: Only a few base elements are defined, the assignment to the specific MoC is done by the namespace

# SystemC AMS extensions - module types

- AMS modules are derived from sca_core::sca_module which is derived from sc_core::sc_module
  - note: not all sc_core::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_coree::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 sca_core::sca_interface which is derived from sc_core: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

# SystemC AMS base Language Element Composition

- sca_module – base class for SystemC AMS primitive
- sca_in / sca_out – non-conservative (directed in/outport)
- sca_terminal – conservative terminal
- sca_signal – non-conservative (directed) signal
- sca_node / sca_node_ref – conservative node

- The MoC is assigned by the namespace e.g.:
  - sca_tdf::sca:module - base class for timed dataflow primitives modules
  - sca_lsf::sca_in - a linear signalflow inport
  - sca_tdf:sca_in<int> - a TDF inport
  - sca_eln::sca_terminal - an electrical linear network terminal
  - sca_eln::sca_node - an electrical linear network node





Fraunhofer IIS

# SystemC AMS Language Element Composition - Converter

- Converter elements are composed by the namespaces of booth domains:
  - **sca_tdf::sc_core::sca_in**<int>  - is a port of a TDF primitive module, which can be connected to an sc_core::sc_signal<int> or to a sc_core::sc_in<int>
    - Abbreviation: **sca_tdf::sc_in**
  - **sca_eln::sca_tdf::sca_voltage** – is a voltage source which is controlled by a TDF input
    - Abbreviation: **sca_eln::sca_tdf_voltage**
  - **sca_lsf::sc_core::sca_source** – is a linear signal flow source controlled by a SystemC signal ( sc_core::sc_signal<double>)
    - Abbreviation: **sca_lsf::sca_sc_source**

# Timed Data Flow (TDF) Modeling Constructs (selection)

- Module declaration macros

```
SCA_TDF_MODULE(<name>)
struct <name> : public sca_tdf::sca_module
```

- Port declarations dataflow ports

```
sca_tdf::sca_in< <type> >,
sca_tdf_sca_out< <type> >
```

- Port declaration converter ports (for TDF primitives only)

```
sca_tdf::sc_in< <type> >,
sca_tdf::sc_out< <type> >
```

- Virtual primitive methods called by the simulation kernel – overloaded by the user tdf primitive
- 

```
void set_attributes()
void initialize()
void processing()
void ac_processing()
```

- Methods for setting module activation timestep

```
void set_timestep(const sca_time&);
```

- Method for getting current module activation time

```
sca_time get_time()
```

- Constructor macro / constructor

```
SCA_CTOR(<name>)
<name>(sc_module_name nm)
```

- Channel/signal for connecting sca_tdf::sca_in / sca_tdf::sca_out ports

```
sca_tdf::sca_signal< <type> >
```





Fraunhofer IIS

# TDF Port Methods (selection)

- Sets/gets number of sample delay

```
void set_delay(unsigned long nsamples)
unsigned long get_delay()
```

- Sets/gets number of samples to read/write to the port per activation

```
void set_rate(unsigned long rate)
unsigned long get_rate()
```

- Sets/gets time distance of samples

```
void set_timestep(const sca_time&)
sca_time get_time_step()
```

- Gets absolute sample time

```
sca_time get_time(unsigned long sample)
```

- Writes initial value to delay buffer

```
void initialize(const T& value,
                unsigned long sample_id=0)
```

- Reads value from inport

```
const T& read(unsigned long sample_id=0)
```

- Writes value to outport

```
void write( const T& value,
            unsigned long sample_id)
```





Fraunhofer IIS

# 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. rate: in1.set_rate(2); or delay: in1.set_delay(1);
  }

  void initialize()
  {
    // put your initial values here e.g. in1.initialize(0.0);
  }

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

  SCA_CTOR(mytdfmodel) {}
};
```

# Linear Dynamic Behavior for TDF Models 1/2

- TDF Models can embed linear equation systems provided in the following three forms:

$$H(s)=\frac{b_n \cdot s^n + b_{n-1} \cdot s^{n-1} + \ldots + b_0}{a_m \cdot s^m + a_{m-1} \cdot s^{m-1} + \ldots + a_0}$$

- Linear transfer function in numerator / denumerator representation

$$H(s)=k \cdot \frac{(s-z_0)\cdot(s-z_1)\cdot \ldots \cdot(s-z_n)}{(s-p_0)\cdot(s-p_1)\cdot \ldots \cdot(s-p_n)}$$

- Linear transfer function in pole-zero representation

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

- State Space equations

# Linear Dynamic Behavior for TDF Models 2/2

- The equation systems will be represented and calculated by objects:
  - **sca_tdf::sca_ltf_nd** - Numerator / denumerator representation
  - **sca_tdf::sca_ltf_zp** - Pole-zero representation
  - **sca_tdf::sca_ss** - State space equations

- The result is a continuous time signal represented by a “artificial” object (sca_tdf::sca_ltf_proxy or sca_tdf::sca_ss_proxy)
  - This object performs the time discretization (sampling) in dependency of the context – this makes the usage more comfortable and increases the accuracy
  - This mechanism permits additionally a very fast calculation for multi-rate systems





Fraunhofer IIS

# Linear Signalflow (LSF) Modeling

- Library of predefined elements
- Permits the description of arbitrary linear equation systems
- Several converter modules to/from TDF and SystemC (sc_core::sc_signal) available
- Models for switching behavior like mux / demux available

- LSF models are always hierarchical models

- Ports:
  - sca_lsf::sca_in - inport
  - sc_lsf::sca_out - outport
- Channel / Signal:
  - sca_lsf::sca_signal

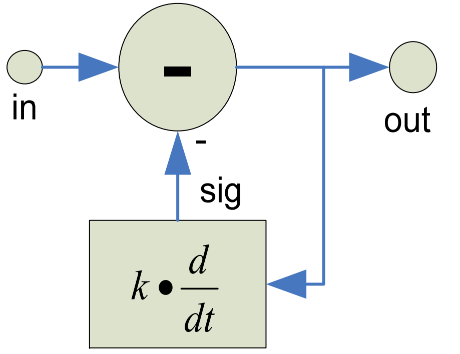
# LSF predefined modules

- 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
- sca_lsf::sca_ss

- sca_lsf::sca_tdf::sca_source (sca_lsf::sca_tdf_source)
- sca_lsf::sca_tdf::sca_gain (sca_lsf::sca_tdf_gain)
- sca_lsf::sca_tdf::sca_mux (sca_lsf::sca_tdf_mux)
- sca_lsf::sca_tdf::sca_demux (sca_lsf::sca_tdf_demux)
- sca_lsf::sca_tdf::sca_sink (sca_lsf::sca_tdf_sink)
- sca_lsf::sc_core::sca_source (sca_lsf::sca_sc_source)
- sca_lsf::sc_core::sca_gain (sca_lsf::sca_sc_gain)
- sca_lsf::sc_core::sca_mux (sca_lsf::sca_sc_mux)
- sca_lsf::sc_core::sca_demux (sca_lsf::sca_sc_demux)
- sca_lsf::sc_core::sca_sink (sca_lsf::sca_sc_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);
  }

};
```








Fraunhofer IIS

# Electrical Linear Network (ELN) Modeling

- Library of predefined elements
- Permits the description of arbitrary linear electrical network
- Several converter modules to/from TDF and SystemC (sc_core::sc_signal) available
- Models for switching behavior like switches

- ELN models are always hierarchical models

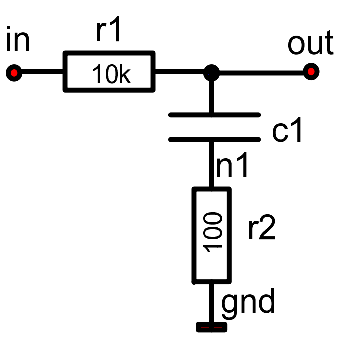
- Ports:
  - sca_eln::sca_terminal - conservative terminal
- Channel / Node:
  - sca_eln::sca_node – consevative node
  - sca_eln::sca_node_ref – reference node, node voltage is always zero

# ELN predefined elements

- sca_eln::sca_r
- sca_eln::sca_l
- sca_eln::sca_c
- sca_eln::sca_vcvs
- sca_eln::sca_vccs
- sca_eln::sca_ccvs
- sca_eln::sca_cccs
- sca_eln::sca_nullor
- sca_eln::sca_gyrator
- sca_eln::sca_ideal_transformer
- sca_eln::sca_transmission_line
- sca_eln::sca_vsource
- sca_eln::isource
- sca_eln::sca_tdf::sca_vsink
  - sca_eln::sca_tdf_vsink
- sca_eln::sca_tdf::sca_vsource
  - sca_eln::sca_tdf_vsource
- sca_eln::sca_tdf::sca_isource
  - sca_eln::sca_tdf_isource
- sca_eln::sc_core::sca_vsource
  - sca_eln::sc_vsource
- sca_eln::sc_core::sca_isource …
- sca_eln::sca_tdf::sca_r …
- sca_eln::sca_tdf::sca_l …
- sca_eln::sca_tdf::sca_c …
- sca_eln::sc_core::sca_r …
- sca_eln::sc_core::sca_l …
- sca_eln::sc_core::sca_c …
- …

# Example: ELN language constructs

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

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

  sca_eln::sca_r *r1, *r2;
  sca_eln::sca_c *c1;

  SC_CTOR(myelnmodel)                // standard constructor
  {
    r1 = new sca_eln::sca_r("r1");  // instantiate predefined
    r1->p(in);                      // primitive here (resistor)
    r1->n(out);
    r1->value = 10e3;                      //named parameter association

    c1 = new sca_eln::sca_c("c1", 100e-6); //positional parameter association
    c1->p(out);
    c1->n(n1);

    r2 = new sca_eln::sca_r("r2",100.0);
    r2->p(n1);
    r2->n(gnd);
  }

};
```








Fraunhofer IIS

# Small Signal Frequency Domain Analysis (AC-Analysis)

- Support of two flavors:
  - “classical” AC domain setup and calculates linear complex equation system
  - AC noise domain – setup linear complex equation system and solves it for each noise source contribution (other source contributions will be neglected) – adds the results arithmetically
- ELN and LSF description are specified in the frequency domain
- TDF description must specify the linear complex transfer function of the module using the method ac_processing (otherwise they do not contribute – the out values will be assumed as zero)
- This transfer function can depend on the current time domain state (e.g. the setting of a control signal)
- Two simulation start commands:
  - sca_ac_start(double startf,double endf,unsigned long npoints, sca_ac_scale)
  - sca_ac_noise_start(…)





Fraunhofer IIS

# Tracing of analog Signals

- SystemC AMS has a own trace mechanism:
  - Analog / Digital timescales are not always synchronized
  - the vcd file format is in general inefficient for analog
- Traceable are:
  - all sca_signals, sca_nodes (voltage) and sc_core::sc_signals
  - Most ELN modules – the current through the module
  - for TDF a traceable variable to trace internal model states
- Two formats supported:
  - Tabular trace file format - **sca_util::sca_create_tabular_trace_file**
  - VCD trace file format - **sca_util::sca_create_vcd_trace_file**
- Features to reduce amount of trace data:
  - enable / disable tracing for certain time periods, redirect to different files
  - different trace modes like: sampling / decimation



Fraunhofer IIS

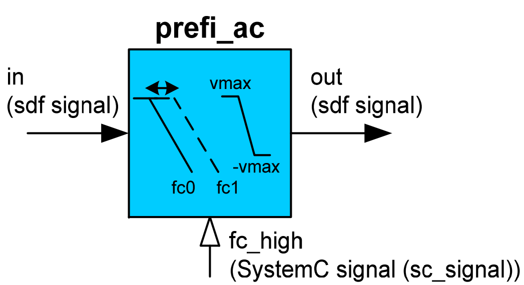
# Code example

# TDF Module – Example with LTF

```
SCA_TDF_MODULE(prefi_ac)
{
  sca_tdf::sca_in<double>    in;
  sca_tdf::sca_out<double> out;

  // control / DE signal from SystemC
  // (connected to sc_signal<bool>)
  sca_tdf::sc_in<bool>   fc_high;

   double fc0, fc1;
   double v_max;

    // filter equation objects
   sca_tdf::sca_ltf_nd ltf_0, ltf_1;
   sca_util::sca_vector<double>  a0, a1, b;
   sca_util::sca_vector<double> s;

   void initialize()
   {
     const double r2pi = M_1_PI * 0.5;
     b(0)   = 1.0;     a1(0) = a0(0) = 1.0;
     a0(1) = r2pi/fc0; a1(1) = r2pi/fc1;
   }
```

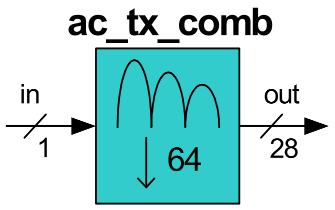
```
void processing()
{
  double tmp; // high or low cut-off freq.
  if(fc_high)  tmp = ltf_1(b, a1, s, in);
  else         tmp = ltf_0(b, a0, s, in);

  //output value limitation
  if      (tmp >  v_max)  tmp =  v_max;
  else if (tmp < -v_max)  tmp = -v_max;

    out.write(tmp);
 }

  SCA_CTOR(prefi_ac)
  {    // default parameter values
   fc0 = 1.0e3;  fc1=1.0e5;  v_max  = 1.0;
  }
};
```

$$H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}$$



# Frequency Domain Specification

```
SCA_TDF_MODULE(ac_tx_comb)
{
  sca_tdf::sca_in<bool>          in;
  sca_tdf::sca_out<sc_int<28> > out;

  void set_attributes()
  {
    in.set_rate(64);     // 16 MHz
    out.set_rate(1);    // 256 kHz
  }

  void ac_processing()
  {
    double      k  = 64.0;
    double      n  =   3.0;

    // complex transfer function:
    sca_complex h;
    h = pow( (1.0 - sca_ac_z(-k)) /
             (1.0 - sca_ac_z(-1)),n);

      sca_ac(out) = h * sca_ac(in) ;
  }

  void processing()
  {
      int x, y, i;
      for (i=0; i<64; ++i) {
          x = in.read(i);
      …
      out.write(y);
  }

  SCA_CTOR(ac_tx_comb)
  {
      …
  }
};
```




$$H(z)=\left(\frac{1-z^{-k}}{1-z^{-1}}\right)^{n}$$

$$z=e^{j2\pi f/f_s}$$

# 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;

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

  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);

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

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

**SC_MODULE used for hierarchical structure**

**usage of different signals**

**High-level mixer model (TDF module)**

**LPF close to implementation level (ELN module)**

**easy to combine with normal SystemC modules !**





Fraunhofer IIS

# SystemC AMS Testbench

```
#include <systemc-ams.h>
           :
int sc_main(int argn,char* argc)
{
  //instantiate signals, modules, …
  //from abritrary domains e.g.:

  sca_tdf::sca_signal<double>  s1;;
  sca_eln::sca_node            n1;
  sca_lsf::sca_signal         slsf1;
  sca_core::sca_signal<bool>  scsig1;
                :
  dut i_dut("i_dut");
    i_dut->inp(s1);
    u_dut->ctrl(scsig1);
           :
  //no difference/restriction to
  //"classical" SystemC
//"classical" SystemC tracing
sc_trace_file*
sctf=sc_create_vcd_trace_file("sctr");
   sc_trace(sctf,scsig1,"scsig1"); …
```

```
sca_trace_file*
satf=sca_create_tabular_trace_file("mytr.dat");
  sca_trace(satf,s1,"s1");
  sca_trace(satf,n1,"n1"); …

 //start time domain simulation for 2ms
 sc_start(2.0,SC_MS);
 satf->disable(); //stop writing
 sc_start(2.0,SC_MS); //continue 2ms
 satf->enable();     //continue writing
 sc_start(2.0,SC_MS); //continue 2ms

 //close time domain file, open ac-file
 satf->reopen("my_tr_ac.dat");
 //calculate ac behavior at current op
 sca_ac_start(1.0,1e6,1000,SCA_LOG);
 //reopen transient file, append
 satf->reopen("mytr.dat",std::ios::app);
 //sample results with 1us time distance
 satf->set_mode(sca_sampling(1.0,SC_US));
 sc_start(100.0,SC_MS); //continue
}
```

# Fraunhofer Add-on Libraries

# SystemC AMS Nonlinear Extension

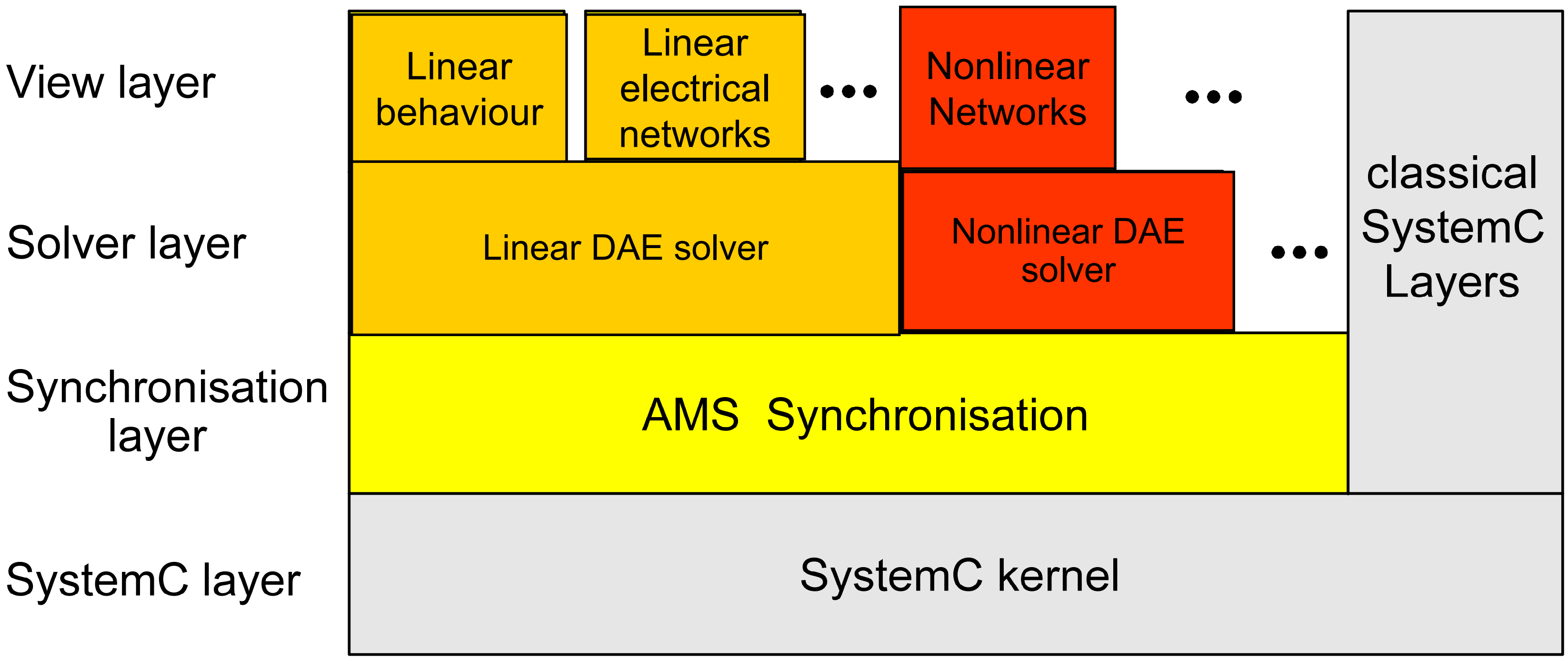
- Also on system level, there are parts of the circuit which can not be modeled idealized with the required accuracy
- Especially the influence of **front/back ends** like driver stages have often a complicated not negligible dynamic non-linear behavior
- The nonlinear extension enables modeling possibilities like the languages VHDL-AMS, Verilog-AMS or Modellica
- Permits the description (and simulation) of nonlinear dynamic modules using equations and there connection via a conservative network
- Description of reactivity to events from the de-SystemC kernel
- Detection of threshold crossings and scheduling of events at crossing time





Fraunhofer IIS

# SystemC-AMS Generic Concept





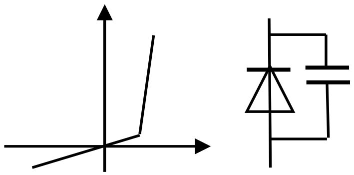



Fraunhofer IIS

# Outlook Nonlinear conservative MoC

```
SCA_NLN_MODULE(sca_nl_rdiode)
{
  sca_nln::sca_terminal<electric> a;
  sca_nln::sca_terminal<electric> b;

  double v_thres;
  double r_on;
  double r_off;
  double cj;

  sca_nln::sca_var  v_diode;

  void equations();

  SCA_CTOR(sca_nl_rdiode)
  {
    v_thres  = 0.7;
    r_on     = 1e-2;
    r_off    = 1e7;
    cj       = 1e-12;
  }

};
```

```
void sca_nl_rdiode::equations()
{
  eq(v_diode) <<
             v_diode == ( a.v() - b.v() );

  double i_diode;

  if ( v_diode.above(v_thres) )
  {
    i_diode = ( v_diode - v_thres ) / r_on
                          + v_thres / r_off;
  }
  else
  {
    i_diode = v_diode / r_off;
  }

  i_diode += cj * v_diode.dt();

   a += i_diode;
   b -= i_diode;
}
```



# SystemC-AMS Extension for Statistic Modeling

- Based on VHDL-AMS standard SAE J2748 (Society of Automotive Engineers)
- Independent from tool implementation, platform, compiler version, …
- The SAE standard includes descriptions for different distribution functions, such as:
  - UNIFORM
  - NORMAL
  - BERNOUILLI
  - PWLPDF (piecewise linear probability density function)
  - PWLCDF (piecewise linear cumulative density functions)
  - ...
- Permits the description of user defined distribution functions
- Simulation control e.g. to perform Monte Carlo available





Fraunhofer
IIS

# Simple application example: correlated resistors

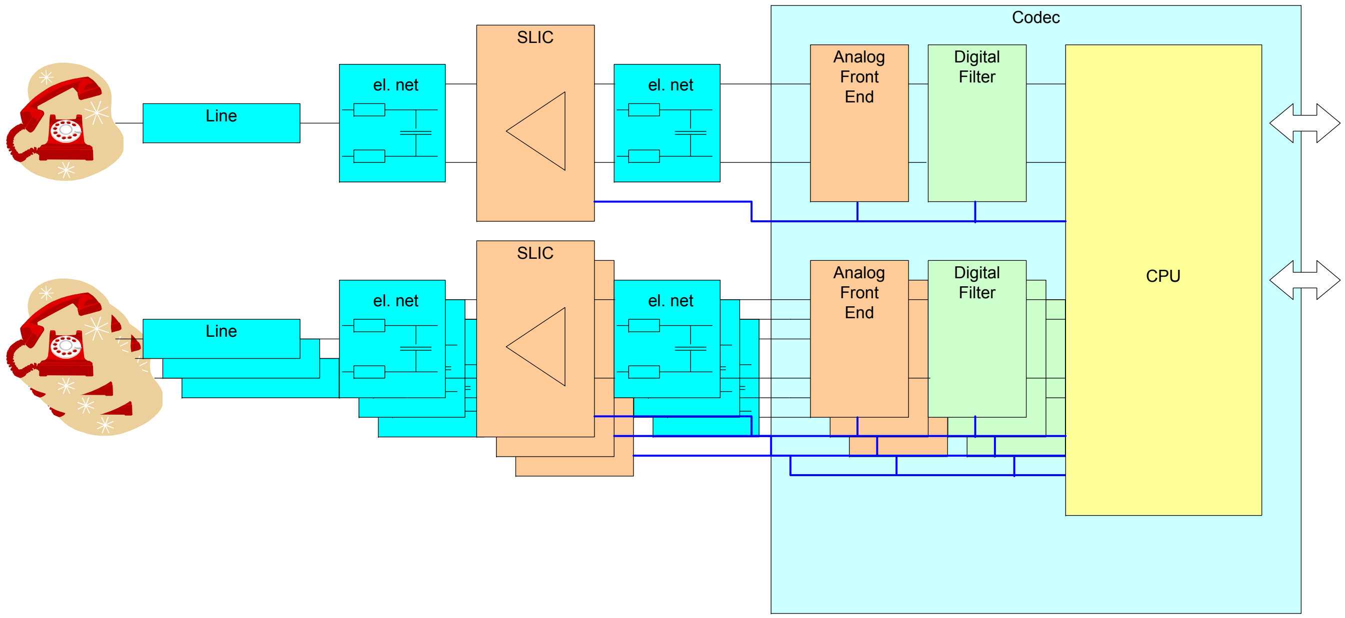
- Statistic information added while parametrizing
  - Example shows correlated resistors
  - Statistic information will be used by simulation control
  - Monte Carlo, Corner Case or more sophisticated algorithm for parameter selection possible

```
…
// definition of a reference resistance value
const double rvalue_global = normal(100.0, 0.2);

sca_r r1;
r1.p(el_node_n1);
r1.n(el_node_n2);
r1.value = uniform(5.0 * rvalue_global, 0.01);

sca_r r2;
r2.p(el_node_n3);
r2.n(el_node_n4);
r2.value = uniform(20.0 * rvalue_global, 0.003);
```

# Application Examples

# POTS System Design

- Complete System functionality modeled
- All relevant analogue effects
- Digital parts “bittrue”, original code of embedded software
- Hundreds of simulation scenarios as regression tests available
- Simulation scenarios partially re-used for silicon verification
- Embedded software debugged before silicon



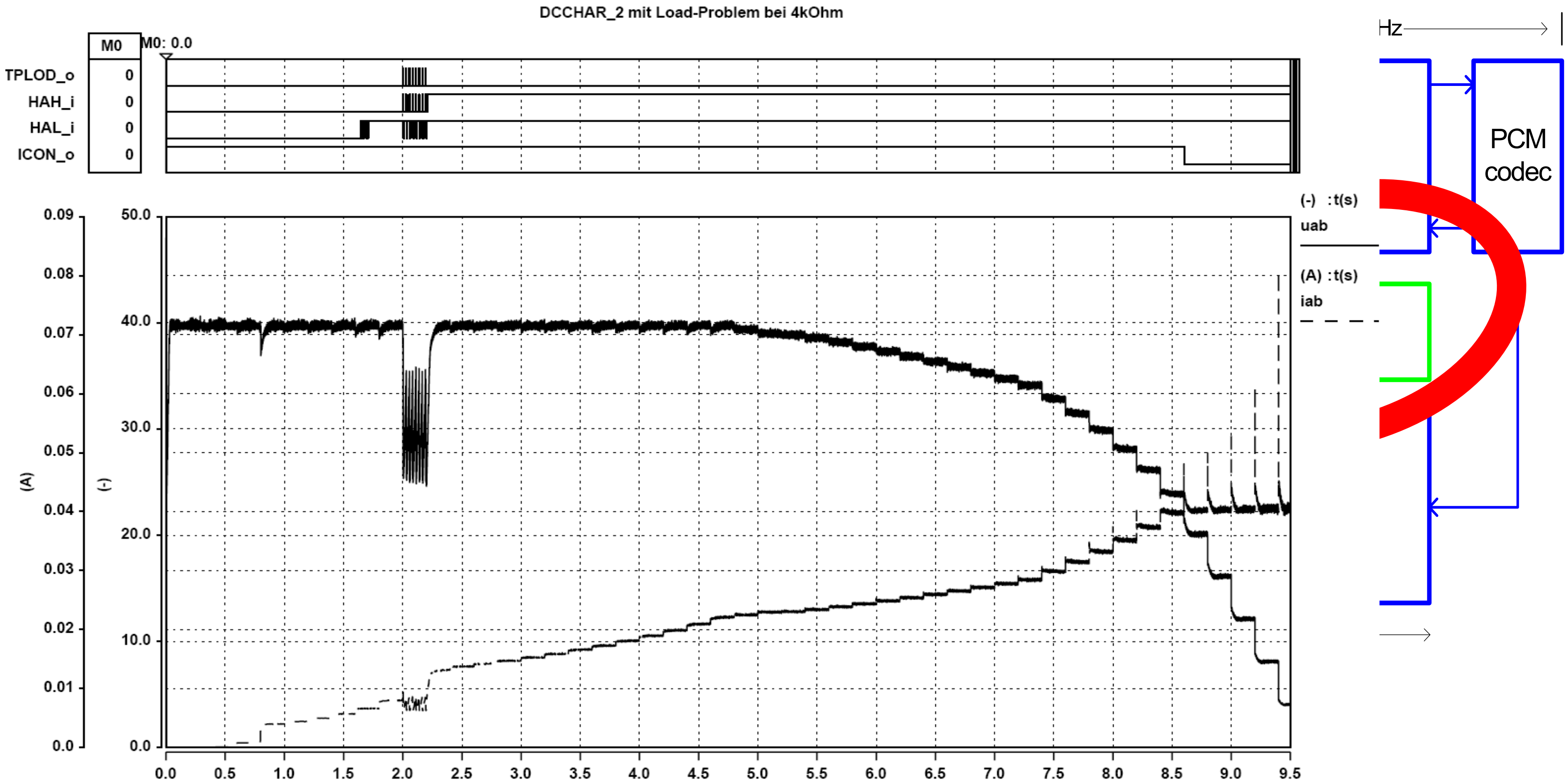



Source: Gerhard Nössing LANTIQ



Fraunhofer IIS

# Hardware / Software Cosimulation






Source: Gerhard Nössing
LANTIQ



Fraunhofer IIS

# Simulation Time for Vinetic 2CPE System

SystemC-AMS Simulation

- 1 sec realtime ➔ 1,5h simulation time

- 2 channel including: SLIC, externals, AFE, DFE, ASDSP and part of Carmel FW

VHDL RTL

- 1 sec realtime ➔ 300h simulation time

- 2 channel including: AFE, DFE, ASDSP, Carmel and Interfaces

Nano Sim (Fast CMOS simulator)

- 1 ms realtime ➔ 15h simulation time

- 2 channel including: AFE top level

Titan Simulation

- 1 ms realtime ➔ 500h simulation time

- 2 channel including: AFE top level

SystemC-AMS Simulation for FW development

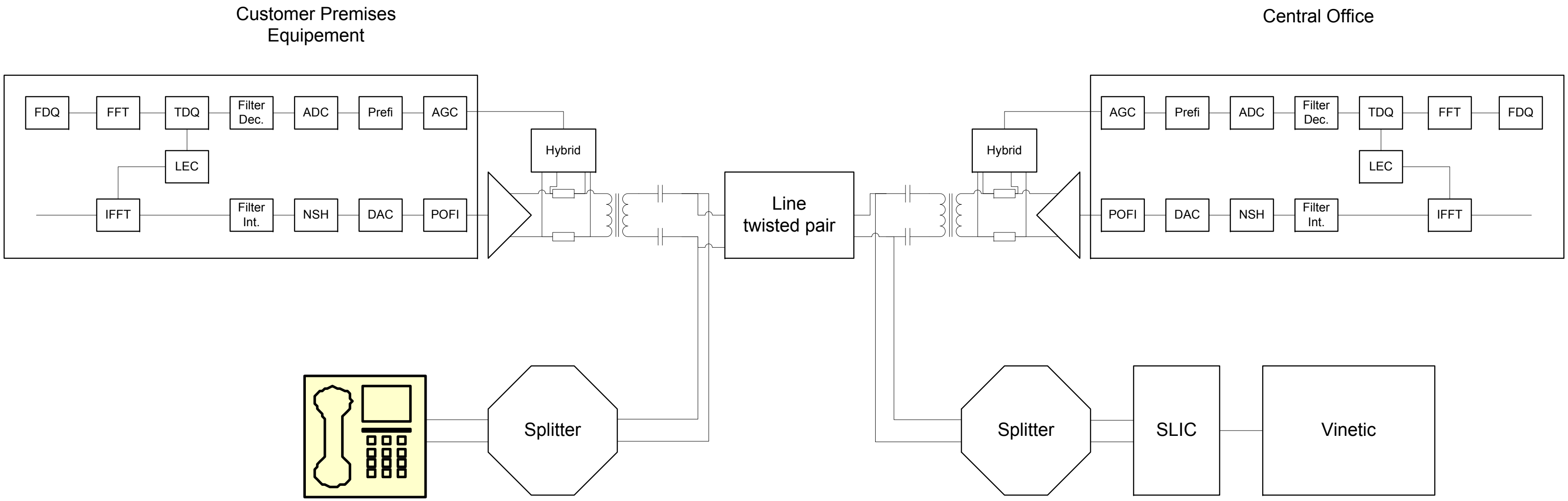
- 1sec realtime ➔ 90 sec simulation time

- only one channel
- reduce sampling rate for analog blocks

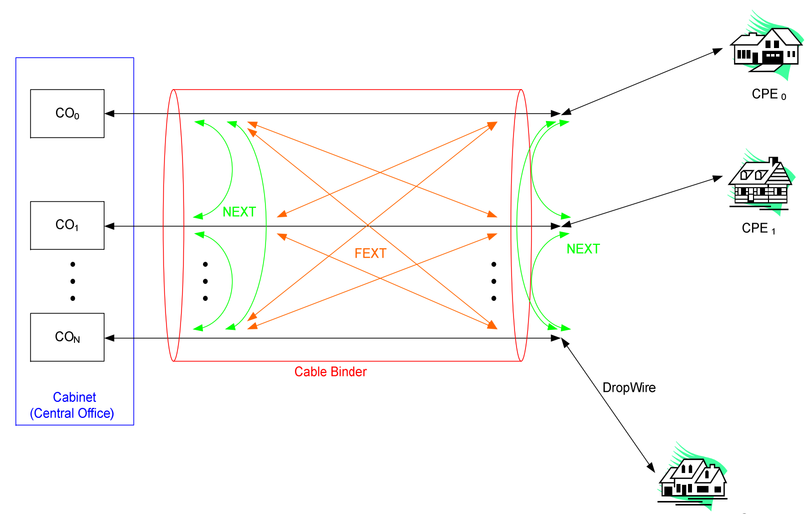


Source: Gerhard Nössing
LANTIQ

# ADSL / VDSL Systems




- Transient settling behavior
- Interaction Voice / Data transmission
- Training algorithm
- BER estimations
- Numerous of use scenarios
- Interaction of different lines
- Multi level simulation environment essential



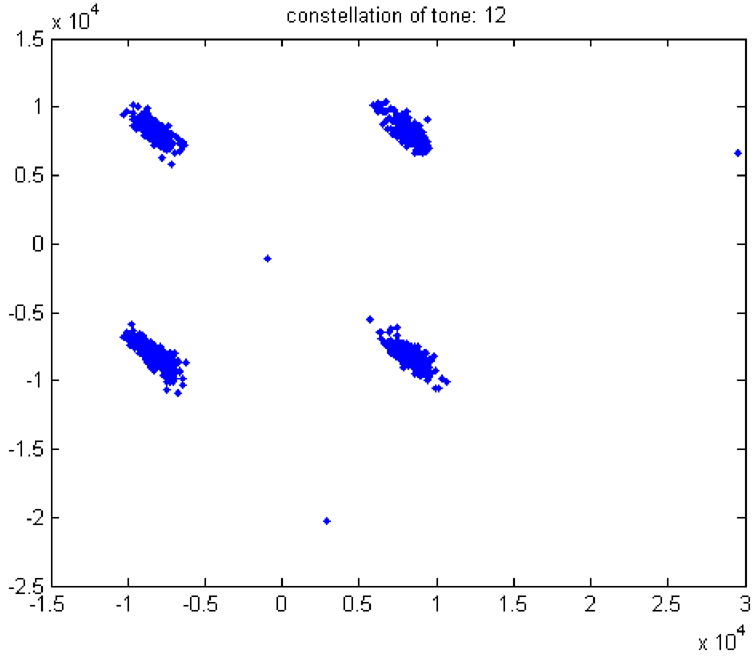


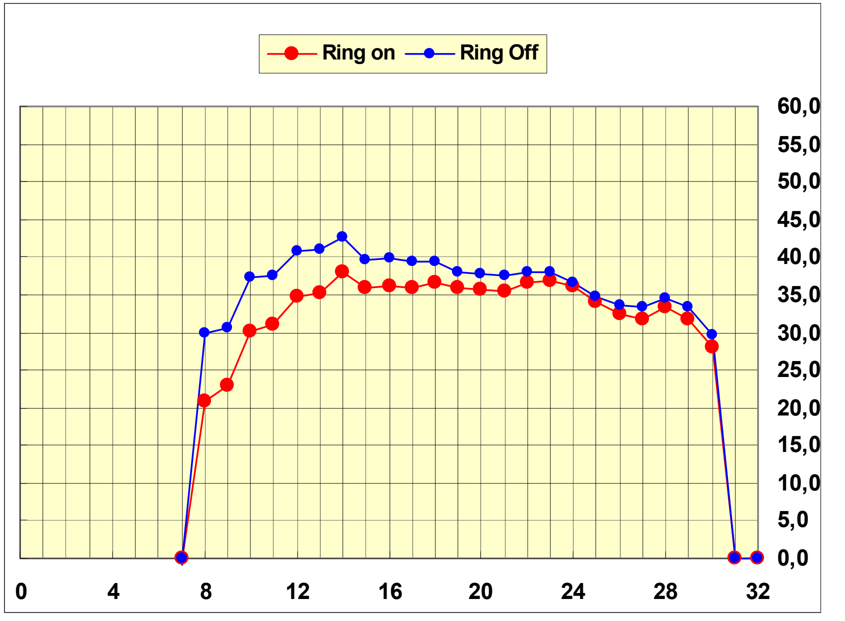



Source: Gerhard Nössing
LANTIQ

# ADSL Simulation

Backlash Voice to data path








Source: Gerhard Nössing
LANTIQ

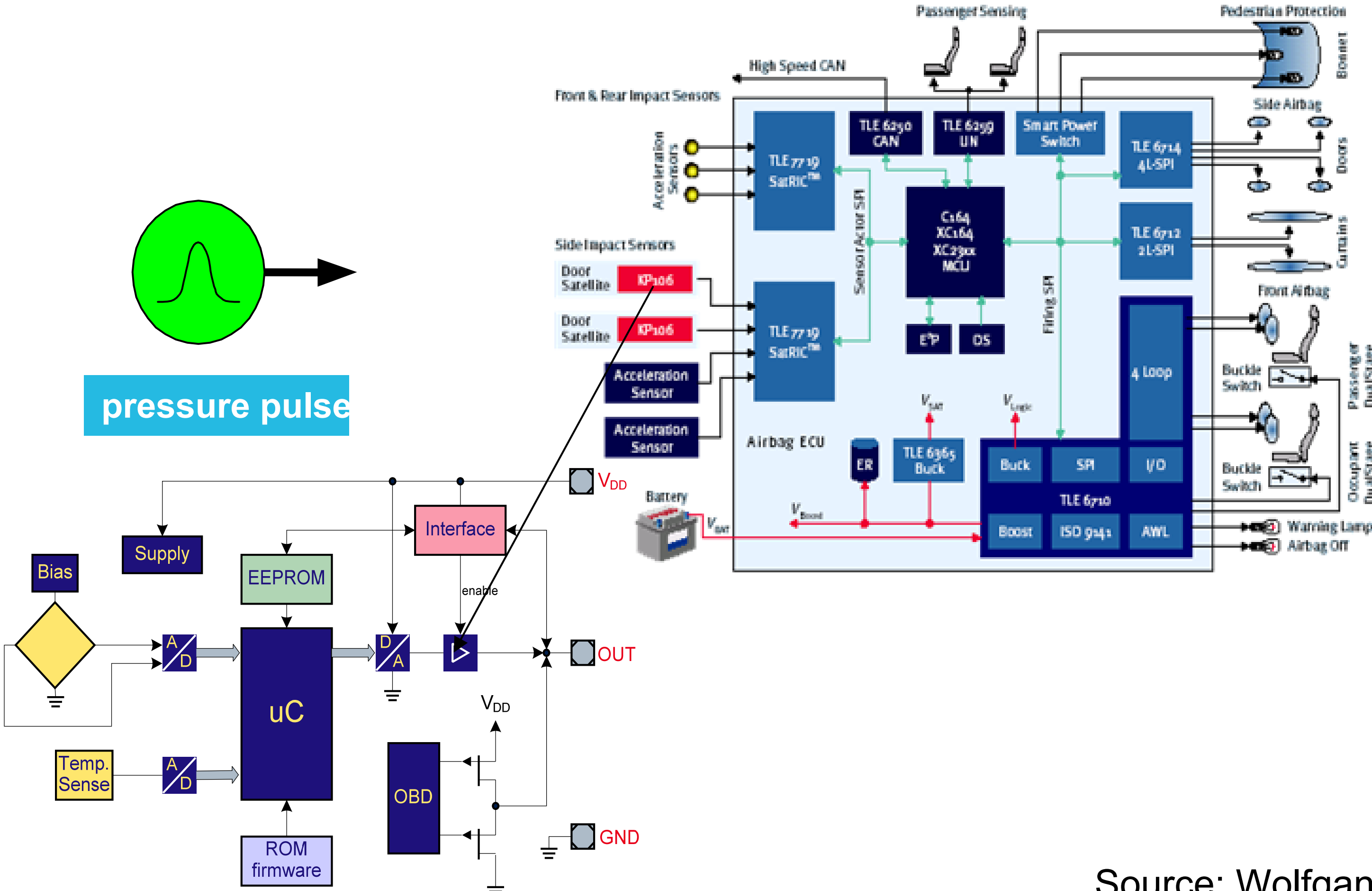
Fraunhofer IIS

# Automotive Sensor Applications

TIER2

TIER1

OEM




Source: Wolfgang Scherr Infineon AIM



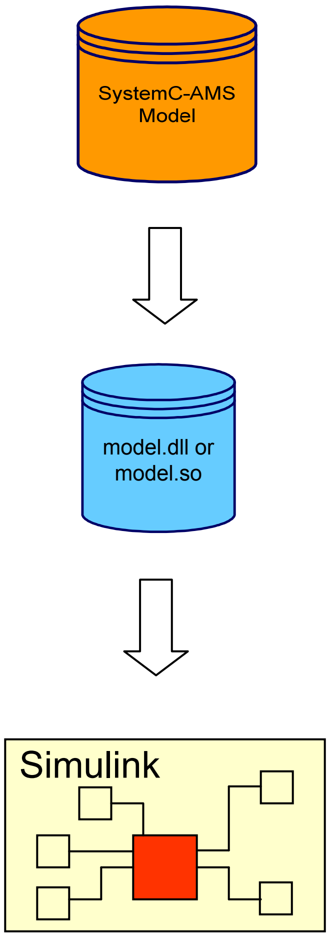


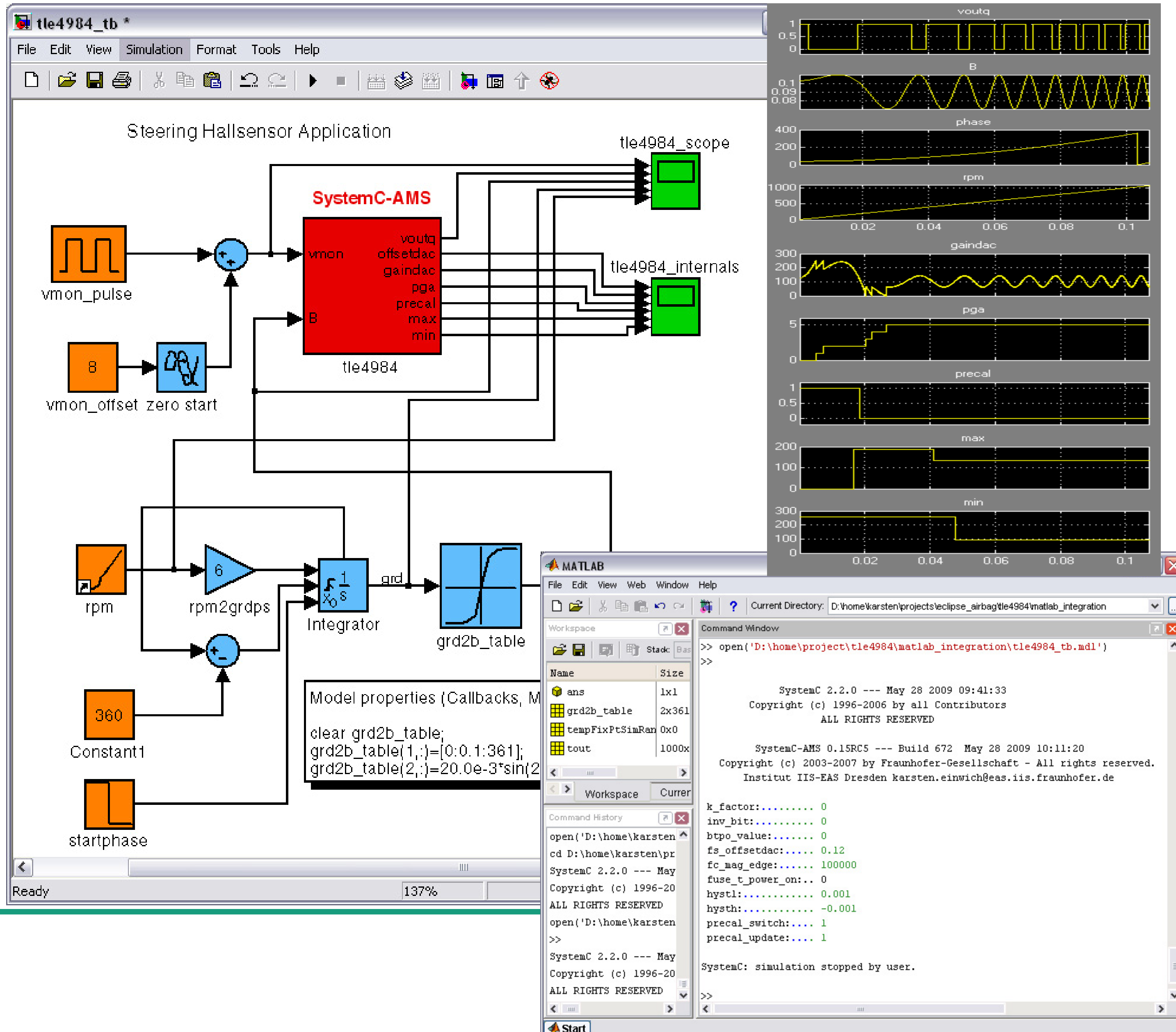
Fraunhofer IIS

# Automotive Sensor Projects

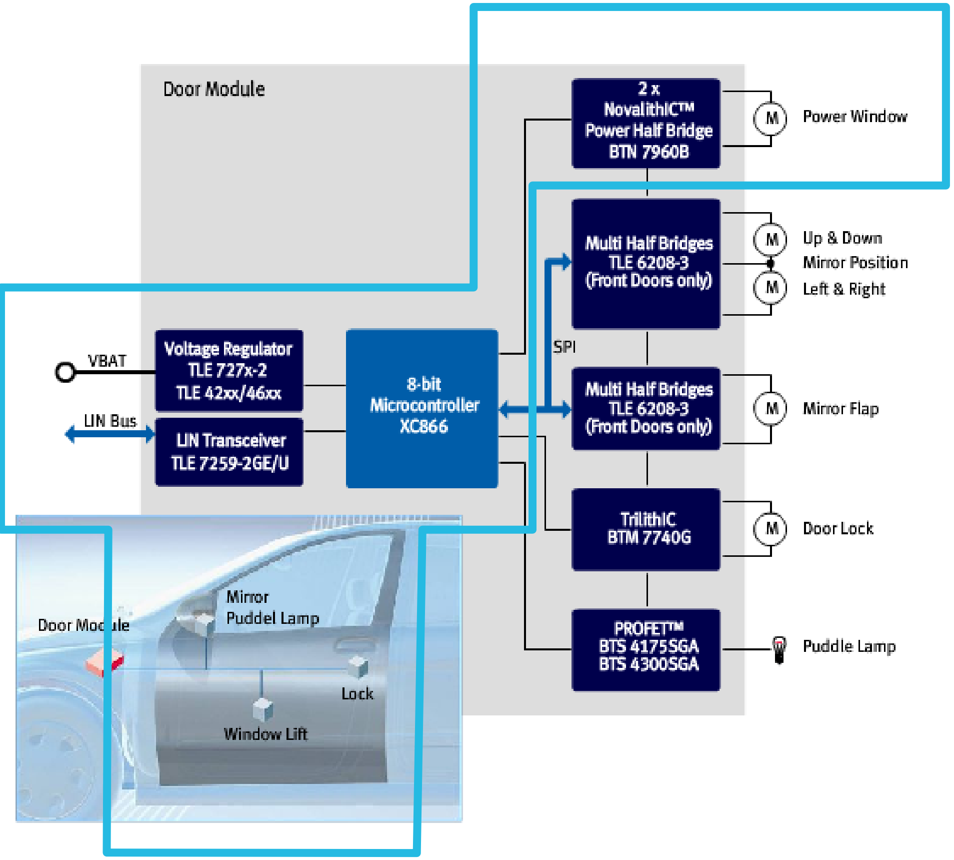
- Systemlevel model including the embedded processor on a cycle accurate level
- Switched capacitor converter
- Diagnose modes, offset calibration, temperature dependencies, noise, manchester interface, synchronization via supply voltage, ...
- Original code of embedded software
- TLM based modeling for processor communication
- IP protected customer model as Matlab/Simulink Module (mex – dll)
- Simulation performance ~ 10min /sec

# SystemC AMS Modelexchange via Simulink Integration



# Window Lifter – Substantial Subset of Door ECU




SystemC(-AMS) simulations (functional and system MC) for substantial parts of ECU on

- ECU architecture
- Chip architecture in system context
- Chip design in system context
- ECU design

This research project (AutoSUN) is supported by the German Government, Federal Ministry of Education and Research under the grant number 01M3178

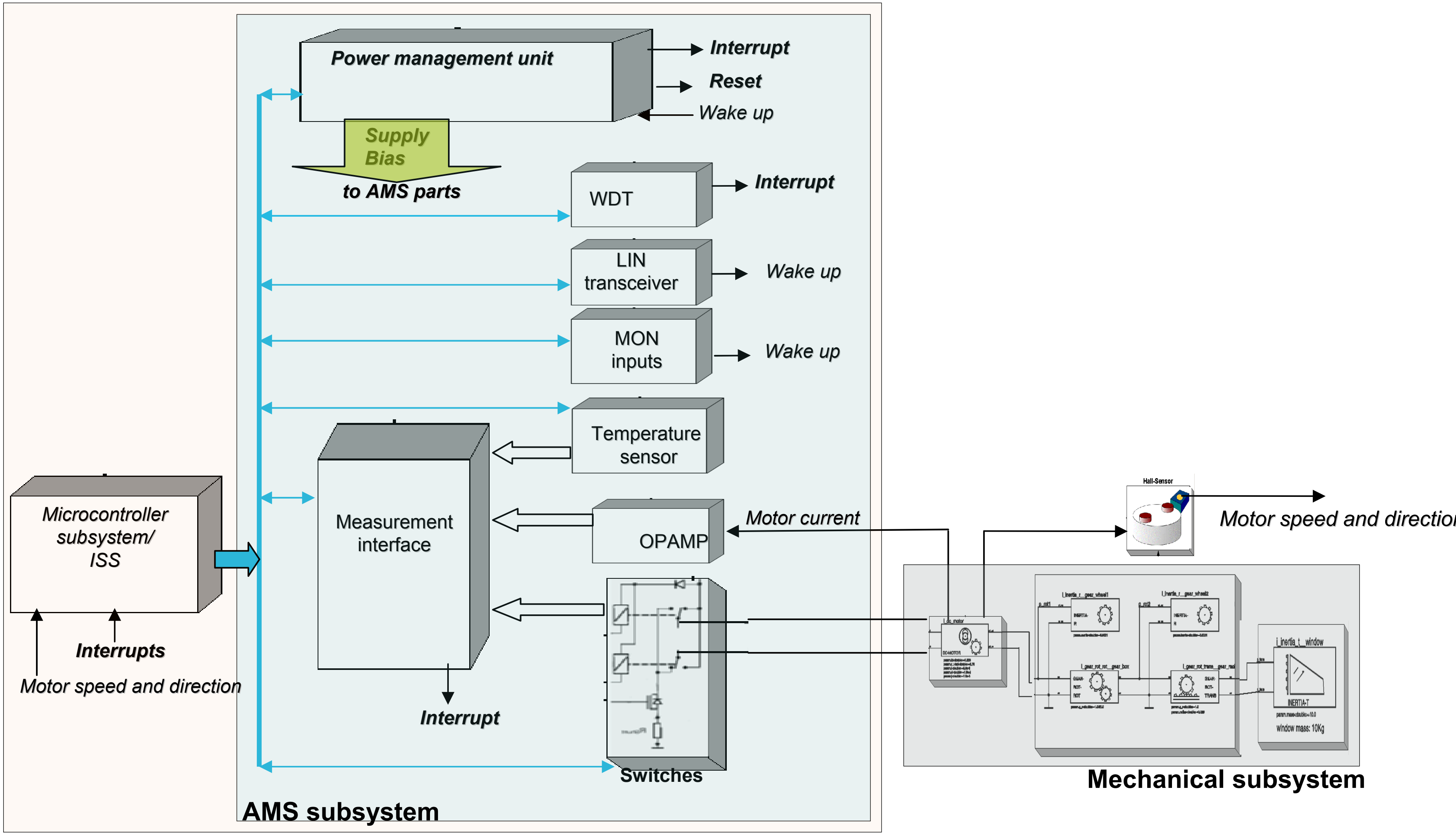
Bundesministerium
für Bildung
und Forschung





Source:
Georg Pelz Infineon Technologies

# Window Lifter Model Overview

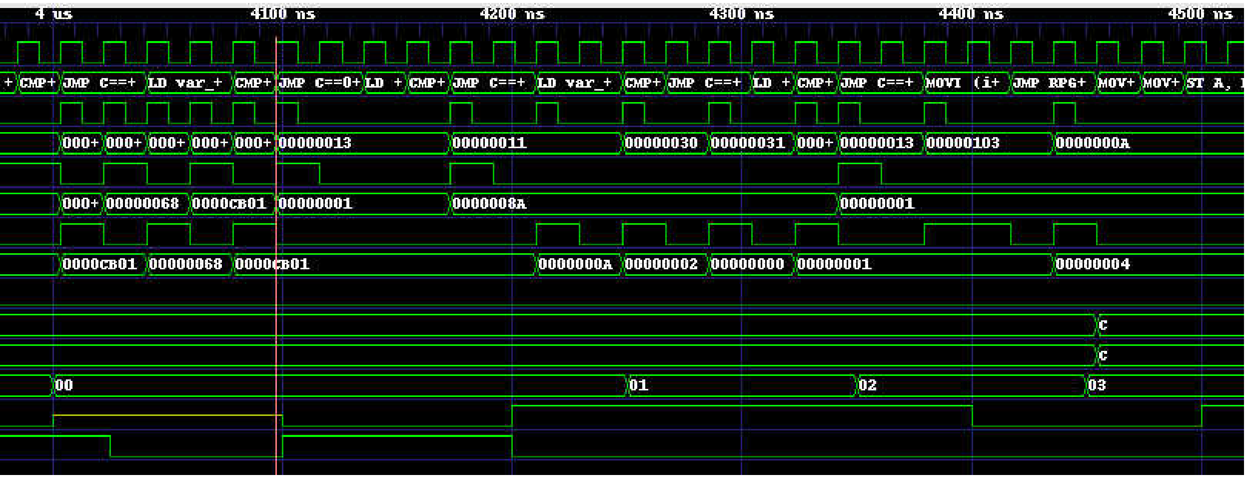




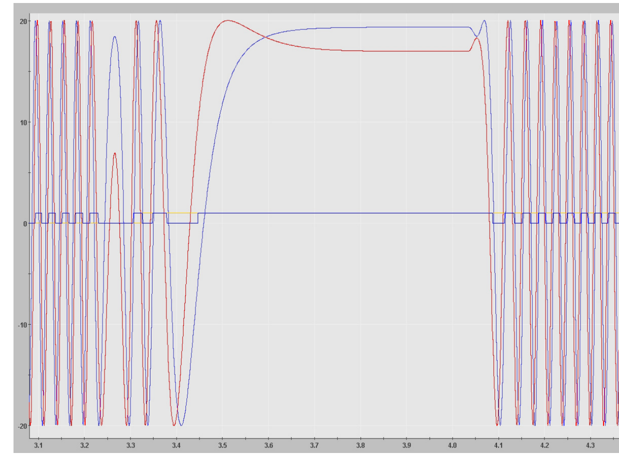



Source:
Georg Pelz, Monica Rafaila
Infineon Technologies

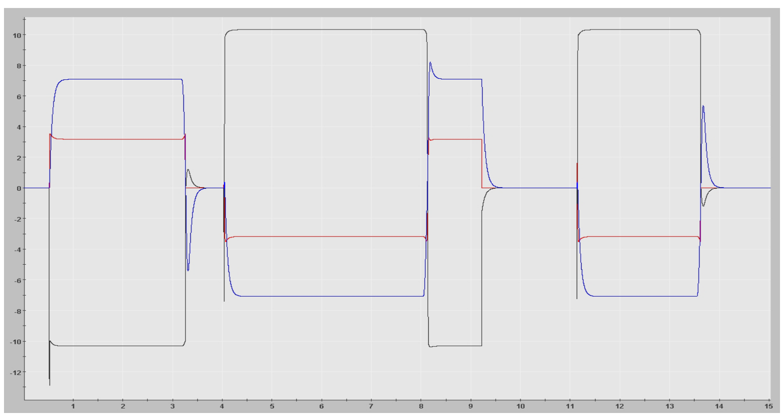
# Window Lifter Simulation Results



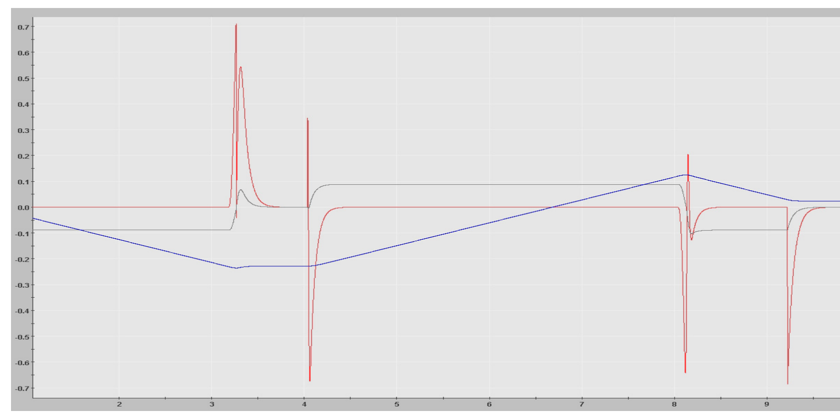

A huge amount of digital HW/SW …



Magnetic flux -> digital sensor out



Electronics: voltages/currents -> torque



Mechanics: position, torque, forces, …

# System Level Parameter Variations

Major parameters of a window lifter:

Environment

- Battery voltage
- Ambient temperature

Power bridge

- Slew-rate / on-resistance / impedance
- Over-current limits
- Over-temperature limits
- Free-wheeling behavior

Electric motor

- Armature L, R, friction, inertia
- Torque constant / backEMF Gear
- Friction, inertia

Mechanical load

- Friction, inertia
- Position of stop

→ > 20 parameters

All parameters have a distribution

Major goal of a development is cost minimization

- Resulting in even bigger spec windows for parameters
- Classical way out blocked: belt and suspenders

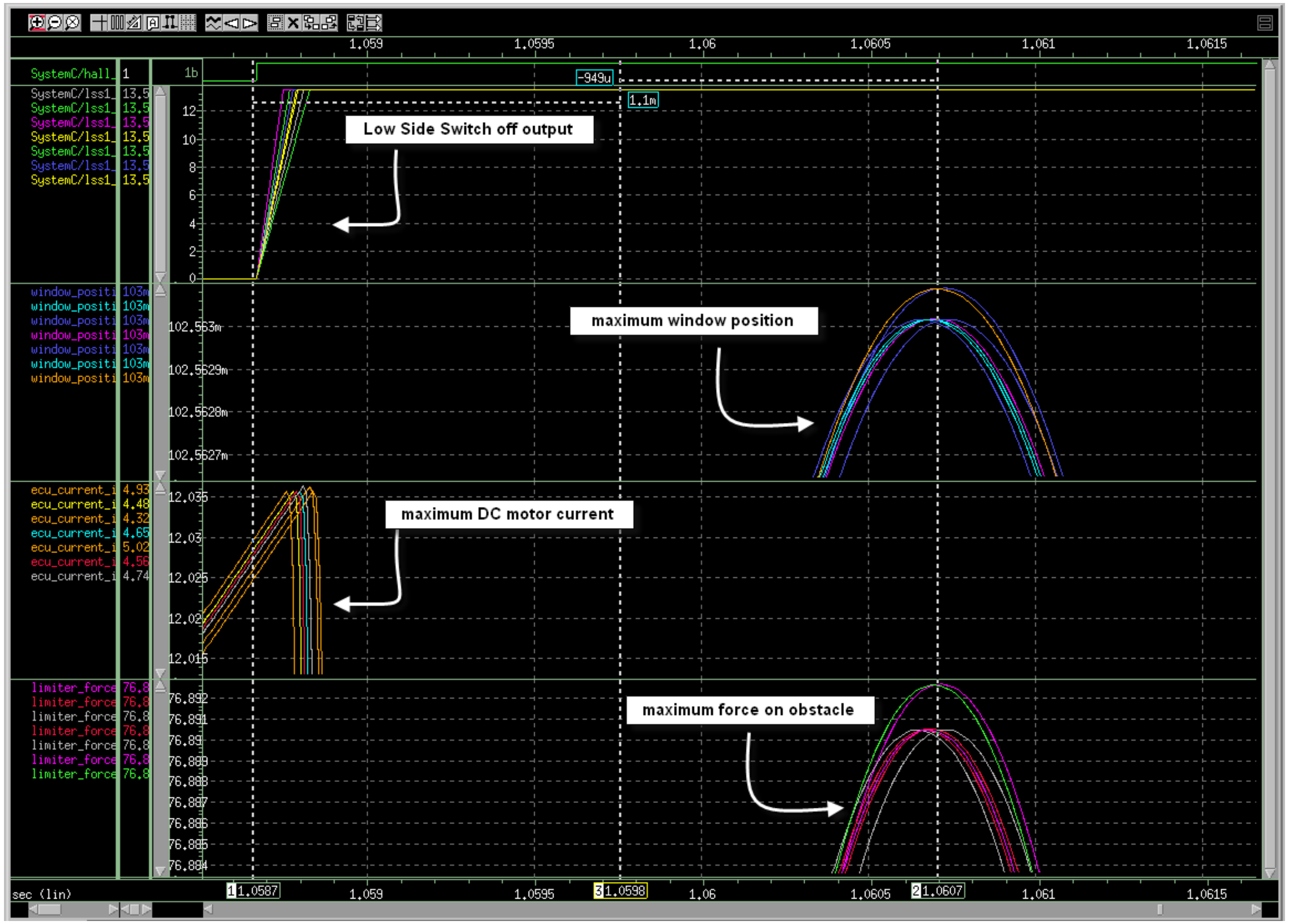
Nonetheless, function has to be proven for any choice of parameters





Source:
Georg Pelz, Monica Rafaila
Infineon Technologies

# Tons of Simulations with MC-Varied Parameters




- Tons of simulations deliver tons of data
- Here's one way to illustrate the results including the parameter and results correlations …





Source:
Georg Pelz, Monica Rafaila
Infineon Technologies

# Simulation Performance – Complete Window Lifter

How to measure simulation performance:

- Evaluate factor F:

$$F = \frac{\text{simulation time}}{\text{simulated time}}$$

**SystemC-AMS**
**Software on algorithm level**,
i.e. directly bound-in

- Factor F ≈ 5 – 100

**SystemC-AMS**
**Cycle-accurate MCU model**

- Factor F ≈ 1.000 – 5.000

**VHDL-AMS / VHDL**

- Factor F ≈ 100.000 – 1.000.000

**FastMOS (Ultrasim)**

- Factor F ≈ 1.000.000.000 – 10.000.000.00 (start-up only)

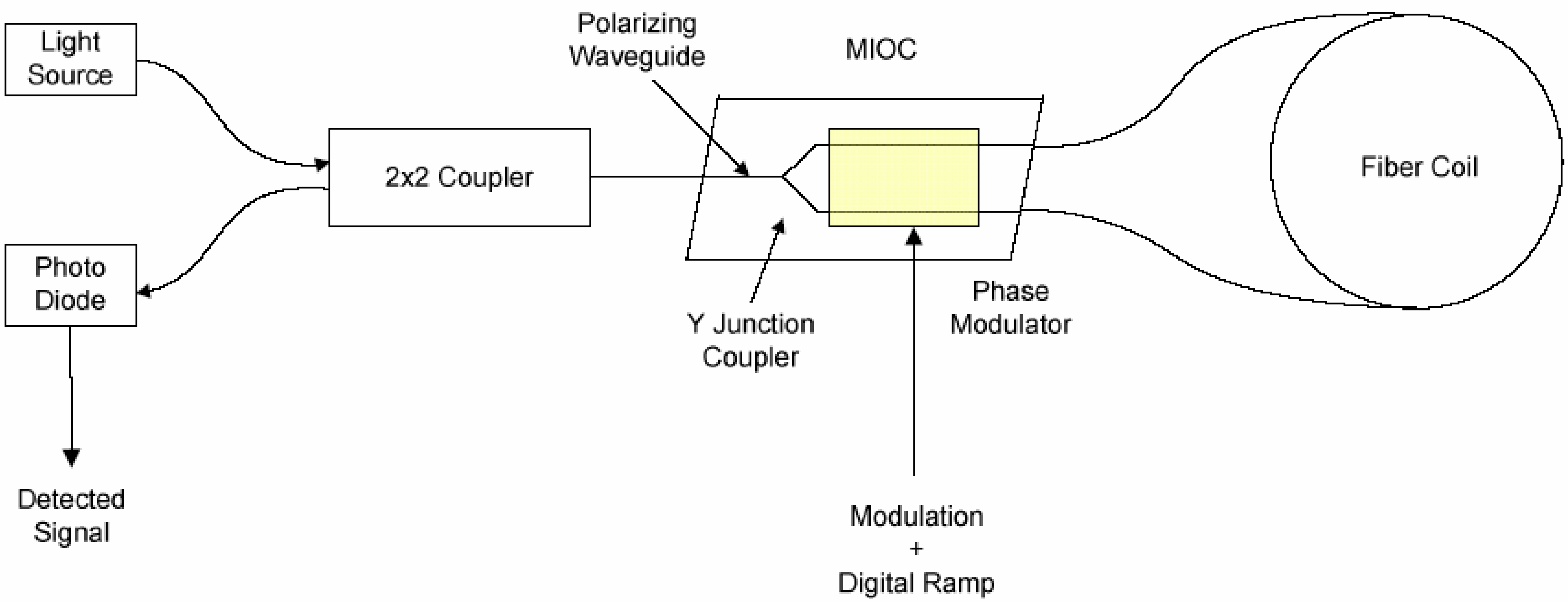
**Spice-alike (Spectre)**

- No way!



Source:
Monica Rafaila, Georg Pelz
Infineon Technologies

# Fiber optical Gyrosensor




This research project (Syena) is supported by the German Government, Federal Ministry of Education and Research under the grant number 01M3178

Bundesministerium für Bildung und Forschung





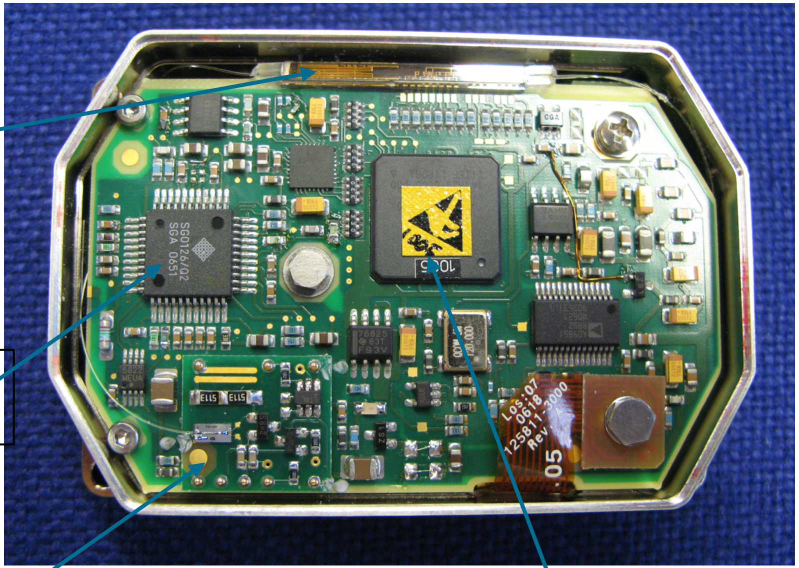
NG LITEF GmbH

Fraunhofer IIS

# Fiber optical Gyrosensor

NORTHROP GRUMMAN

**Sensor Electronic Module (SEM)**



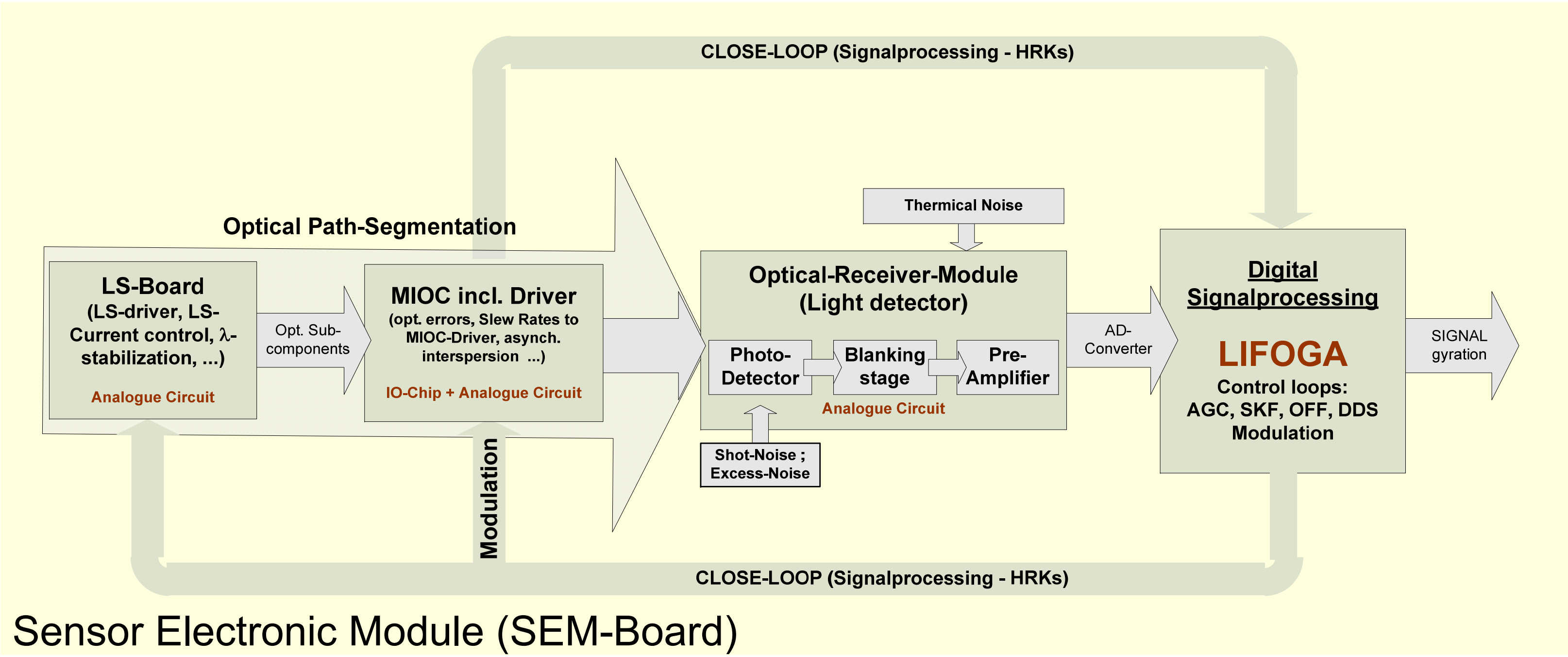

## Northrop Grumman LITEF GmbH




NG LITEF GmbH

SyEnA Fraunhofer IIS

# Fiber optical Gyrosensor




- Interconnection of different physical domains – optic, thermal and electronic
- Interconnection of digital and anlogue hard- and software

Important aspects:

- Dynamic and time resolution
- Device behavior like slew-rates, rise times , overshoots, ...
- Device - environment interaction like temperature to ADC

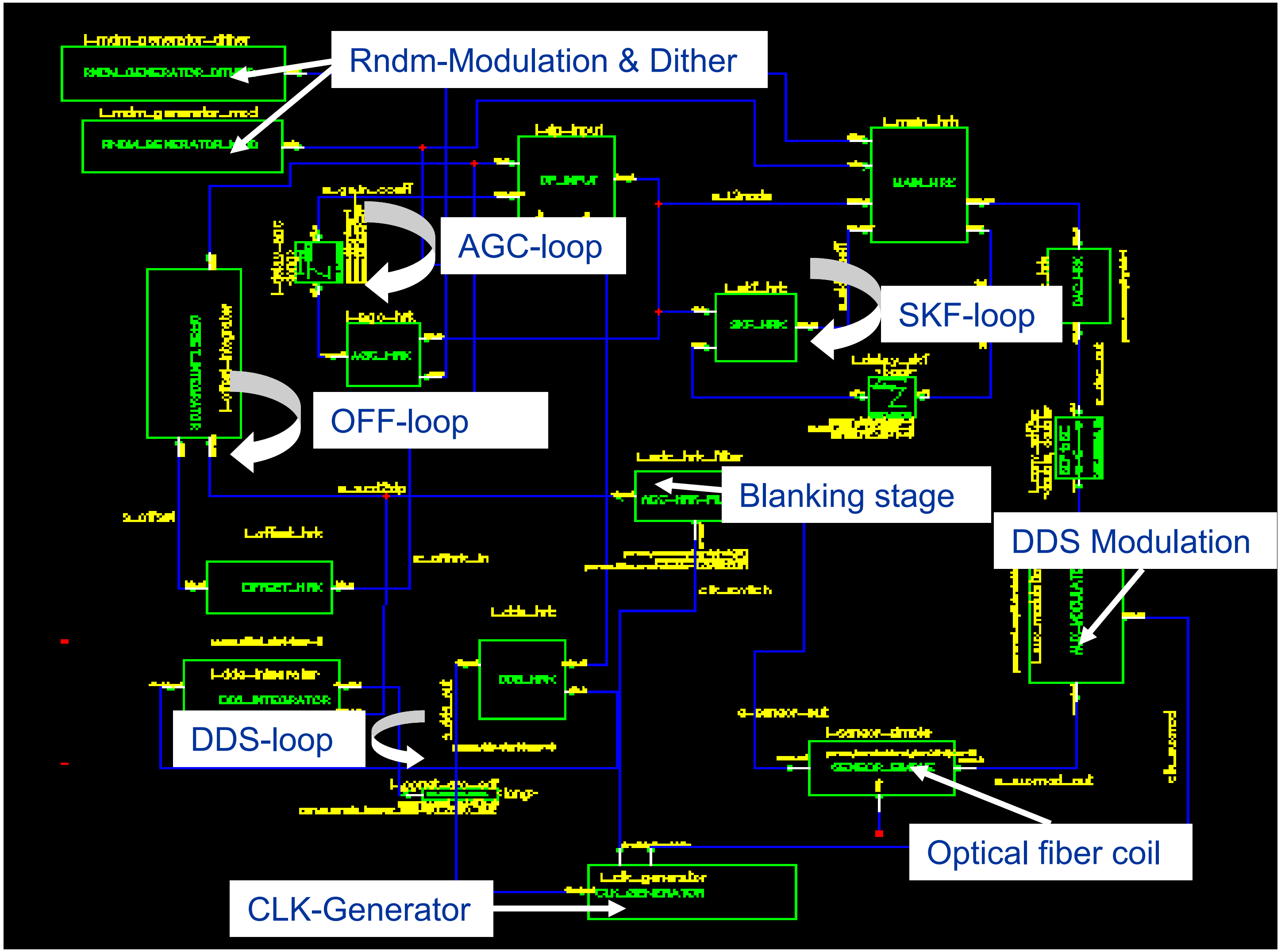
Source:
Andrea Dahlhoff
NG LITEF





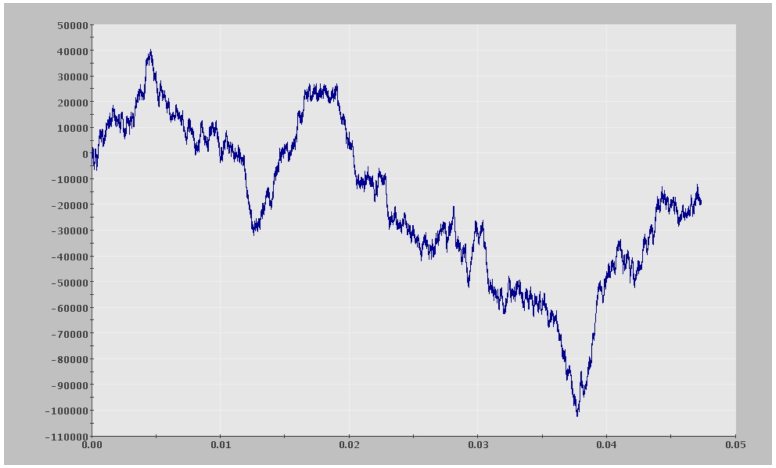
NG LITEF GmbH

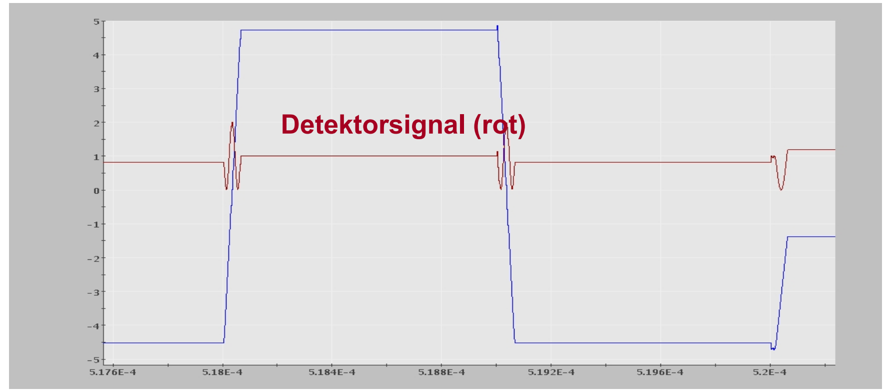
# Toplevel Blockdiagram Systemlevel Model

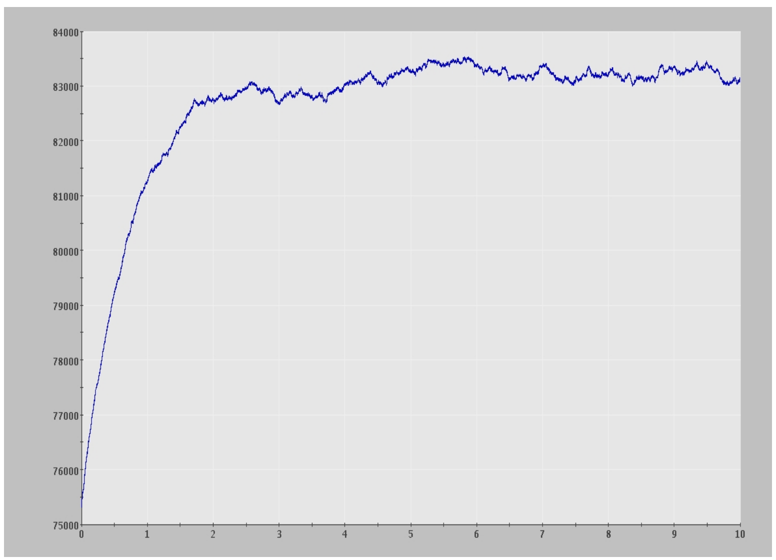


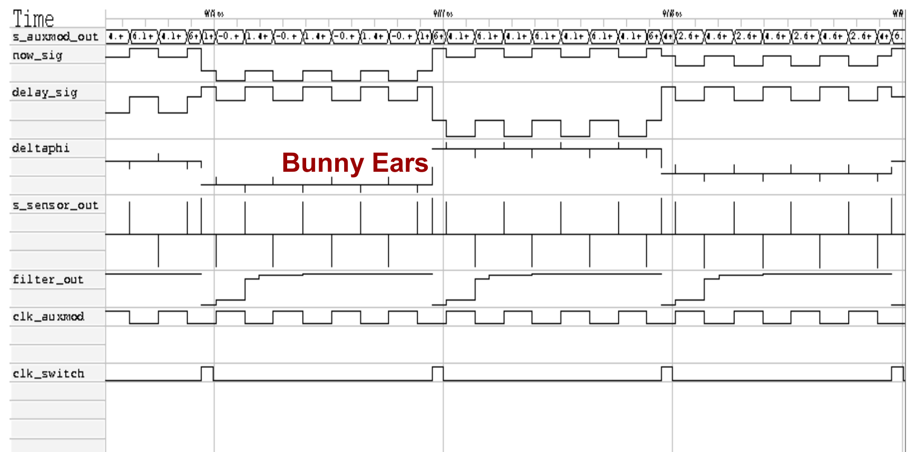

Source:
Andrea Dahlhoff
NG LITEF

# Fiber optical Gyrosensor












NG LITEF GmbH

Fraunhofer IIS

# Conclusion

- SystemC together with the extension SystemC AMS is suitable for creating executable specification, virtual prototypes and architectual level models for EAMS systems
- An experimental prototype can be downloaded under: www.systemc-ams.org (not compatible with the DRAFT 1 standard)
- SystemC AMS DRAFT 1 standard is public available: www.systemc.org
- OSCI SystemC AMS 1.0 standard is expected in December 2009
- Information of the Fraunhofer SystemC AMS activities and documentation: www.systemc-ams.eas.iis.fraunhofer.de