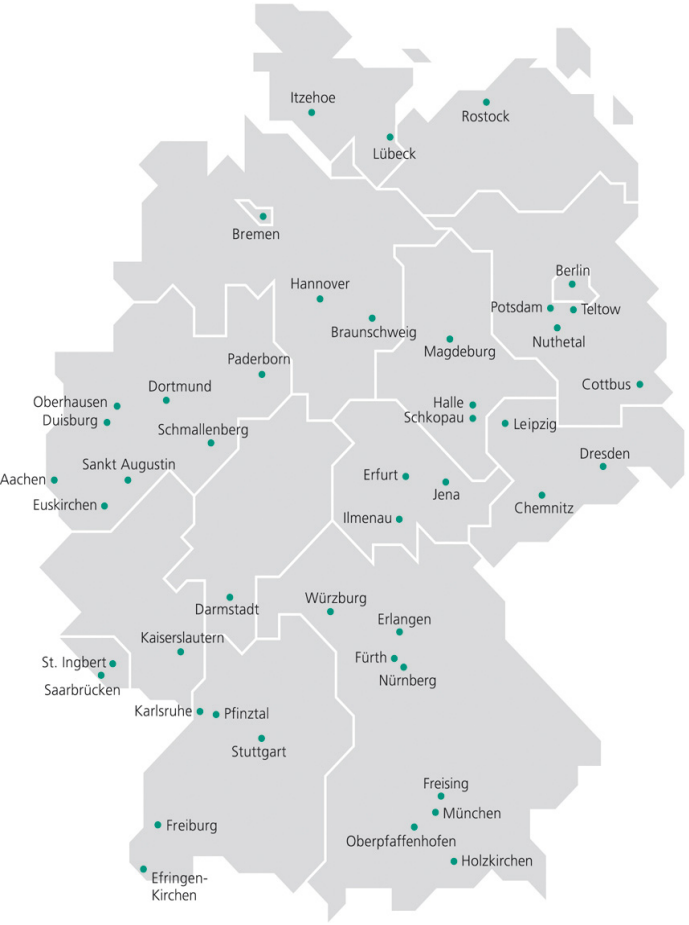
Introduction to SystemC-AMS and the Prototype Simulator

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

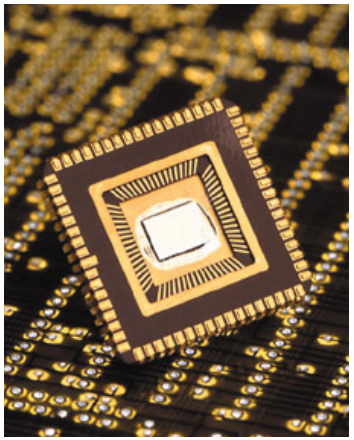
Fraunhofer IIS

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

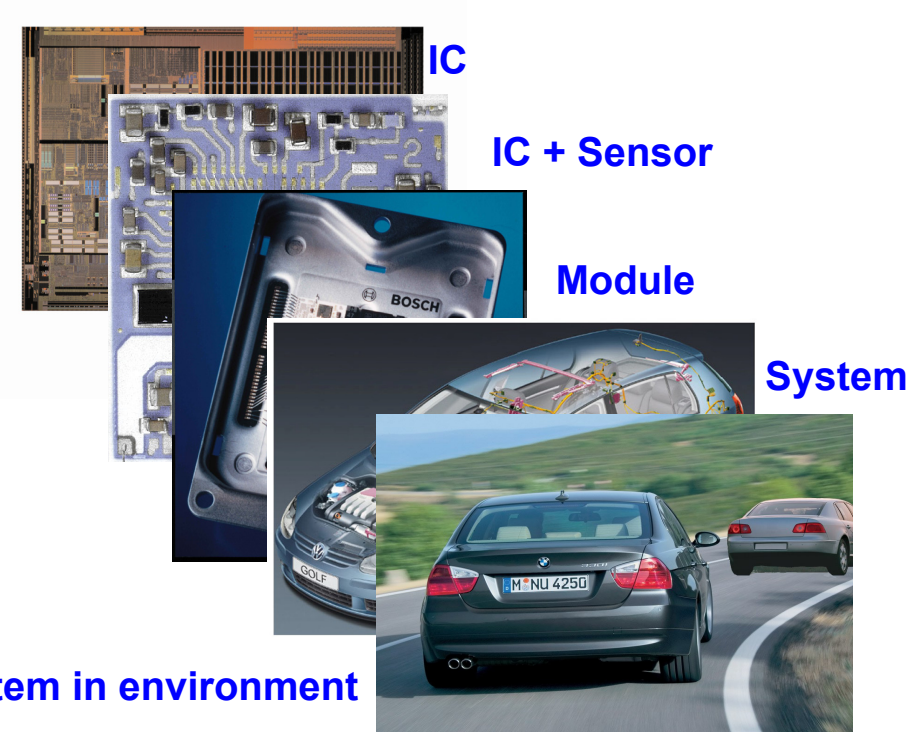
Design Automation Division EAS. Dresden

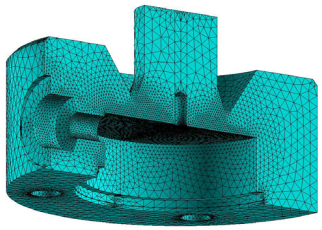
Founded in 1993

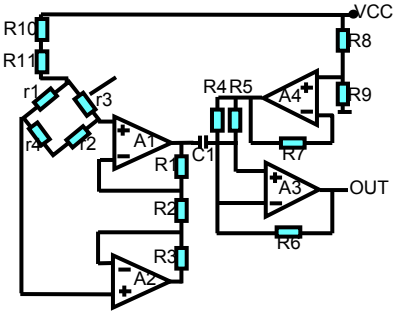
Approx. 80 staff

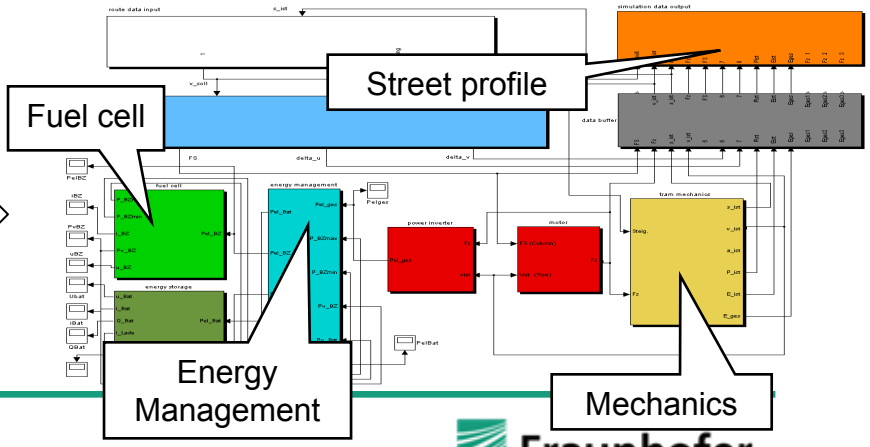
Budget: approx. € 6 million

- 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

SystemC AMS is …

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

SystemC is …

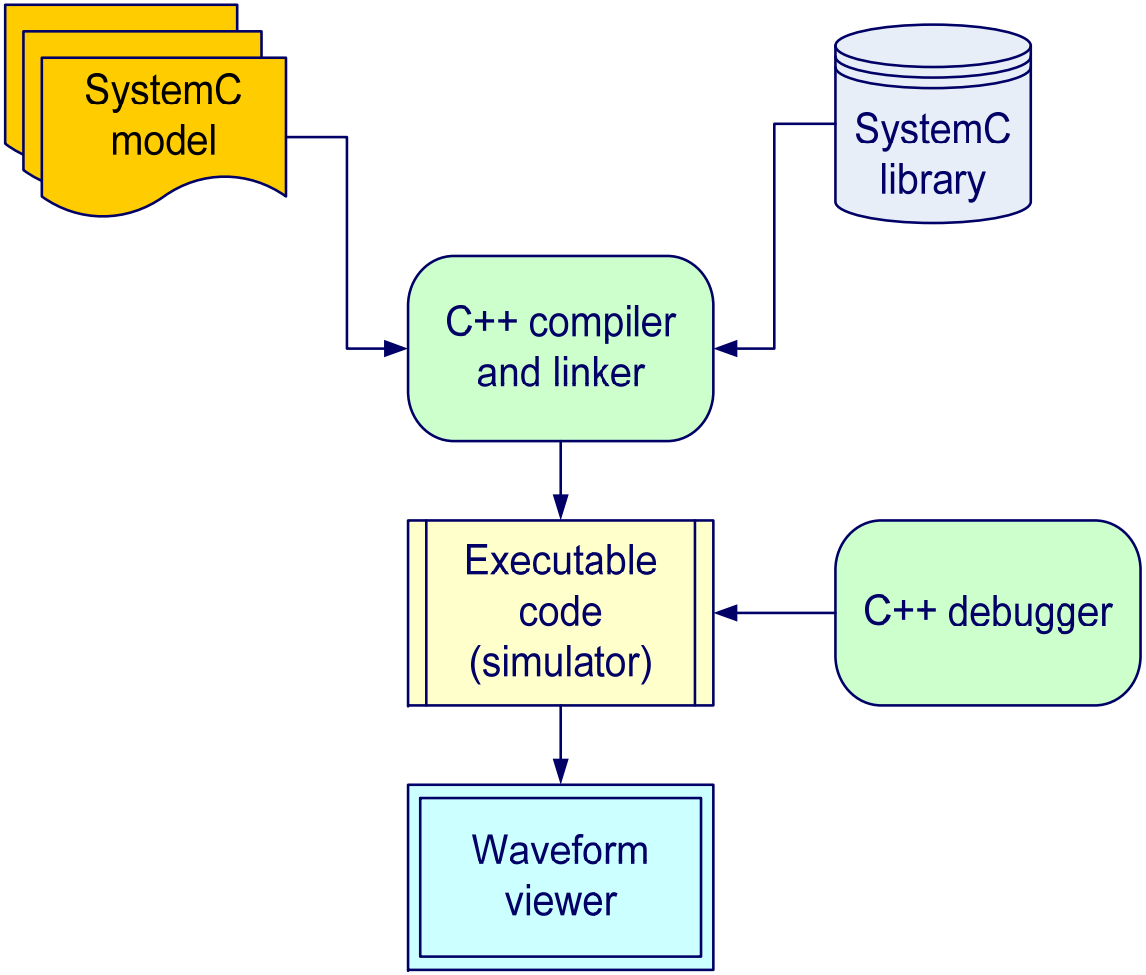
a **definition of C++ language constructs** for the description of complex digital hardware systems on different abstraction levels, using different Models of Computation (MoC)

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

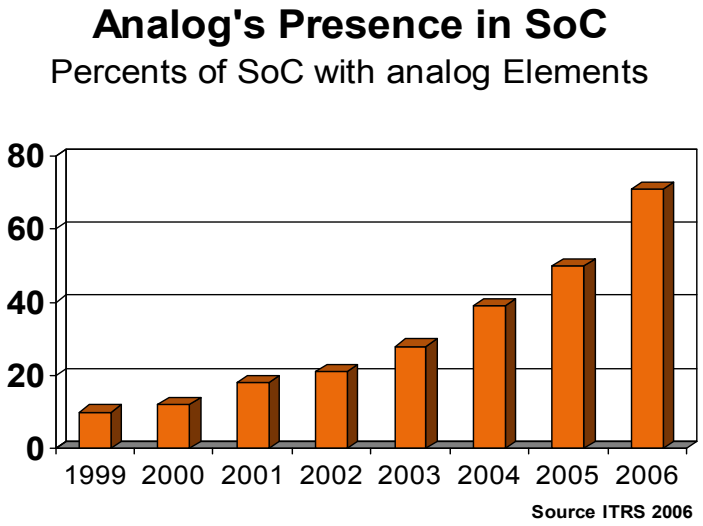
- Best suited for higher abstraction levels like system level design and architectural level exploration
- Easy integration of software

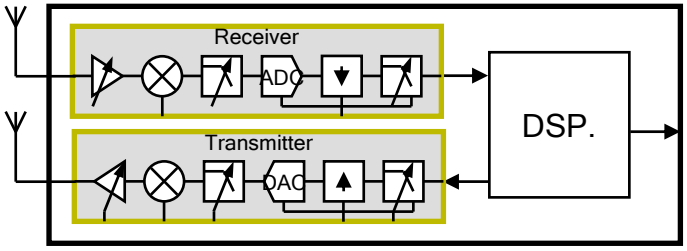
- Support of special methodologies like TLM

Why AMS Extensions for SystemC ?

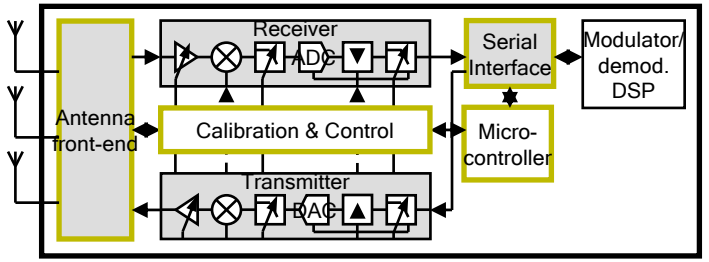
- It's an analog world – analog is how we interact with the real world
- Each digital system is embedded in the analogue world

- 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





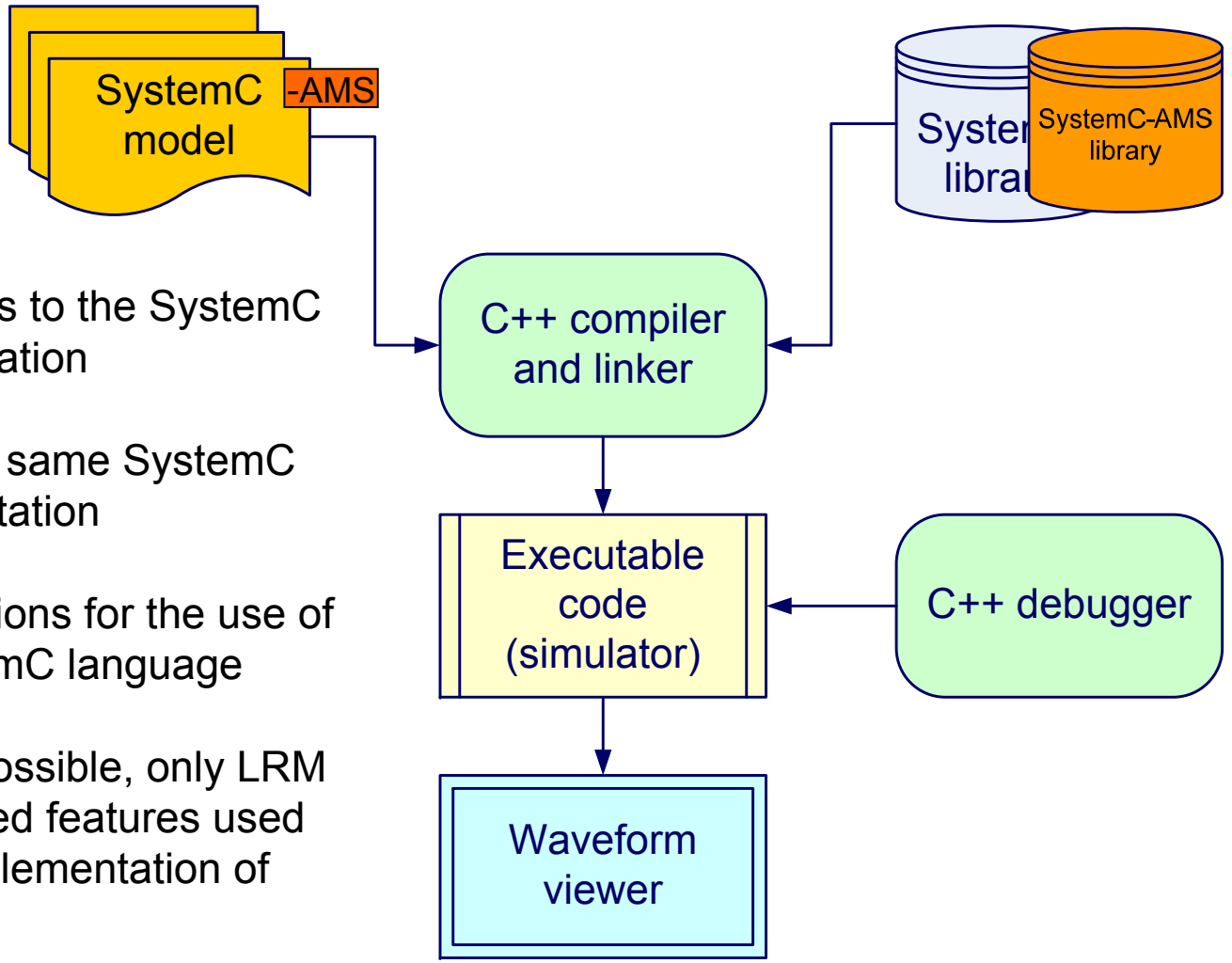
Traditional architecture



Digital assisted analog

Fraunhofer IIS

SystemC-AMS is an extension of SystemC



- no changes to the SystemC implementation

➡ use of the same SystemC implementation

➡ no restrictions for the use of the SystemC language

- 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

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

SystemC AMS Extension Basics

The SystemC AMS Objectives

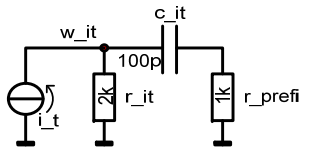
- Unified and standardized modeling approach to design Embedded AMS systems
- AMS model descriptions supporting a design refinement methodology, from functional specification to implementation
- AMS constructs and semantics in a SystemC compatible class library implemented in C++
- Providing an interoperable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

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

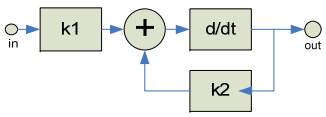
What's different between analog and digital ?

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



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

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

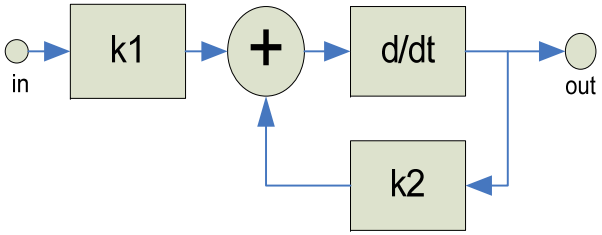


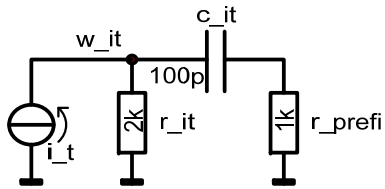
$$out = k1 \bullet \frac{din}{dt} + k2 \bullet out$$

->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?

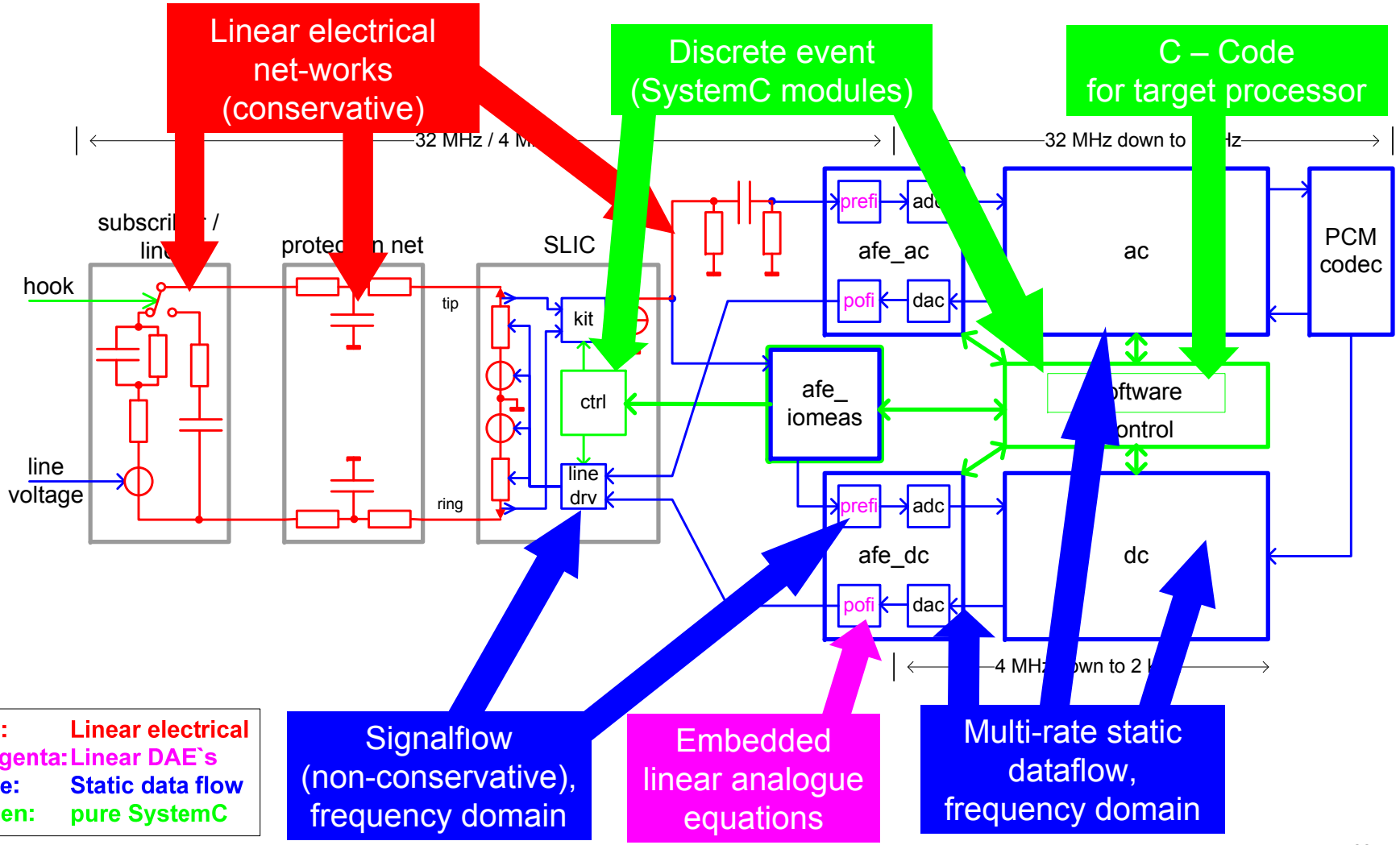
- Modeling on **different abstraction** / accuracy levels yields the possibility to apply specialized algorithms, which are **orders of magnitude faster** than a general approach.

- 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)

Modeling with multiple MoC



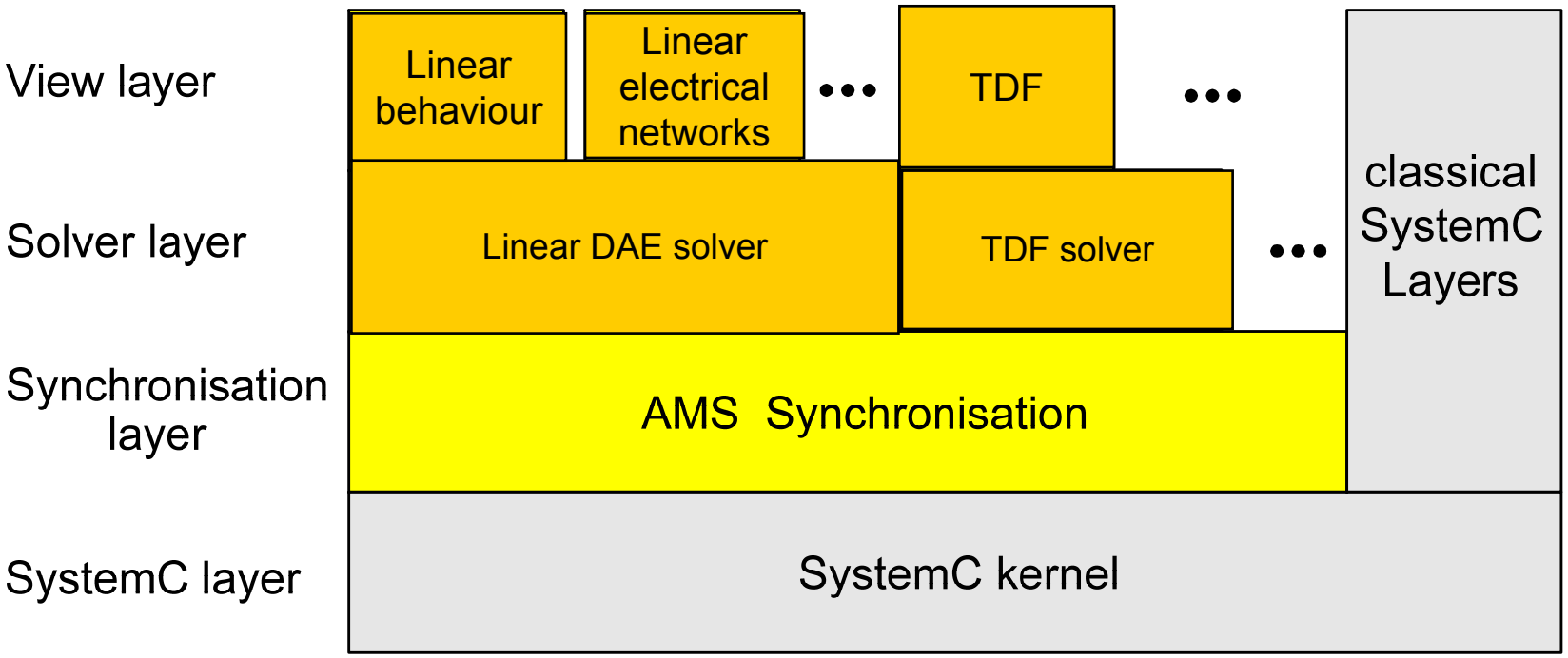
Fraunhofer IIS

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

SystemC-AMS Generic Concept



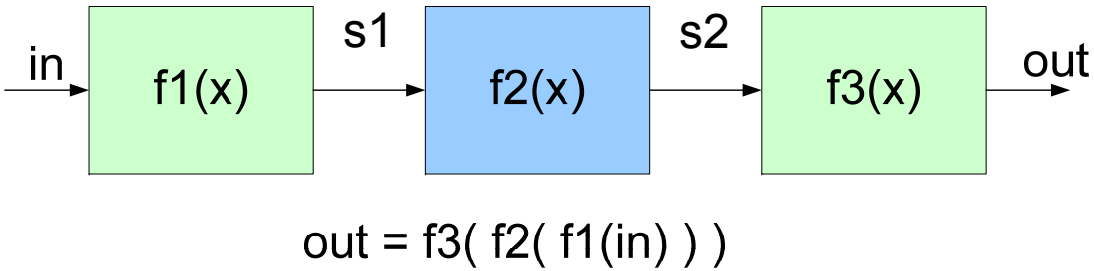
Fraunhofer IIS

SystemC-AMS MoC

- Public prototype 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

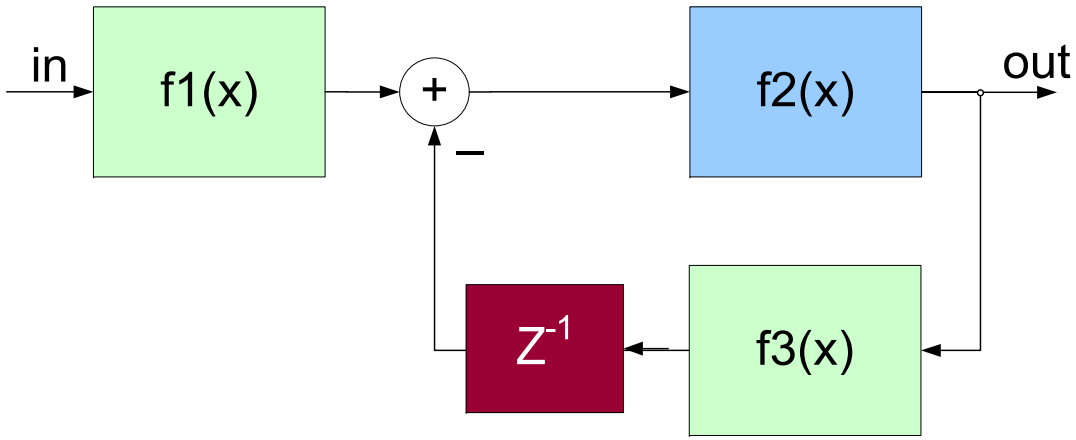


equation system:

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

- 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.

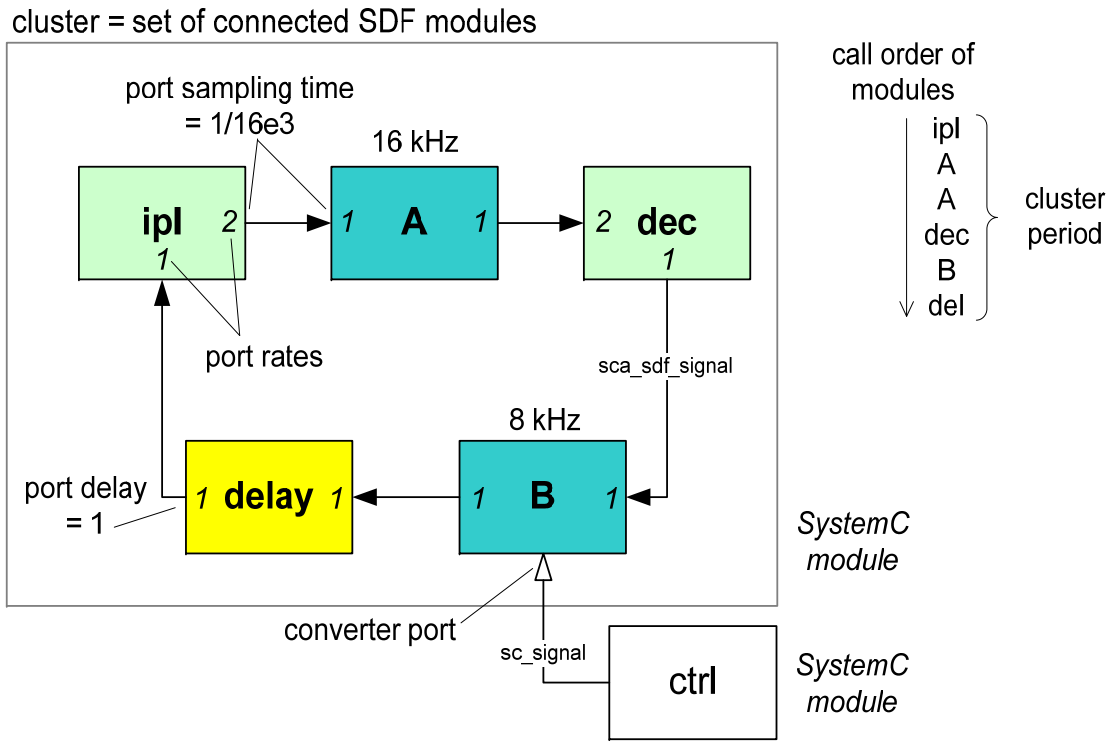
- Thus, at least one delay in the loop is crucial!

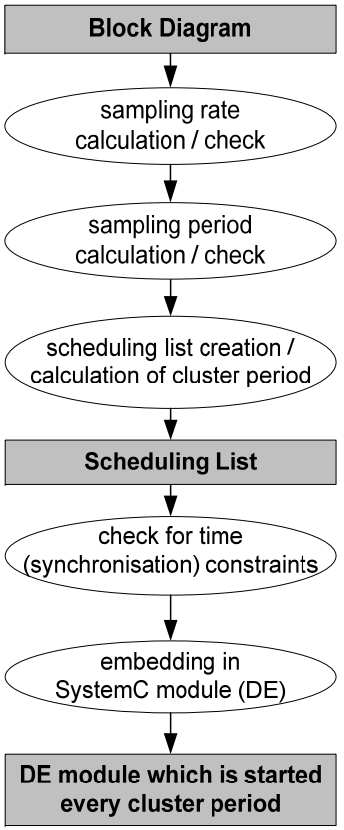
$$\text{out} = f2(\, f1(\text{in}) - f3(\text{out})\,) \longrightarrow \text{out} = f2(\, f1(\text{in}) - f3(\text{out})\, z^{-1}\,)$$

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

Implementation of Multi-Rate SDF in SystemC-AMS

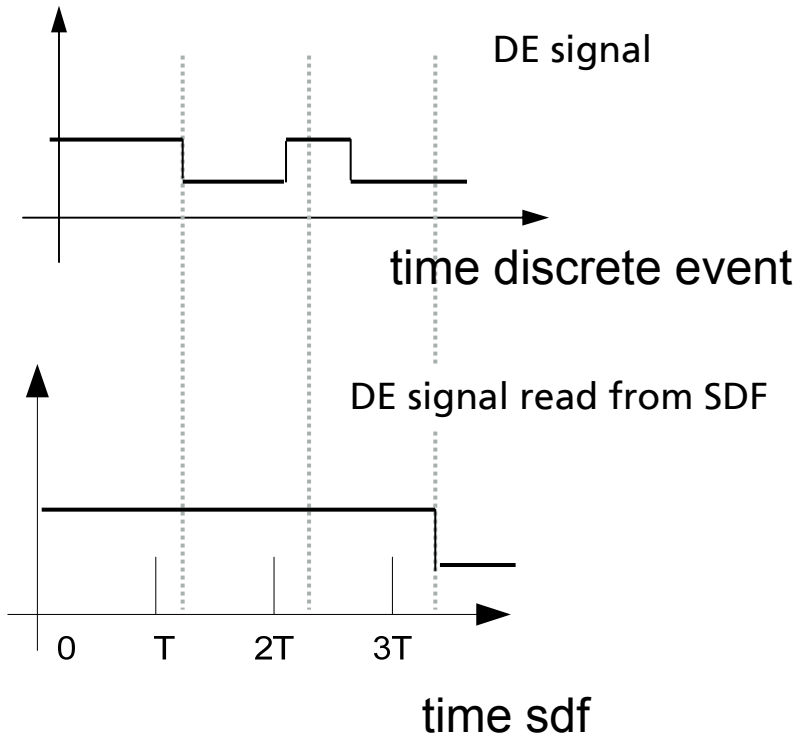




Fraunhofer IIS

Synchronization between SDF and DE

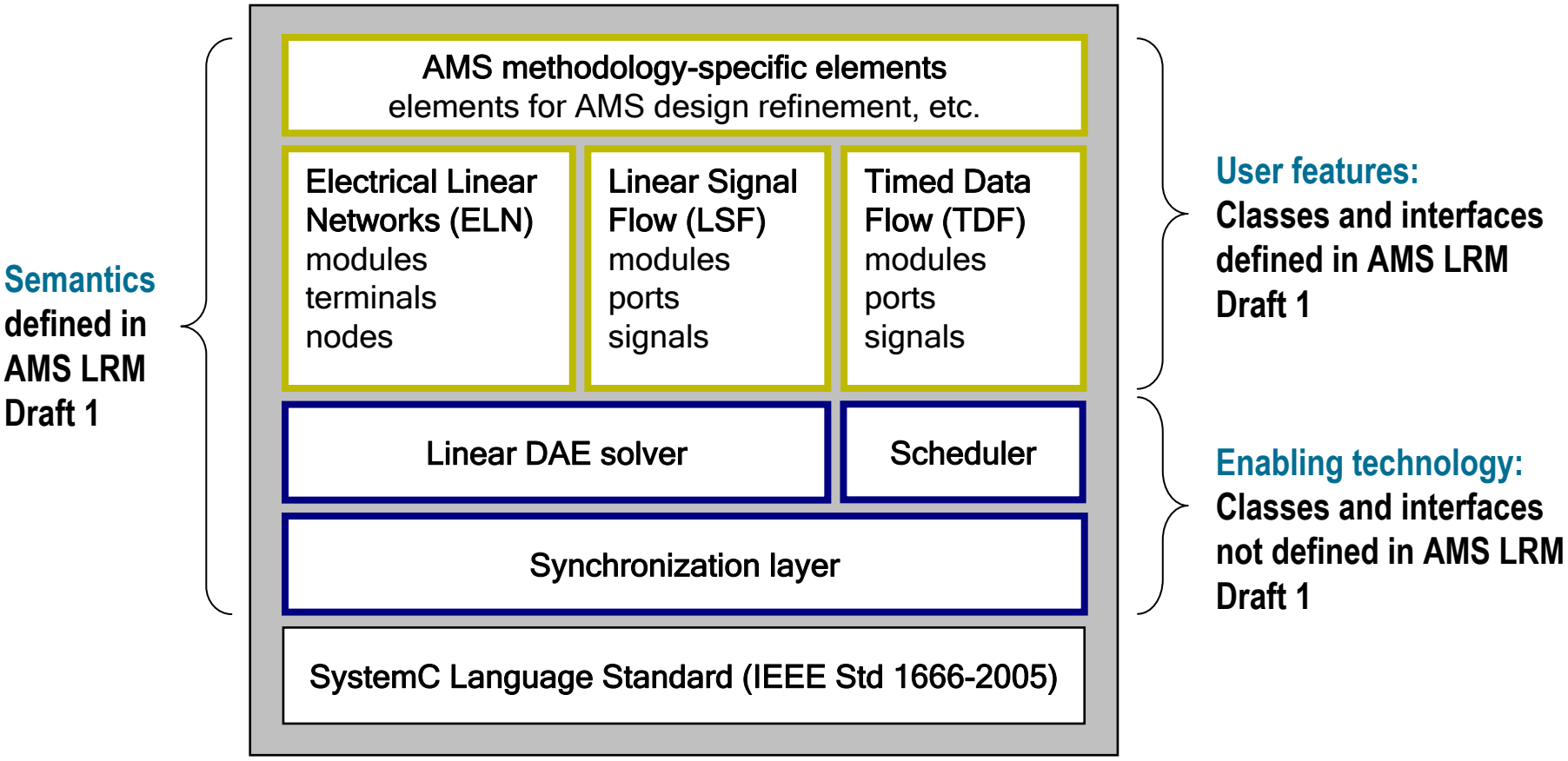
- SDF samples are mapped to sc_time.
- SystemC (DE) signals are sampled at Δ=0 of the specified sampling period. SDF samples are scheduled at Δ=0 as well (and thus valid at least at Δ=1).
- The sampling period T is specified as port attribute and propagated along the SDF signals of the cluster.
- That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



Fraunhofer IIS

AMS Draft Standard / SystemC-AMS Prototype

SystemC AMS extensions LRM Draft 1



Fraunhofer IIS

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_core::sc_module (or SC_MODULE macro)**
- Depending on the MoC, AMS modules are pre-defined or user- defined

SystemC AMS base Language Element Composition - New in DRAFT1

- 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

SystemC AMS Language Element Composition – Converter – New in DRAFT1

- 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> >
```

Basic SDF Modeling Constructs –Old Prototype

- Module declaration macros
- Port declarations dataflow ports
- Converter ports to connect SystemC-signals (the SystemC signals will be sampled)
- Virtual primitive dataflow methods, called automatically by schedule
- Constructors (currently the same like SC_CTOR – however use SCA_CTOR for compatibility with future extensions)

```
SCA_SDF_MODULE(<name>)

sca_sdf_out < <type> >  // outport
sca_sdf_in  < <type> >  // inport

sca_scsdf_in  < <type> >
sca_scsdf_out< <type> >


void attributes()          // before elaboration
void init()                // before simulation start
void sig_proc()            // during simulation
void post_proc()       // after simulation

sc_time sca_get_time()   //gives the absolute time of
                           //the current module call


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


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)
```

Attributes of SDF - Ports

- Rate is the number of samples which has to be read / written per module (sig_proc()) invocation, the default value is 1, has to be set in attributes().
- Delay sets the number of samples which has to be written during initialisation, this is the number of sample delay or the number of sampling periods delay, the default value is 0, has to be set in attributes().
- T is the time distance between samples, T has to be set at least for one port per dataflow cluster, default is unset, has to be set in attributes().
- get_T() returns the analysed sampling time, may be called earliest in init().

```
port.set_rate(2);


outport.set_delay(1);
port.set_T(double, sc_time_unit);
port.set_T(sc_time);


port.get_T();  // get analysed sample
               // period as sc_time
port.get_T().to_seconds();  // in seconds

port.get_rate();     //get set rate as long
port.get_delay();  //get set delay as long
port.get_time();   //get absolute time of the
                   //first sample
```

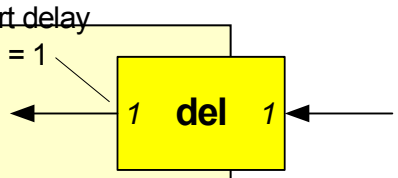
Example: TDF language constructs

```
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) {}
};
```

Syntax Example of SDF Module – Old Prototype

- module declaration
- port declarations
- set port attributes
- Initialisation
- process method
- post processing
- constructor

```
SCA_SDF_MODULE(delay_sdf)
{

  sca_sdf_in<double>  in;
  sca_sdf_out<double> out;


  void attributes() {
                        out.set_delay(1);
                      }

  void init() {
                  out.write(0.0); // initial value
                }


  void sig_proc()  {
                       out.write(in.read());
                      }

  void post_proc()  {
                         cout << name << " done" << endl;
                       }


  SCA_CTOR(delay_sdf){ }
};
```



Fraunhofer IIS

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

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

- State Space equations

Fraunhofer IIS

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) – ***New in DRAFT 1***
  - 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

Linear Signalflow (LSF) Modeling – *New in DRAFT 1*

- 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 - input port
  - sca_lsf::sca_out - output port
- Channel / Signal:
  - sca_lsf::sca_signal

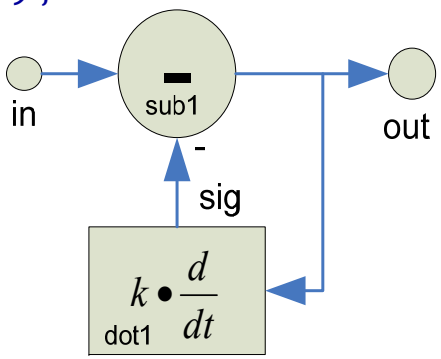
LSF predefined modules - *New in DRAFT 1*

- 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

  mylsfmodel(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

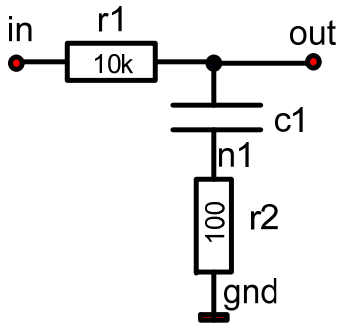
- Ports:
  - sca_eln::sca_terminal - conservative terminal
- Channel / Node:
  - sca_eln::sca_node – consevative node
  - sca_eln::sca_node_ref – reference node, node voltage is always zero

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);
  }

};
```

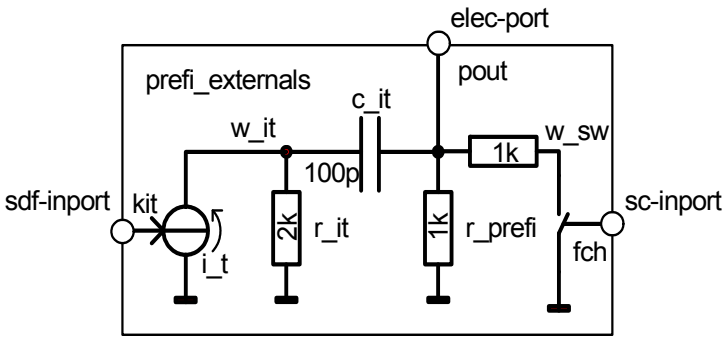


Fraunhofer IIS

Conservative MoC – Linear Network – Old Prototype

```
SC_MODULE(prefi_externals)
{
  // synchronous dataflow inport
  sca_sdf_in<double>   kit

  // connect with sc_signal<bool>
  sc_in<bool> fch;

  // electrical port
  sca_elec_port   pout;

  // internal nodes declaration
  sca_elec_node w_it, w_sw;
  sca_elec_ref     gnd;

  // component declarations
  sca_r             *r_it, *r_prefi, *r_prefi2;
  sca_c             *c_it;
  sca_sdf2i         *i_t;
  sca_sc_rswitch  *sw_prefi;
```

```
SC_CTOR(prefi_externals)
 {
    i_t = new sca_sdf2i("i_t");
       i_t->p(gnd);
       i_t->n(w_it);
       i_t->ctrl(kit);

    r_it = new sca_r("r_it");
       r_it->p(gnd);
       r_it->n(w_it);
       r_it->value=2.0e3;

    …
 }
};
```



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(…)

Tracing of analog Signals

- SystemC AMS has a own trace mechanism:
  - Analog / Digital timescales are not always synchronized
  - Note: 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 – *New***

- Features to reduce amount of trace data:
  - enable / disable tracing for certain time periods, redirect to different files
  - different trace modes like: sampling / decimation

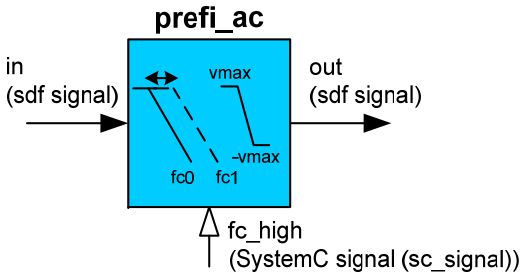
Code example

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;
  }
};
```

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



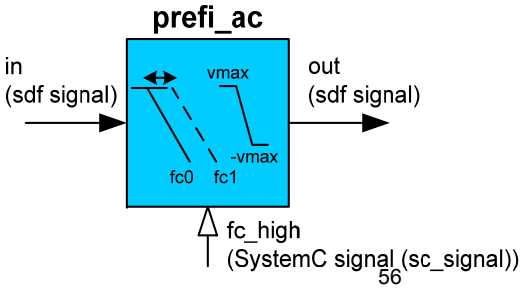
Fraunhofer IIS

SDF Module – Example with LTF – Old Prototype

```
SCA_SDF_MODULE(prefi_ac)
{
   sca_sdf_in<double>    in;   // signal inport
   sca_sdf_out<double> out;  // signal outport

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

   double fc0, fc1;    // cut-off frequency
   double v_max;     // max. out value

   sca_ltf_nd  ltf_0, ltf_1;       // filter equation obj.
   sca_vector<double>  a0, a1, b;
   sca_vector<double>  s;      // state vector

   void init() // filter coeffs for transfer function
   {
      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 sig_proc() {
     double tmp;    // high or low cut-off freq.
     if(fc_high.read())    tmp = ltf_1(b, a1, s, in.read());
     else                  tmp = ltf_0(b, a0, s, in.read());

     if (tmp >  v_max)    tmp =   v_max;  //output voltage
     else if (tmp < –v_max)    tmp = –v_max;  // limitation

      out.write(tmp);  // assign output voltage to port
  }

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

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

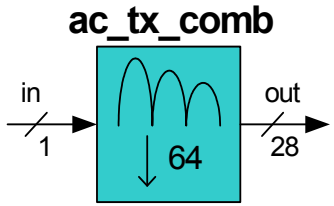


Fraunhofer IIS

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)
  {
      …
  }
};
```



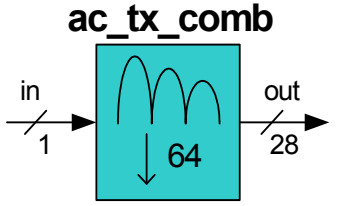
$$H(z)=\left(\frac{1-z^{-k}}{1-z^{-1}}\right)^{n} \qquad z=e^{j2\pi f/f_s}$$

Fraunhofer IIS

Frequency Domain Specification – Old Prototype

```
SCA_SDF_MODULE(ac_tx_comb)
{
   sca_sdf_in<bool>              in;
   sca_sdf_out<sc_int<28> > out;

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

   void ac_sig_proc()
   {
     double        k  = 64.0;    // decimation factor
     double        n  =  3.0;    // order of comb filter

     sca_complex z1 =  sca_ac_z(in.get_T().to_seconds(), -1);

      // complex transfer function:
     sca_complex h = pow((1.0 – pow(z1,k)) / (1.0 – z1), n);

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

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

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



$$H(z)=\left(\frac{1-z^{-k}}{1-z^{-1}}\right)^{n} \qquad z=e^{j2\pi f/f_s}$$

Fraunhofer IIS

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 !

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
}
```

Parameters / Generics Handling

Issues:

- Hierarchical descriptions must be instantiated in the Constructor (SystemC – rule)
- Parameters which influence the elaboration must be available in the constructor – except for sdf/tdf-modules there the void attributes() method is called before elaboration

Solutions:

- Two principal possibilities – public member variables and Constructor parameters
- Member variables can be easier handled than Constructor parameters
- Member variables cannot be used for hierarchical descriptions and primitive parameter which influence the elaboration (except sdf)

Recommended Coding Style for Parameter

```
SC_MODULE(comp)
{
  sc_in<bool> p1;

  struct params
  {
    long threshold1, threshold2;

    params() //optional defaults
    {
      threshold1=12;
      threshold2=13;
    } };

   void access_demo() {  long a=p.threshold1; … }
            :
  SC_HAS_PROCESS(comp) // !!! important !!!!
  comp(sc_module_name nm, params par=params()) :
                                                  p(par)
  {
            :
  }
 private:
   params p;
};
```

```
comp::params c1_params;
   c1_params.threshold1=15;
c1=new comp("c1",c1_params);
   c1->p1(s1);
```

Advantages:

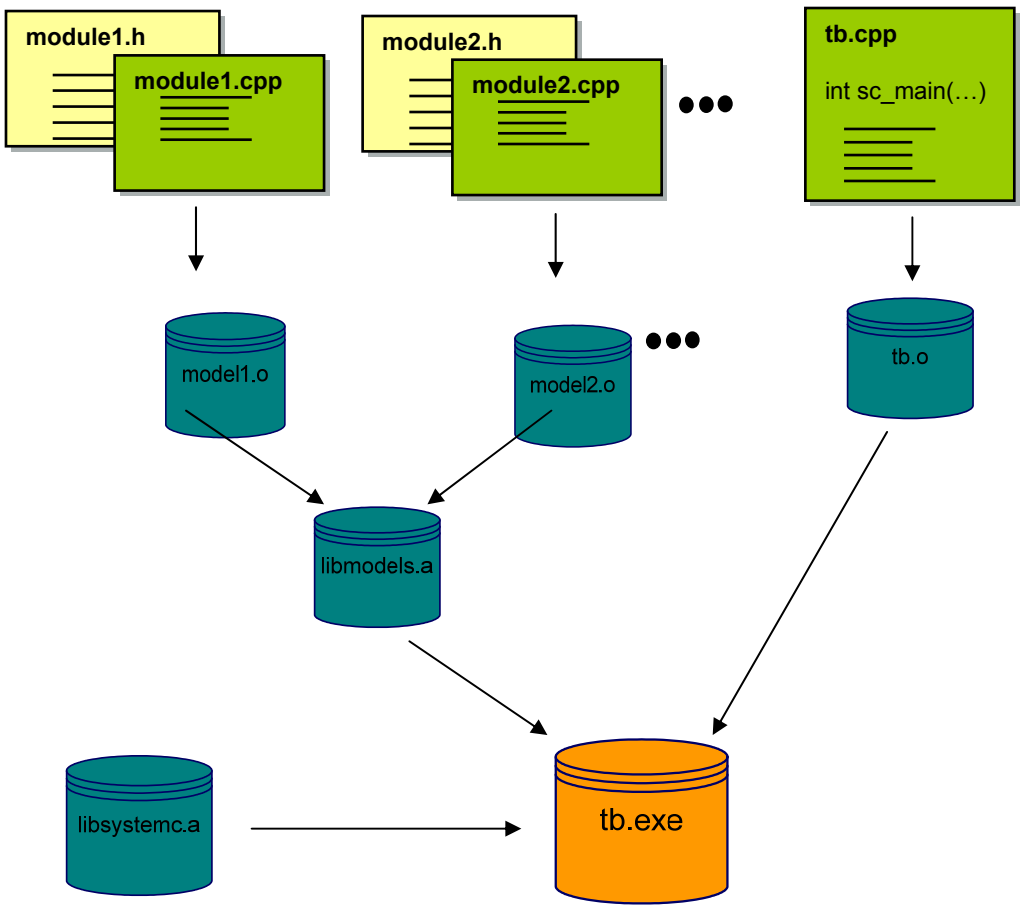
- Easy default value definition/handling
- Named association
- Easy handling of numerous parameters

Getting Started

1. Download systemc from www.systemc.org and the old SystemC-AMS prototype from www.systemc-ams.org
2. Install the libraries for Linux, Solaris, Windows/MinGW or Windows/cygwin (all 32 Bit) , gcc > 3.x required (read readme for details)
   - tar –xzvf systemc.tar.gz
   - cd systemc
   - configure
   - make; make install
   - setenv SYSTEMC_PATH <your install dir> -> may insert in your .cshrc

   - tar –xzvf systemc_ams.tar.gz
   - cd systemc_ams
   - configure
   - make; make install
   - setenv SYSTEMC_AMS_PATH <your install dir> -> may insert in your .cshrc

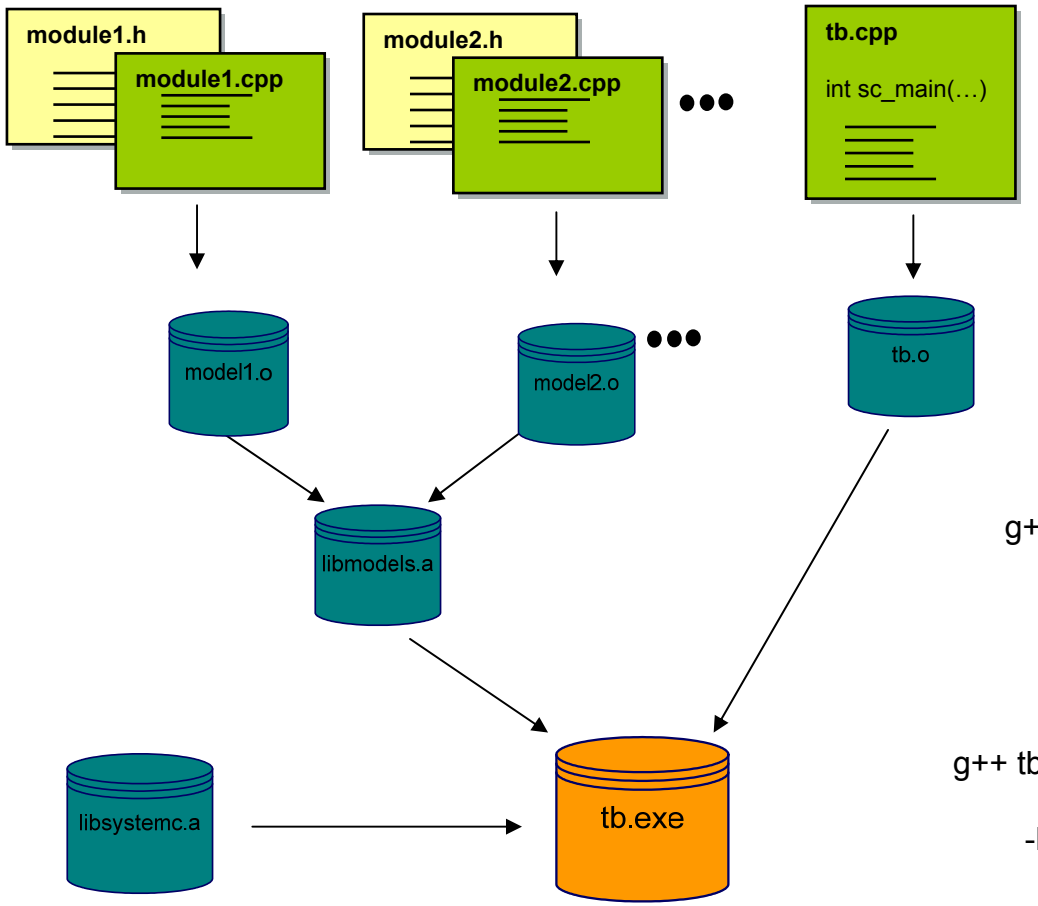
Getting Started SystemC Files



Recommendations:

- Split the module description into a header and a cpp implementation file
- Only one module per header / cpp file
- The name of the module shall be equal to the header / cpp file name
- Do not use capital letters and special characters (like ä,%,&, space, …)

SystemC / SystemC-AMS Compilation




```
g++ -c module1.cpp –I${SYSTEMC_PATH}/include \
                   –I${SYSTEMC_AMS_PATH}/include

g++ -c module2.cpp –I${SYSTEMC_PATH}/include \
                   –I${SYSTEMC_AMS_PATH}/include

ar –rcs libmodels.a module1.o module2.o

g++ -c tb.cpp –I${SYSTEMC_PATH}/include \
              –I${SYSTEMC_AMS_PATH}/include
```

uppercase i

```
g++ tb.o –L${SYSTEMC_PATH}/lib-{TARGET_ARCH} \
         –L${SYSTEMC_AMS_PATH}/lib-{TARGET_ARCH}
    -L. –o tb.exe –lmodels –lsystemc-ams -lsystemc
```

lowercase L !!! Library order Important !!!

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