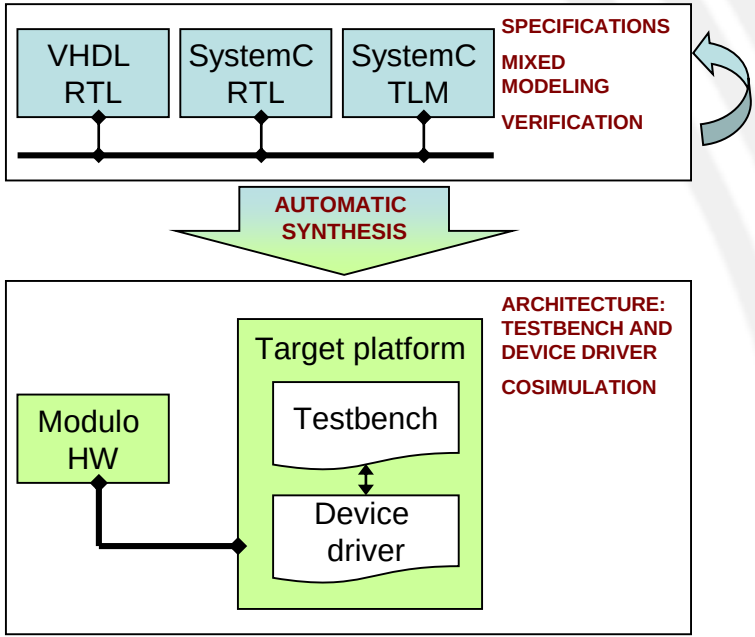
ESD Electronic Systems Design
Synthesis Verification Testing Power Communication

Modeling analog and Continuous Behaviors: SystemC-AMS

Stefano Centomo

Course introduction: Lab topics

- **Emulate the design flow of a real embedded/cyber-physical system**
- **Specification**
  - Compilation/execution/debugging of SystemC code
  - Modeling of SystemC at RTL level
  - Modeling of SystemC at TLM level
  - Time evolution in SystemC
  - Modeling analog and continuous behaviors: SystemC-AMS
  - Components integration
  - Mixing SystemC RTL, TLM and AMS



- **SW Synthesis**
  - Platforms, testbenches and drivers
  - Model Based Design

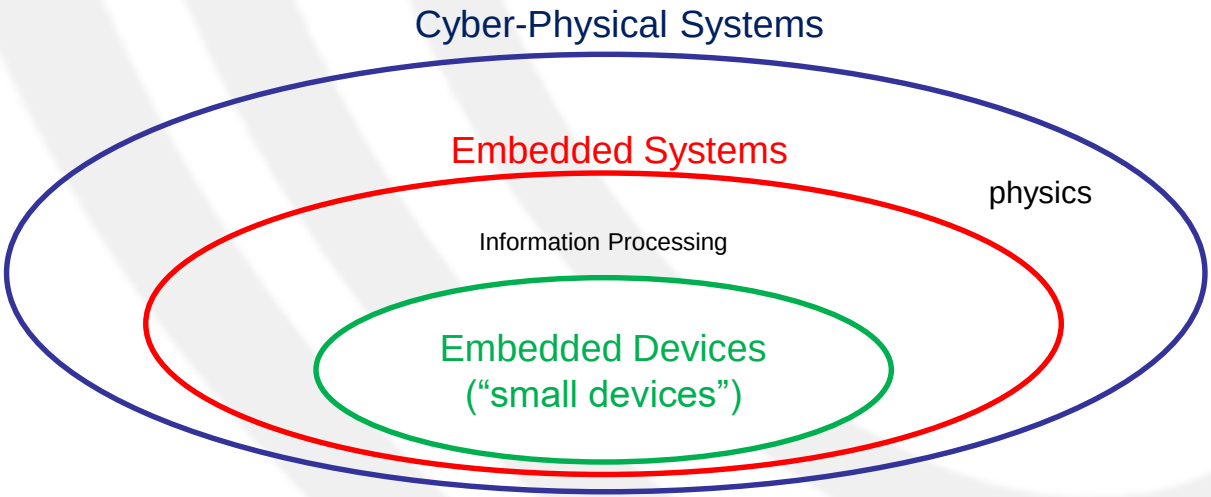
- **HW Synthesis**
  - VHDL modeling at RTL
  - Automatic Synthesis from TLM descriptions
  - Automatic Synthesis of RTL VHDL code

Overview

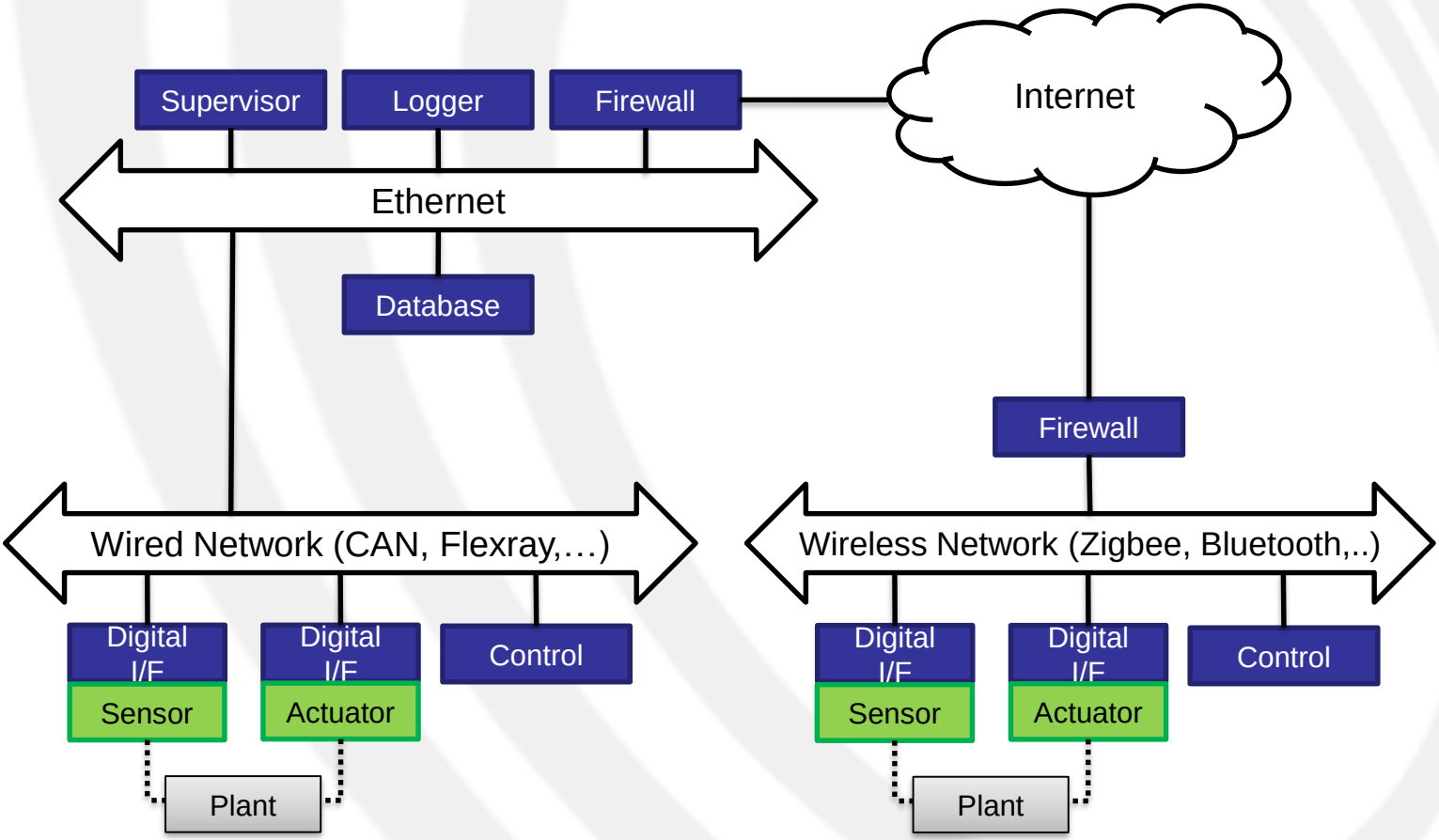
Cyber-Physical Systems

- **Embedded Systems (ES)**: information processing embedded into a large product [Marwedel, 03]
- **Cyber-Physical Systems (CPS)**: integration of computation with physical processes [Lee, 06]

CPS = ES + physical environment



What a Cyber-Physical System is



SystemC-AMS

- Extension of the SystemC language standard
  - System-level modeling for **Analog Mixed-Signal Systems**
  - Introduction of new execution semantics
  - Standardized by Accellera
    - Current standard version: 2.1
  - Standardized by IEEE in 2016: **IEEE 1666.1-2016**

- Methodologies for Mixed-Signal Systems:
  - Executable specifications
  - Virtual prototyping
  - Architecture exploration
  - Integration validation

Use cases

- Executable specification
  - Verify correctness of system requirement using simulation
- Virtual prototyping
  - High-level (untimed/timed) model of HW architecture
- Architecture exploration
  - Evaluate mapping between behavior and system architecture
- Integration validation
  - Verify the correctness of integrated components

Discrete-time vs. Continuous-time descriptions

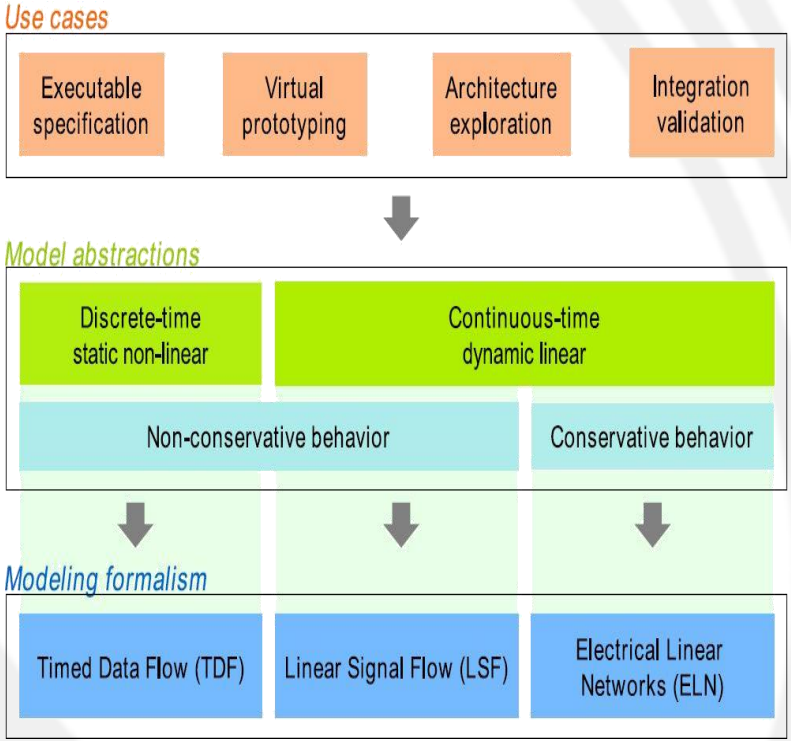
- Discrete-time descriptions:
  - Signals and physical quantities defined at discrete time points
    - Assumed constant between time points
  - Behavior as procedural assignments involving sampled signals
  - Well suited for describing signal-processing dominated behaviors
    - Signals are naturally (over)sampled
  - Used also for describing (approximation of) continuous-time behaviors
- Continuous-time descriptions:
  - Signal and physical quantities described as real-valued functions of time
    - Time considered as a continuous value
  - Behavior as Differential algebraic equations (DAE) or Ordinary differential equations (ODE)
    - Solved by linear or non-linear solver (complex algorithms)
  - Well suited for describing physical behaviors of dynamic systems

Non-conservative vs. conservative descriptions

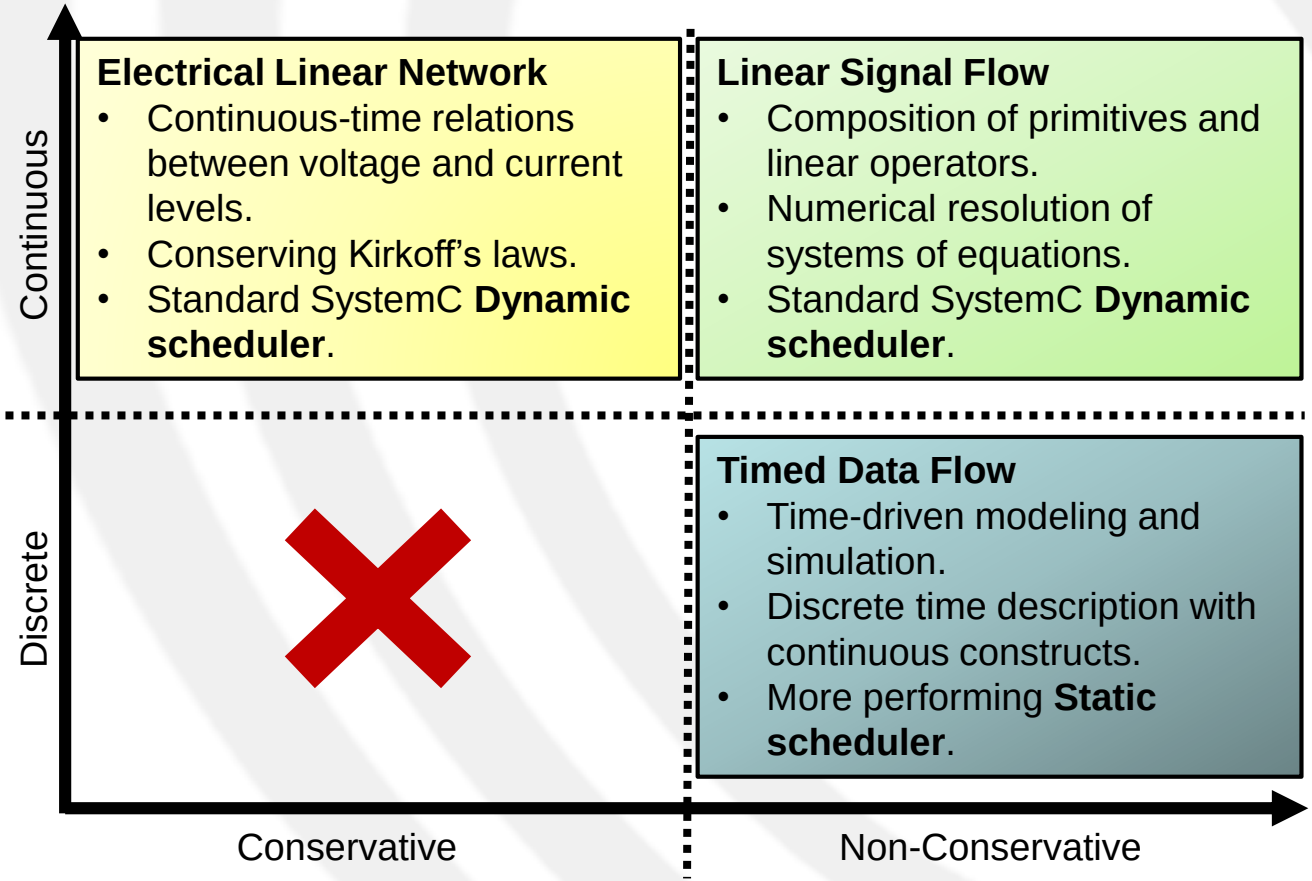
- Non-conservative descriptions:
  - Behavior expressed as directed flows of continuous time signals or variables
    - Processing functions (e.g., filtering or integration) are applied
  - Non-linear dynamic can be described
  - Mutual effects and interaction between AMS componets not supported
    - E.g., impedances or loads
- Conservative descriptions:
  - Based on satisfaction of Kirchhoff's laws
  - Huge sets of equations to solve are inferred
    - Computational hard

Modeling formalisms

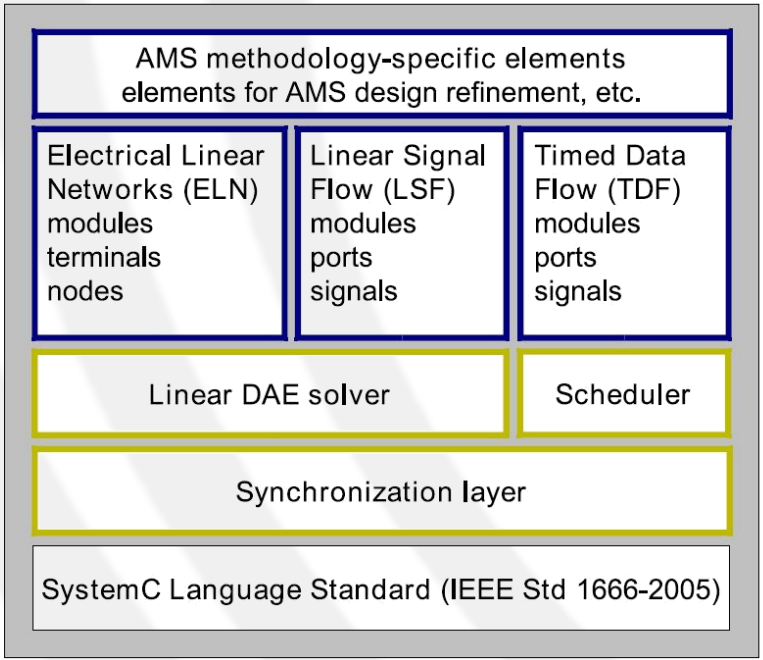
- Timed Data Flow (Discrete Event Models)
  - Discrete time, non-conservative modeling
  - Static scheduler (based on dataflow)
- Linear Signal Flow (LSF)
  - Continuous, non-conservative modeling
  - DAE and ODE based
    - Represented by connection of primitives for real-valued time-domain signals
  - Symbolic and numeric solvers
- Electrical Linear Network (ELN)
  - Continuous, conservative modeling
  - Modeling of electrical networks
    - Linear network primitives (e.g., resistors, capacitor etc…)
    - Continuous relations between voltage and currents



SystemC-AMS: Modeling formalisms



Language Architecture



Semantics defined in the AMS language standard

User features
Classes and interfaces defined in the AMS language standard

Enabling technology
Classes and interfaces *not* defined in the AMS language standard
(implementation defined)

Discrete-time non-conservative MoC:

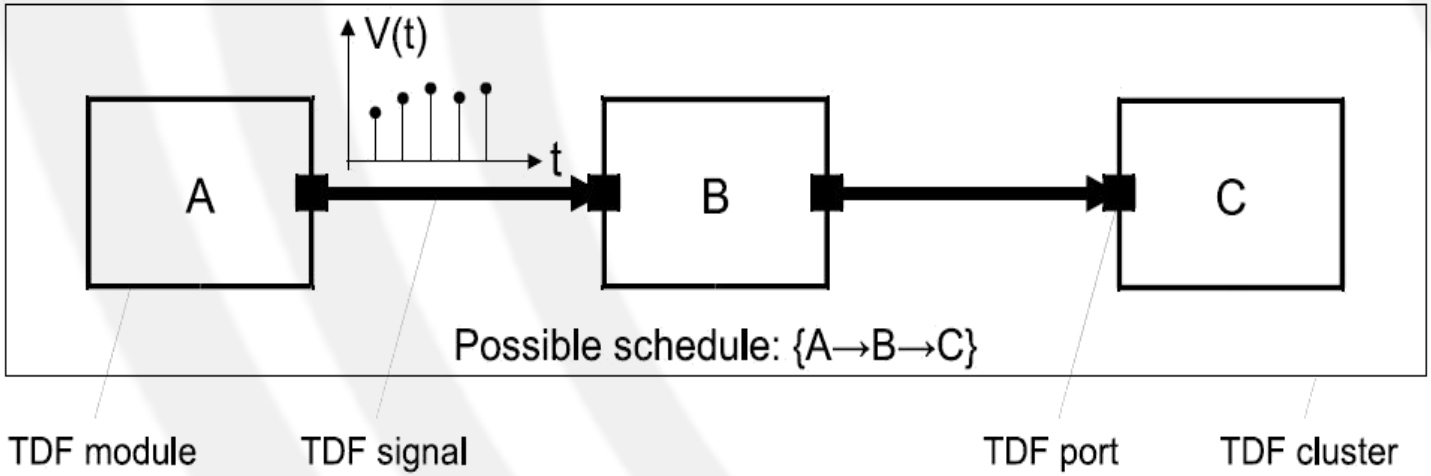
Time Data Flow

Fundamentals

- Based on Synchronous Data Flow (SDF)
  - SDF untimed, Discrete Event Models discrete-time
- Each Discrete Event Models module contains a C++ method
  - Computes a mathematical function
    - Depending on its direct inputs
    - Can depend also on its internal states
  - Composition of modules in appropriate order

Composition: Example

- Overall behavior given by: $f_C(f_B(f_A))$

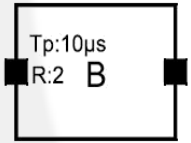


Function execution

- A given function is executed iff there are enough samples available at the input port
  - Fixed number of consumed and produced samples
- Every sample has a time stamp
  - Fixed interval called time step

Discrete Event Models module and port attributes

- Time step (module)
- Time step (port)
- Rate (port)
- Delay (port)
- Time offset (port)
  - Specialized port



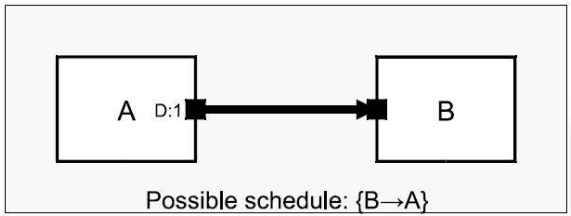
Scheduling

- Discrete Event Models Cluster: set of connected Discrete Event Models modules which belong the same static schedule
  - Parameters must be compatible
  - Defines a sequence in wich Discrete Event Models modules are executed

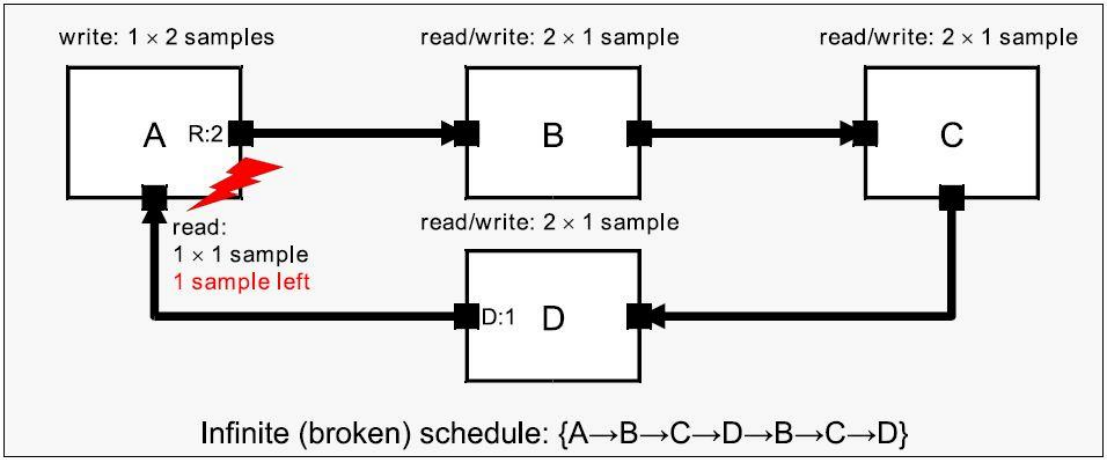
- Overcome SystemC event-based scheduler
  - Increase in efficiency
  - Interaction with pure SystemC through specific converter ports

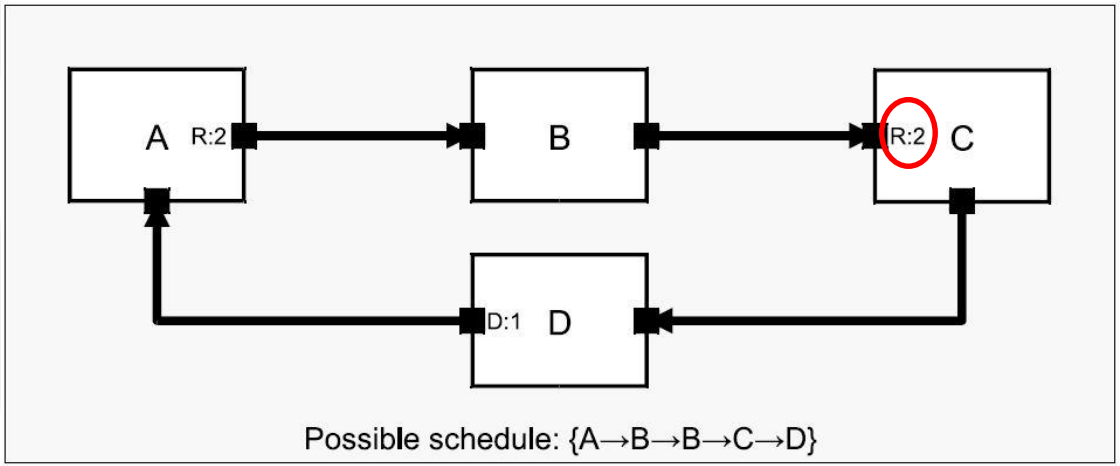
Scheduling feasibility

- Loops are sources for problems:
  - Every loop must present at least one delay port
  - No leftover samples allowed
- Delay ports can lead to inconsistency:
  - Initial value should be specified



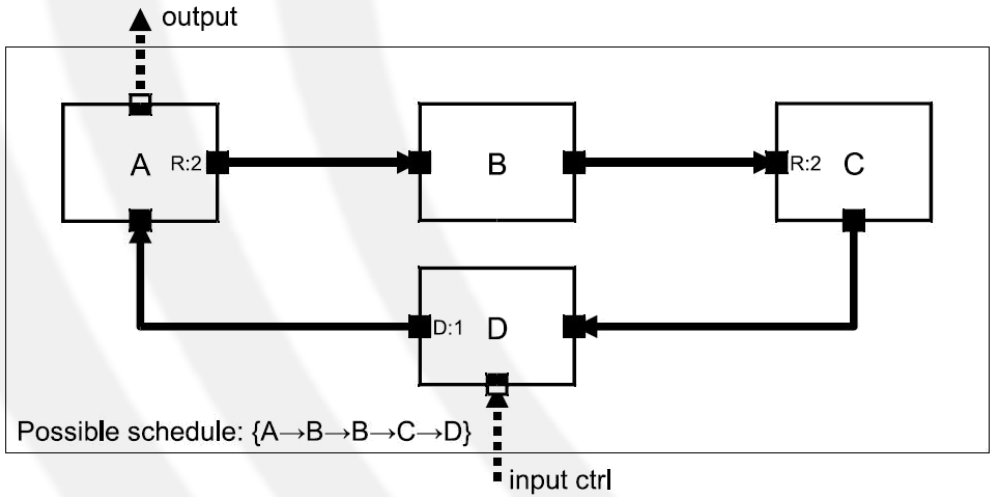
Scheduling loops





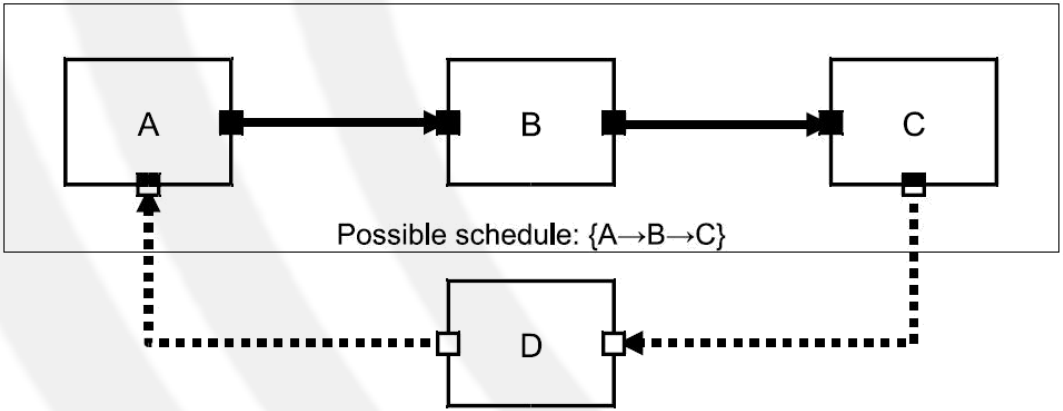
Connection to discrete-event models

- Converter input/output port are given
  - With a fixed time offset



Closed loop and discrete domain

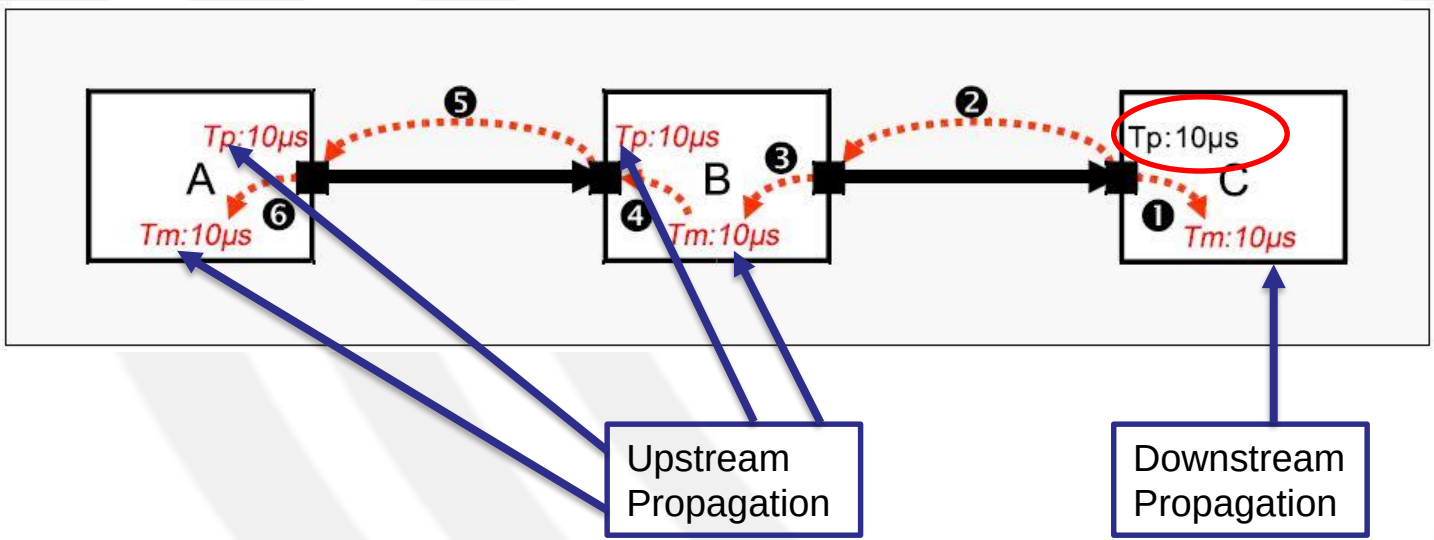
- Special converter ports introduce implicit delay
  - Data readed by A, reach C in the same Delta cycle
  - Data produced by C are read by A after 1 cycle



Time step assignment and propagation

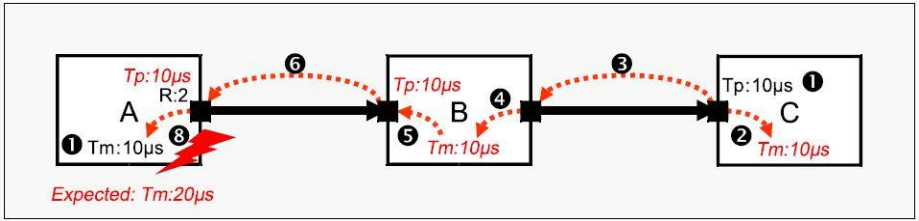
- Port rates and delays are very useful
  - Handle different frequency domains
  - Express models involving nested loops
- Compatibility mandatory for cluster consistency
  - Consistency is indipendent from sampling period
- Compatibility assure inference of the time step
  - Upstream and Downstream

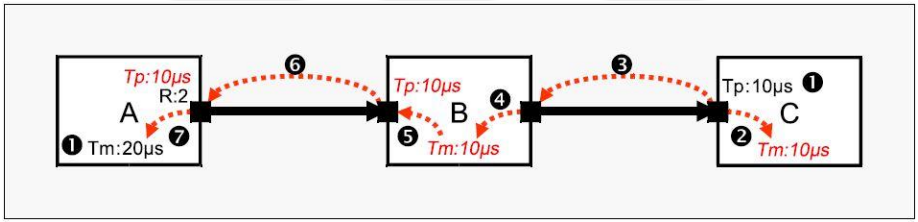
Time step assignment and propagation (Example)



Consistency

- $T_m = T_{p_{input}} * R_{input} = T_{p_{output}} * R_{output}$





Multiple clusters

- A Discrete Event Models model can present more clusters
  - Every cluster is scheduled independetly
  - Every cluster can be connected to the others

- Communication between Discrete Event Models cluster evolves accordingly to the SystemC discrete-event scheduler!

Language Constructs

- Discrete Event Models Modules
- Discrete Event Models Ports
- Discrete Event Models Signals

Discrete Event Models Modules

```
SCA_TDF_MODULE(my_tdf_module) ❶
{
  // port declarations
  sca_tdf::sca_in<double>  in; ❷
  sca_tdf::sca_out<double> out;

  SCA_CTOR(my_tdf_module) {} ❸

  void set_attributes() ❹
  {
    // module and port attributes
  }

  void initialize() ❺
  {
    // initial values of ports with a delay
  }

  void processing() ❻
  {
    // time-domain signal processing behavior or algorithm
  }

  void ac_processing() ❼
  {
    // small-signal frequency-domain behavior
  }
};
```

1. Primitive module declaration
   - Macro or extending sca_tdf::sca_module
2. Input and output ports declarations
3. Constructor
   - Macro or by creating a new class
     - Derived from sca_tdf::sca_module
     - Always with paramenter sc_core::sc_module_name
   - Only function that can be called by the user
4. Function to define module and ports attributes
5. Function to initialize data members
   - Representing module state and samples
6. Implementation of functionality
   - Processing function in time-domain
7. Implementation of functionality and noise
   - Small-frequency domain can be used to model noise

Discrete Event Models Modules: constraints on usage

- Hierarchy is not supported
  - Composition of Discrete Event Models modules is possible using regular SystemC classes
- Function to describe discrete-event behavior are not allowed
  - SystemC forbidden constructs:
    - SC_HAS_PROCESS, SC_METHOD, SC_THREAD
    - wait, next_trigger, sensitive
  - Incompatibility in execution semantics
- Local time of a Discrete Event Models module is calculate independently
  - get_time instead of sc_core::sc_time_stamp
- Specialized converter port to use SystemC signals

Discrete Event Models ports

- in and out port but no inout
  - Incopatible with the adopted model of computation
- «intra-cluster» ports:
  - sca_tdf::sca_in<T>
  - sca_tdf::sca_out<T>
- «extra-cluster» ports:
  - sca_tdf::sca_de::sca_in<T>
  - sca_tdf::sca_de::sca_out<T>

Discrete Event Models ports (cont.d)

- Getter and Setter methods are given for ports attributes
  - Available attributes: simestep, rate, delay, timeoffset
  - set_[attribute](value) and get_[attribute]() functions defined
  - Setters called by set_attribute(), Getters by initialize() and process()
- A port with non-zero delay must be always initialized
  - Member function initialize() of the module
- Multirate port permit to read/write on a specific sample
  - E.g., input_port.read(0) or output_port.write(val, 1)

Discrete Event Models Signals

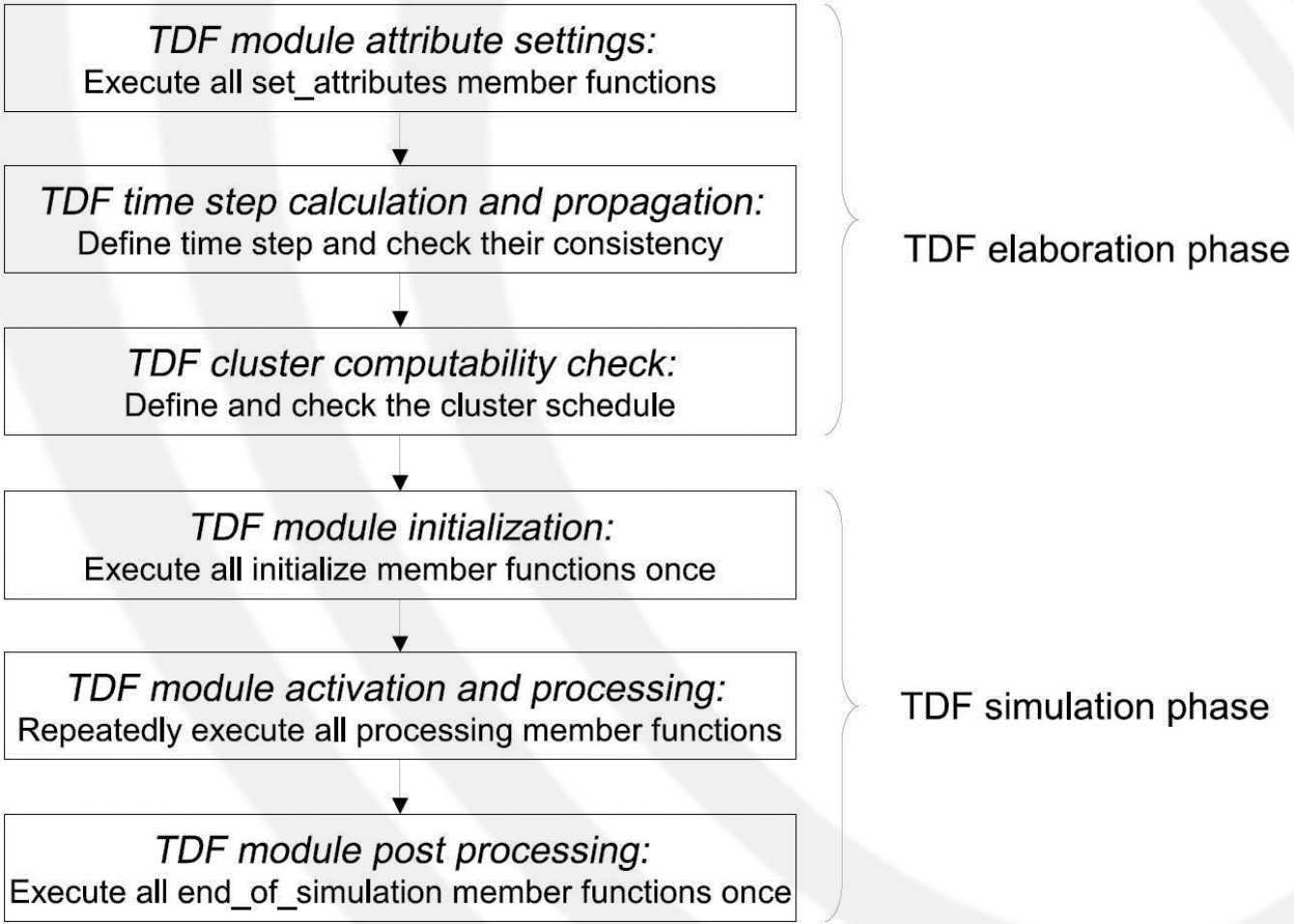
- Used to connect Discrete Event Models ports of different modules
  - They carry the sample, ports determine direction
  - sca_tdf::sca_signal<T>
- Little differences with SystemC signals
  - read() and write() functions are not provided
  - Naming for signal as in SystemC on costructor

Discrete Event Models and continuous-time modeling

- Discrete Event Models is based on production of (discrete) samples

- It is possible to embed linear dynamic equations
  - Linear transfer function (Laplace)
    - Numerator-denominator form (sca_tdf::sca_ltf_nd)
    - Zero-pole form (sca_tdf::sca_ltf_zp)
    - Coefficients as objects of sca_util::sca_vector
    - Complex domain through sca_util::sca_complex
  - State-space equations
    - Matrix based representation (sca_tdf::sca_ss)

Discrete Event Models execution semantics



Continuous-time non-conservative MoC

Linear Signal Flow

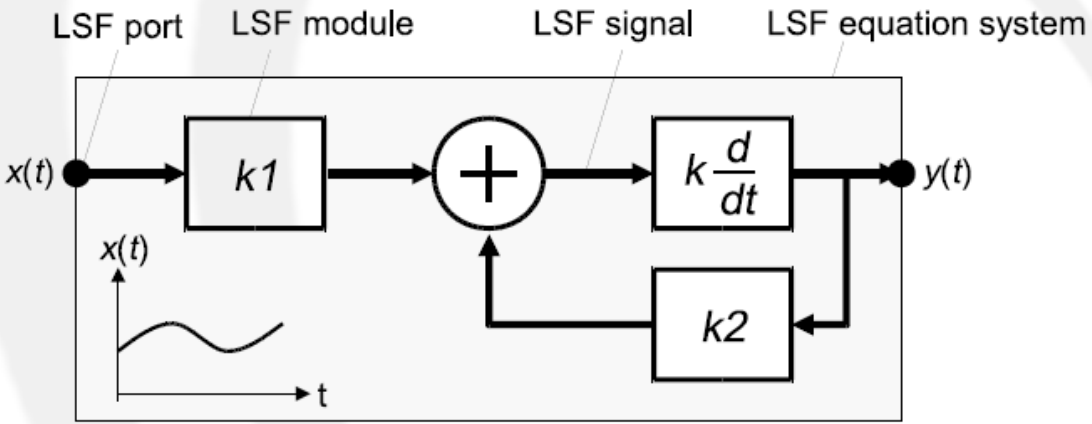
Fundamentals

- Behavior defined as relations between variables of a set of Linear DAE
  - Continuous-time modeling using directed real-value signals
    - Non conservative description
    - Only one real-valued is used to represent each signal
- Signal flow models represented by block diagram notation
  - Elementary parts represented by blocks
  - Signal used to interconnect these blocks
- An LSF model is composed by a set of connected LSF modules (LSF equation system or cluster)

Block diagrams

- Elementary parts or functions
  - LSF Blocks
    - Every block introduces a well-defined function
    - Define within a Standard API
    - Extension of API: “forbidden”!
- Interconnection between basic blocks
  - LSF Signals
    - Specialization of SystemC signals
    - Fixed (Real) data-type
- Interfaces for discrete models
  - Timed Data Flow
  - SystemC-RTL

Setup of LSF equations



- Equations system composed by the constructor
  - Compose mathematical equations
    - Blocks
    - Interconnections
  - Set to default values the non-specified parameters
- E.g., $y(t) = k_1 \frac{dx(t)}{dt} + k_2 \frac{dy(t)}{dt}$

LSF modules

LSF module name	Description
sca_lsf::sca_add	Weighted addition of two LSF signals.
sca_lsf::sca_sub	Weighted subtraction of two LSF signals.
sca_lsf::sca_gain	Multiplication of an LSF signal by a constant gain.
sca_lsf::sca_dot	Scaled first-order time derivative of an LSF signal.
sca_lsf::sca_integ	Scaled time-domain integration of an LSF signal.
sca_lsf::sca_delay	Scaled time-delayed version of an LSF signal.
sca_lsf::sca_source	LSF source.
sca_lsf::sca_ltf_nd	Scaled Laplace transfer function in the time-domain in the numerator-denominator form.
sca_lsf::sca_ltf_zp	Scaled Laplace transfer function in the time-domain in the zero-pole form.
sca_lsf::sca_ss	Single-input single-output state-space equation.
sca_lsf::sca_tdf::sca_gain, **sca_lsf::sca_tdf_gain**	Scaled multiplication of a TDF input signal with an LSF input signal.
sca_lsf::sca_tdf::sca_source, **sca_lsf::sca_tdf_source**	Scaled conversion of a TDF input signal to an LSF output signal.
sca_lsf::sca_tdf::sca_sink, **sca_lsf::sca_tdf_sink**	Scaled conversion from an LSF input signal to a TDF output signal.
sca_lsf::sca_tdf::sca_mux, **sca_lsf::sca_tdf_mux**	Selection of one of two LSF input signals by a TDF control signal (multiplexer).

LSF modules (Cont.d)

LSF module name	Description
sca_lsf::sca_tdf::sca_demux, **sca_lsf::sca_tdf_demux**	Routing of an LSF input signal to either one of two LSF output signals controlled by a TDF signal (demultiplexer).
sca_lsf::sca_de::sca_gain, **sca_lsf::sca_de_gain**	Scaled multiplication of a discrete-event input signal by an LSF input signal.
sca_lsf::sca_de::sca_source, **sca_lsf::sca_de_source**	Scaled conversion of a discrete-event input signal to an LSF output signal.
sca_lsf::sca_de::sca_sink, **sca_lsf::sca_de_sink**	Scaled conversion from an LSF input signal to a discrete-event output signal.
sca_lsf::sca_de::sca_mux, **sca_lsf::sca_de_mux**	Selection of one of two LSF input signals by a discrete-event control signal (multiplexer).
sca_lsf::sca_de::sca_demux, **sca_lsf::sca_de_demux**	Routing of an LSF input signal to either one of two LSF output signals controlled by a discrete-event signal (demultiplexer).

Time step assignement and propagation

- The time step
  - Simulation step for a LSF cluster
  - Disconnected by the numerical integration algorithm
    - Integration step can be smaller (never larger)
    - In PoC implementation (1.0): time step = integration step
- Can be explicitly assigned for every block or every cluster
  - At least one block: time step explicitly set
- Can be inferred by a propagation mechanism
  - Inferred by time step assignments within blocks
  - Connection of LSF and Discrete Event Models
    - Discrete Event Models Dynamic Time Step introduced in 2.0 standard
- Must preserves consistency

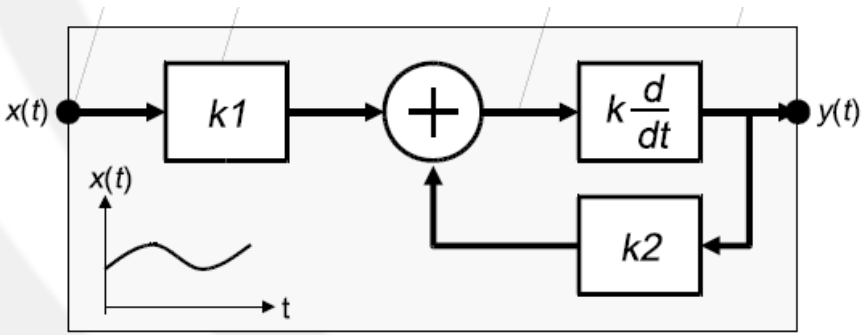
LSF Port and Signals

- Two classes of Ports
  - sca_lsf::sca_in
  - sca_lsf::sca_out
  - Fixed data-type (called *signal flow nature)*
    - Implicitly a double
    - Convertions are forbidden!
  - Special ports to connect the discrete world!

- Signals
  - Used to bind LSF ports and components
  - Determine the direction of the signals
  - Not a templatic class
    - They can't be customized!

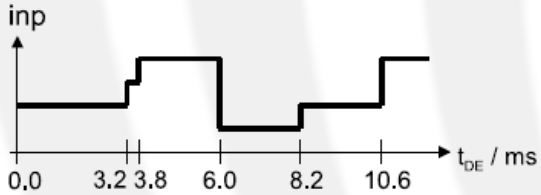
Modeling continuous-time behavior

```
SC_MODULE(my_structural_lsf_model)
{
  sca_lsf::sca_in  x; ❶
  sca_lsf::sca_out y;

  sca_lsf::sca_gain gain1, gain2; ❷
  sca_lsf::sca_dot dot1;
  sca_lsf::sca_add add1;

  my_structural_lsf_model( sc_core::sc_module_name, double k1, double k2 )
  : x("x"), y("y"), gain1("gain1", k1), gain2("gain2", k2), dot1("dot1"), add1("add1"), ❸
    sig1("sig1"), sig2("sig2"), sig3("sig3")
  {
    gain1.x(x); ❹
    gain1.y(sig1);
    gain1.set_timestep(1,sc_core::SC_MS); ❺

    add1.x1(sig1);
    add1.x2(sig3);
    add1.y(sig2);

    dot1.x(sig2);
    dot1.y(y);

    gain2.x(y);
    gain2.y(sig3);
  }

 private:
  sca_lsf::sca_signal sig1, sig2, sig3; ❻
};
```

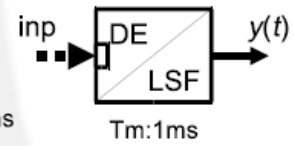


Interaction with discrete world

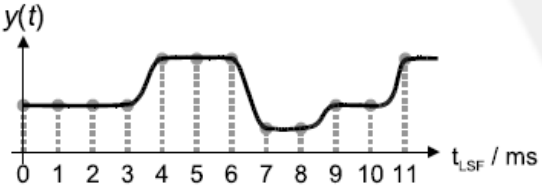
Discrete-event signal
Instance of class
sc_core::sc_signal<double>



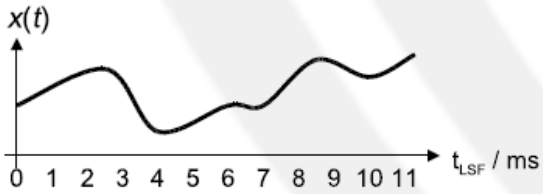
LSF converter module
Instance of class
sca_lsf::sca_de::sca_source



LSF signal
Instance of class
sca_lsf::sca_signal

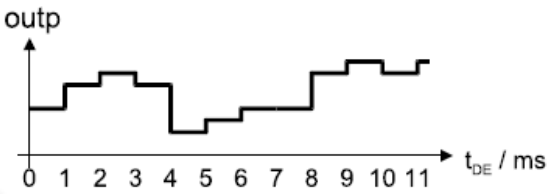


LSF signal
instance of class
sca_lsf::sca_signal



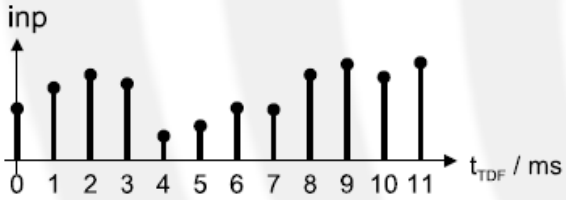
LSF converter module
instance of class
sca_lsf::sca_de::sca_sink

discrete-event signal
instance of class
sc_core::sc_signal<double> or
sc_core::sc_buffer<double>

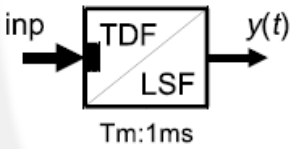


Interaction with Discrete Event Models

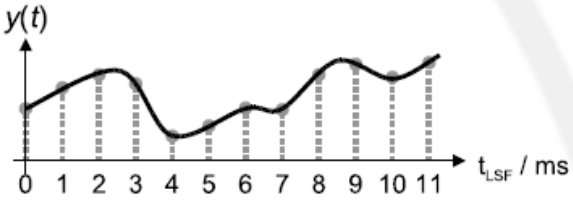
TDF signal
Instance of class
sca_tdf::sca_signal<double>



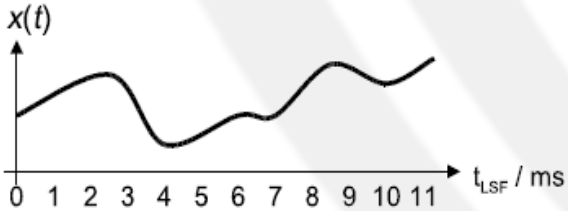
LSF converter module
Instance of class
sca_lsf::sca_tdf::sca_source



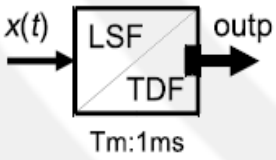
LSF signal
Instance of class
sca_lsf::sca_signal



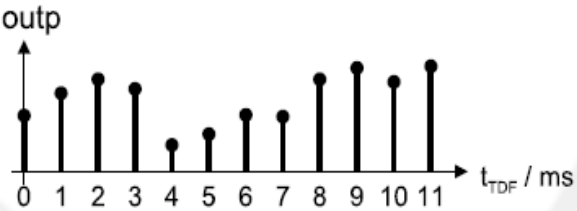
LSF signal
Instance of class
sca_lsf::sca_signal



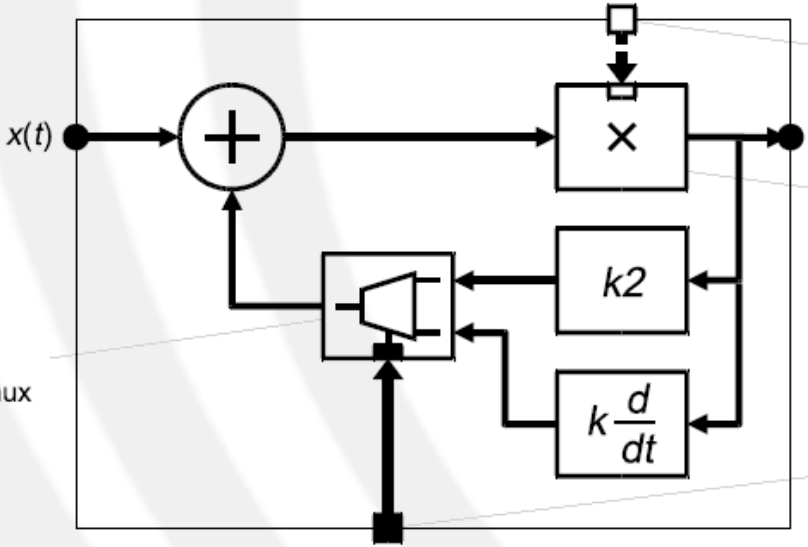
LSF converter module
Instance of class
sca_lsf::sca_tdf::sca_sink



TDF signal
Instance of class
sca_tdf::sca_signal<double>



Using discrete-events



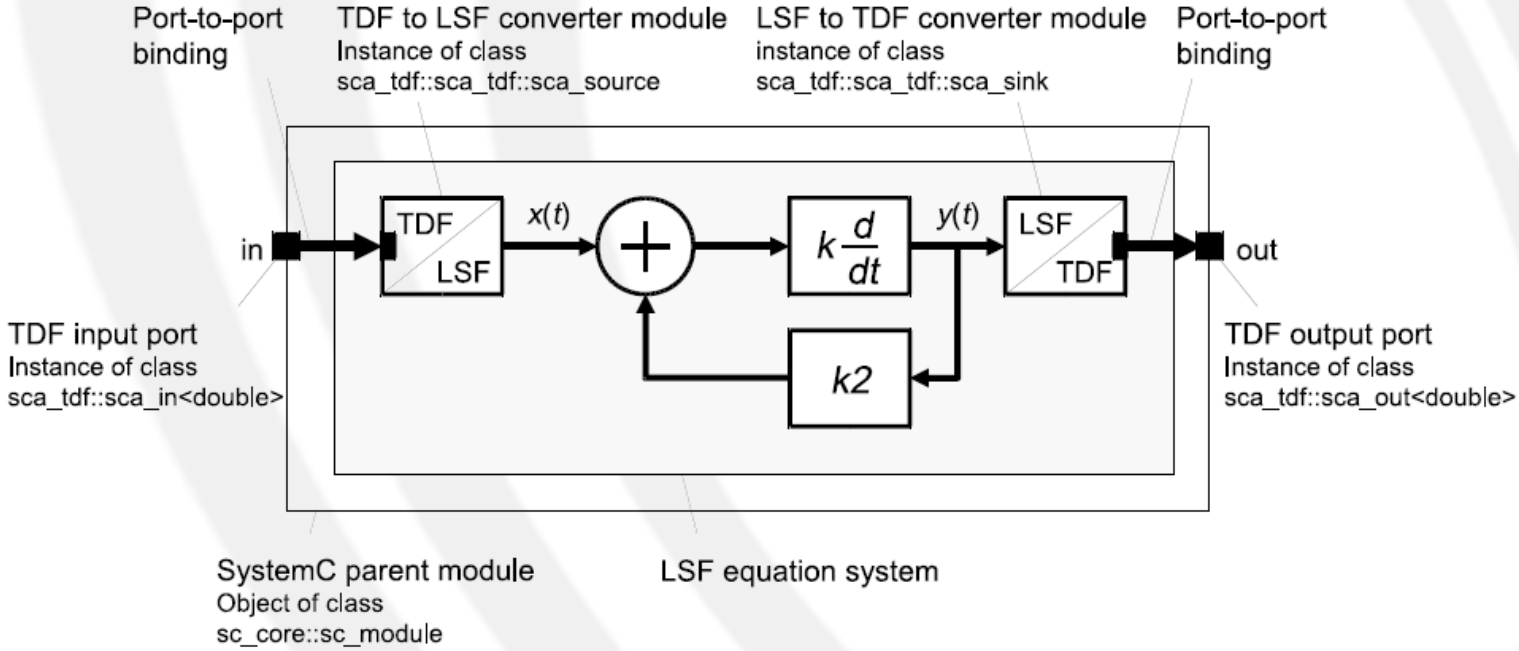
Discrete-event control input
Instance of class
sc_core::sc_in<bool>

LSF multiplication
Instance of class
sca_lsf::sca_de::sca_gain

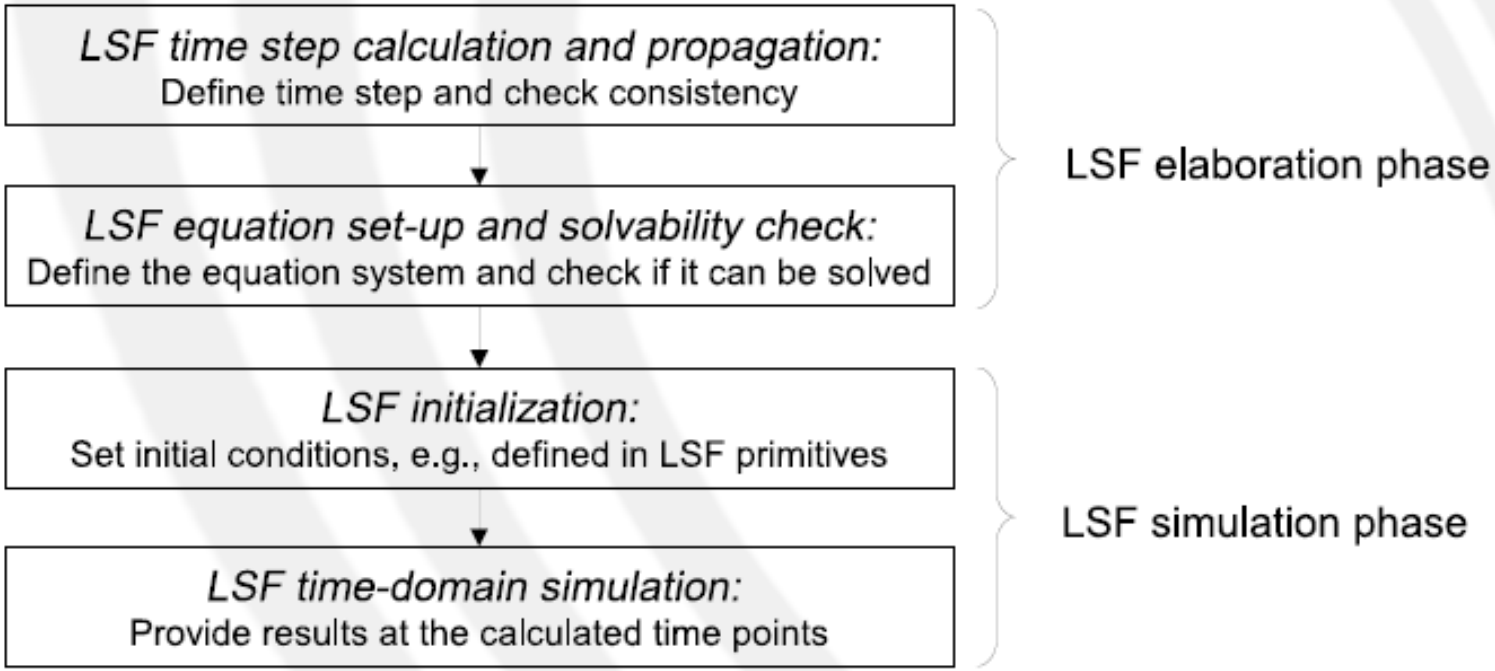
LSF multiplexer
Instance of class
sca_lsf::sca_tdf::sca_mux

TDF control input
Instance of class
sca_tdf::sca_in<bool>

LSF Model encapsulation



Execution Semantics



Continuous-time conservative MoC

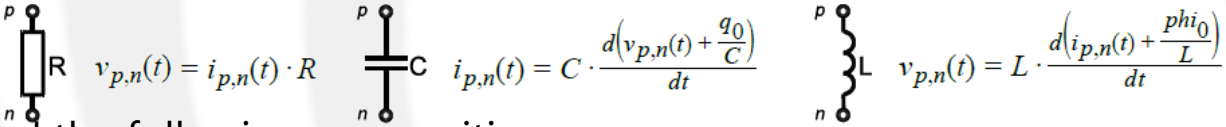
Electrical Linear Network

Fundamentals

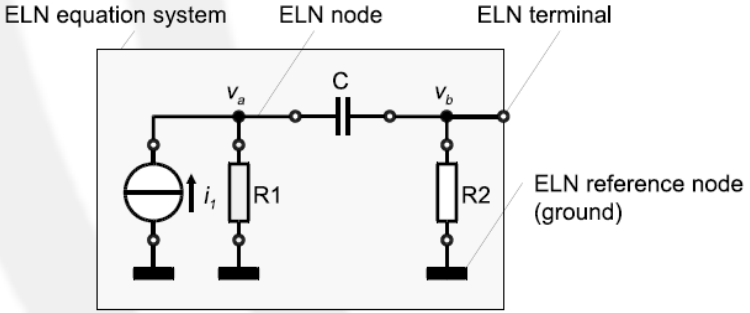
- Composition of electrical primitives and interconnections
  - Every primitive is conservative
  - Interconnections must preserve conservation of energy
- Resolution of systems of equations
  - Every primitive introduce a set of equation
  - "Cluster" of connected components: system of equations
    - Consistency checking at run-time
- Implicit introduction of Kirchhoff's Laws
  - Kirchhoff's Current Law: $\sum_\sigma i_k(t) = 0$ thus, $\sum I_{in} = \sum I_{out}$
  - Kirchhoff's Voltage Law: $\sum V_i(t) = 0$

Equation system (Example)

- Consider the basic components:



- … and the following composition:



- $$\begin{cases} -i_1 + \frac{v_a}{R1} + C * \frac{d\left(v_{a,b} + \frac{q_0}{C}\right)}{dt} = 0 \\ \frac{v_b}{R2} + C * \frac{d\left(v_{a,b} + \frac{q_0}{C}\right)}{dt} = 0 \\ \\ Kirchhoff\ Laws \end{cases}$$

ELN Basic blocks

- ELN Modules
  - Predefined electrical primitive
    - Used to build electrical network
  - Interface composed by *terminal*
  - Parameters for time-domain simulation
    - *time step, delays, offsets etc..*
      - Module specific: check the LRM!

- ELN Terminals: ***sca_eln::sca_terminal***
  - Basic component of ELN interfaces
    - Components are connected using terminals
    - Can be seen as ports
    - No direction!
  - "Bind" to nodes

- ELN Nodes: ***sca_eln::sca_node***
  - Primitive to connect modules
    - Binded to terminals, can be seen as signals
  - Special kind of node: Ground (***sca_eln::sca_node_ref***)
    - Implements the ground of a electrical circuit

ELN Basic Blocks (Cont.d)

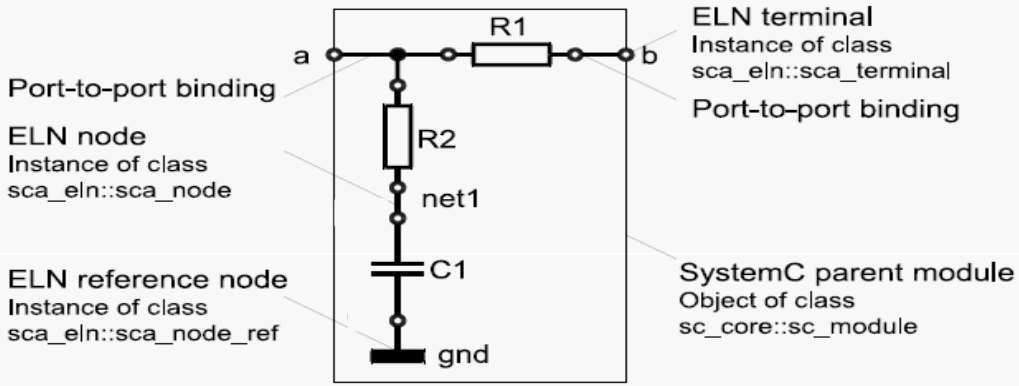
ELN module name	Description
sca_eln::sca_r	Resistor
sca_eln::sca_c	Capacitor
sca_eln::sca_l	Inductor
sca_eln::sca_vcvs	Voltage controlled voltage source
sca_eln::sca_vccs	Voltage controlled current source
sca_eln::sca_ccvs	Current controlled voltage source
>sca_eln::sca_cccs	Current controlled current source
sca_eln::sca_nullor	Nullor (nullator - norator pair), ideal op-amp
sca_eln::sca_gyrator	Gyrator
sca_eln::sca_ideal_transformer	Ideal transformer
sca_eln::sca_transmission_line	Transmission line
sca_eln::sca_vsource	Independent voltage source
sca_eln::sca_isource	Independent current source
sca_eln::sca_tdf::sca_r, **sca_eln::sca_tdf_r**	Variable resistor controlled by a TDF input signal
sca_eln::sca_tdf::sca_c, **sca_eln::sca_tdf_c**	Variable capacitor controlled by a TDF input signal
sca_eln::sca_tdf::sca_l, **sca_eln::sca_tdf_l**	Variable inductor controlled by a TDF input signal
sca_eln::sca_tdf::sca_rswitch, **sca_eln::sca_tdf_rswitch**	Switch controlled by a TDF input signal

ELN Basic Blocks (cont.d)

ELN module name	Description
sca_eln::sca_tdf::sca_vsource, sca_eln::sca_tdf_vsource	Voltage source driven by a TDF input signal
sca_eln::sca_tdf::sca_isource, sca_eln::sca_tdf_isource	Current source driven by a TDF input signal
sca_eln::sca_tdf::sca_vsink, sca_eln::sca_tdf_vsink	Converts voltage to a TDF output signal
sca_eln::sca_tdf::sca_isink, sca_eln::sca_tdf_isink	Converts current to a TDF output signal
sca_eln::sca_de::sca_r, sca_eln::sca_de_r	Variable resistor controlled by a discrete-event input signal
sca_eln::sca_de::sca_c, sca_eln::sca_de_c	Variable capacitor controlled by a discrete-event input signal
sca_eln::sca_de::sca_l, sca_eln::sca_de_l	Variable inductor controlled by a discrete-event input signal
sca_eln::sca_de::sca_rswitch, sca_eln::sca_de_rswitch	Switch controlled by a discrete-event input signal
sca_eln::sca_de::sca_vsource, sca_eln::sca_de_vsource	Voltage source driven by a discrete-event input signal
sca_eln::sca_de::sca_isource, sca_eln::sca_de_isource	Current source driven by a discrete-event input signal
sca_eln::sca_de::sca_vsink, sca_eln::sca_de_vsink	Converts voltage to a discrete-event output signal
sca_eln::sca_de::sca_isink, sca_eln::sca_de_isink	Converts current to a discrete-event output signal

Structural Composition

```
SC_MODULE(my_structural_eln_model)
{
  sca_eln::sca_terminal a; ❶
  sca_eln::sca_terminal b;

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

  SC_CTOR(my_structural_eln_model)
  : a("a"), b("b"), r1("r1", 10e3), r2("r2", 100.0), c1("c1", 100e-6), net1("net1"), gnd("gnd") ❸
  {
    r1.p(a); ❹
    r1.n(b);

    r2.p(a);
    r2.n(net1);

    c1.p(net1);
    c1.n(gnd);
  }

 private:
  sca_eln::sca_node net1; ❺
  sca_eln::sca_node_ref gnd;
};
```



Structural Composition (Cont.d)

1. The ELN terminals declared inside this module of class **sc_core::sc_module** become part of the structural composition.

2. The ELN primitive modules are declared within the parent module as child modules.

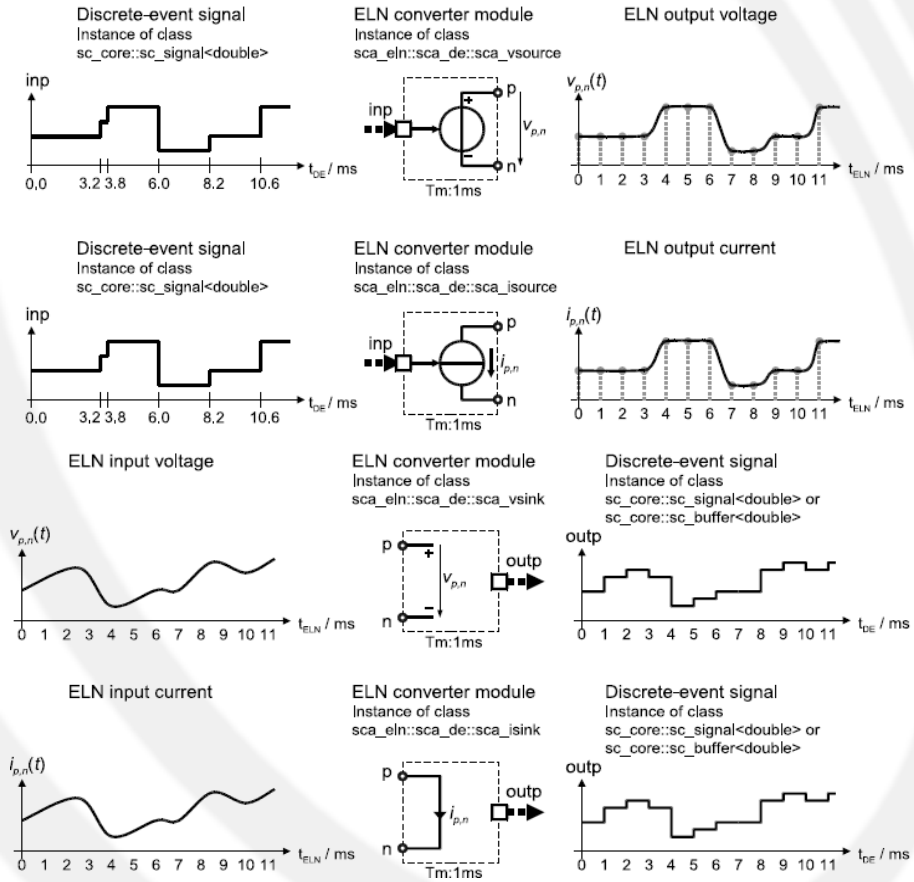
3. The initialization-list in the parent module's constructor propagates the necessary configuration parameters to the ELN terminals, ELN nodes, and child modules.

4. Port (terminal) binding is done inside the constructor of the parent module.

5. Internal ELN nodes are used to connect the ELN terminals and child modules. These nodes are declared in the private space, as they should not be accessible from outside the module.

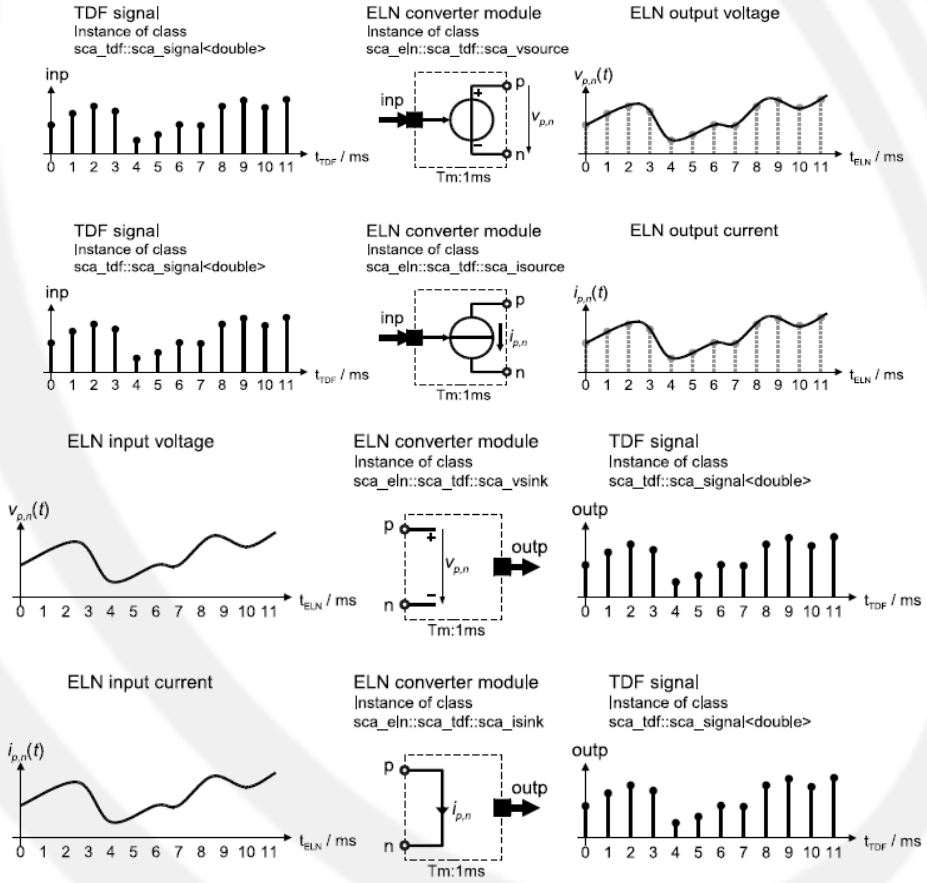
Interaction with Discrete Event Models

- Reading from Discrete Event Models
  - Voltage and Current sources controlled by a discrete signal
  - sca_eln::sca_tdf::sca_vsource
    - sca_eln::sca_tdf_sca_vsource
  - sca_eln::sca_tdf::sca_isource
    - sca_eln::sca_tdf_sca_isource
- Writing to Discrete Event Models
  - Voltage and Current sinks writing into a discrete signal
  - sca_eln::sca_tdf::sca_vsink
    - sca_eln::sca_tdf_sca_vsink
  - sca_eln::sca_tdf::sca_isink
    - sca_eln::sca_tdf_sca_isink

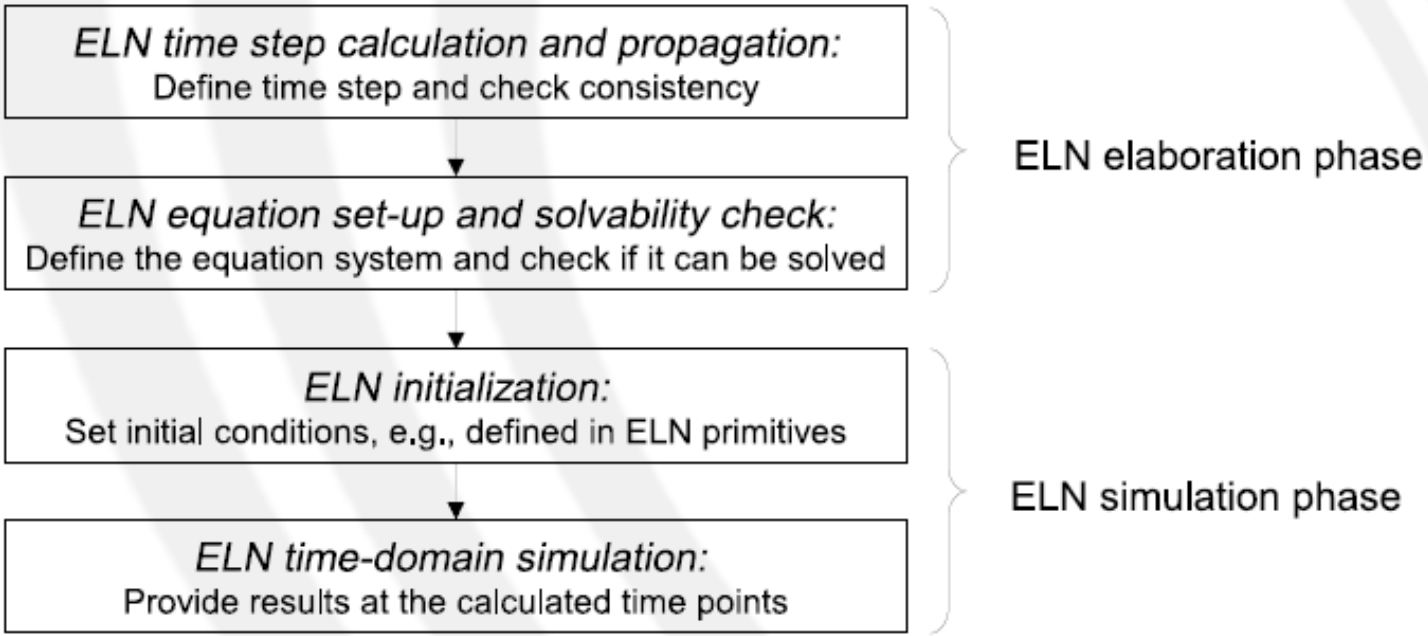


Interaction with Discrete Event Models

- Reading from Discrete Event Models
  - Voltage and Current sources controlled by a discrete signal
  - sca_eln::sca_tdf::sca_vsource
    - sca_eln::sca_tdf_sca_vsource
  - sca_eln::sca_tdf::sca_isource
    - sca_eln::sca_tdf_sca_isource
- Writing to Discrete Event Models
  - Voltage and Current sinks writing into a discrete signal
  - sca_eln::sca_tdf::sca_vsink
    - sca_eln::sca_tdf_sca_vsink
  - sca_eln::sca_tdf::sca_isink
    - sca_eln::sca_tdf_sca_isink



Execution Semantics



Analyzing the simulation

Tracing

Simulation control

- Time-domain simulation
  - Classic discrete-event SystemC simulation with AMS modules
    - *sc_start* function standard
  - Time no longer dependent on events: **natural concept of time**
    - Implicit events: sampling of continuous values
    - Resolution has to be set:
      sc_core::sc_set_time_resolution(1.0, sc_core::SC_FS)
      - Femtosecond: extremely high resolution, good for Analog descriptions
- Small-signal frequency-domain simulation
  - Small-signal (AC) simulation
    - *sc_ac_analysis::sca_ac_start* to start the simulation
  - Frequency-domain noise simulation
    - *sc_ac_analysis::sca_ac_noise_start* to start the simulation
  - Requires at least one time-domain simulation first
    - If not explicitly called, it is called by the kernel (transparent to the user)

Tracing alternatives

- Tracing to a VCD file
  - Special case of VCD tracing functions
    - *sca_util::sca_create_vcd_file("filename.vcd");*
  - Standard VCD tracing can be used to trace AMS signals
    - Attention: AMS signals translated in DE signals using TDF output ports
    - TDF/DE rules for synchronization are implicitly applied

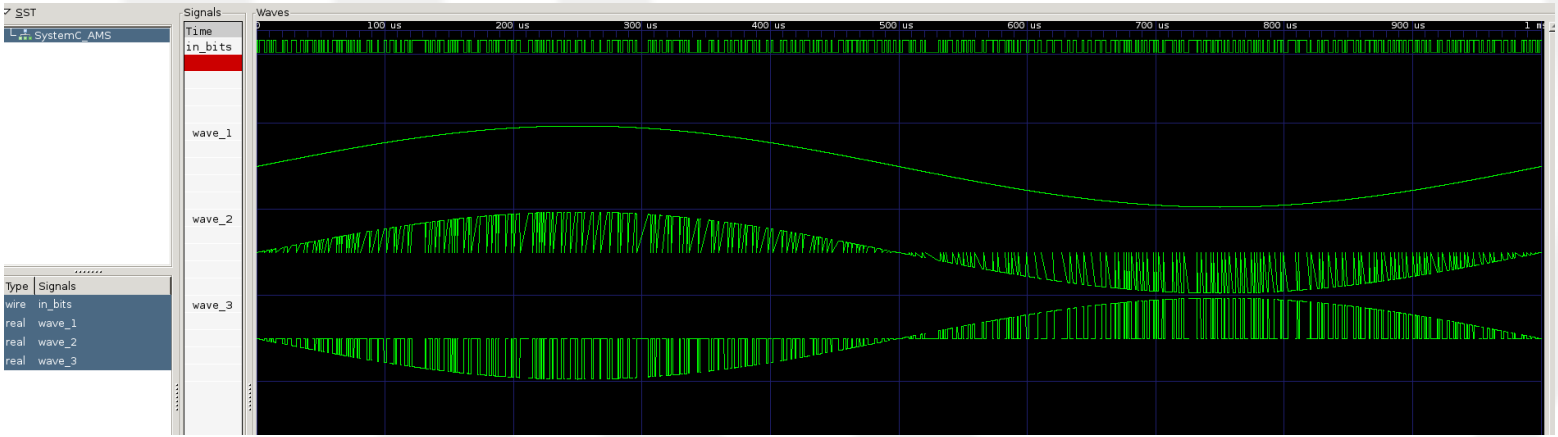
- Tracing to a tabular file
  - Data represented as a table
    - Every line corresponds to an instant of the simulation
    - Every column to a traced signal
      - Time stamp in the first column
  - Adhoc functions
    - *sca_util::sca_create_tabular_trace_file(filename.dat);*
      - Possibility to specify std::cout as file -> output to the out stream to avoid huge files

Trace file control

- Methods in the *sca_util::sca_trace_file* class to control the tracing
  - *enable:* start tracing when it is called (after the start)
  - *disable*: stop the tracing when it is called (before the end of simulation)
  - *reopen:* continue tracing in a new trace file
  - *set_mode*: change the mode of the trace when it is called
    - *sca_util::sca_sampling*: set the sampling time
    - *sca_util::sca_multirate*: defines which signals value should be written to the trace if no actual value is available.
      - *sca_util::SCA_INTERPOLATE*
      - *sca_util::SCA_HOLD_SAMPLE*
      - *sca_util::SCA_DONT_INTERPOLATE* to not write the value!
    - *sca_util::sca_ac_format:* format of the signals to write:
      - *sca_util::SCA_AC_REAL_IMAG:* amplitude and phase
      - *sca_util::SCA_MAG_RAD:* magnitude and radiant
      - *sca_util::SCA_AC_DB_DEG:* dB/degrees
    - *sca_noise_format:* how to write the noise contribution
      - *sca_util::SCA_NOISE_ALL:* noise divided in contributions
      - *sca_util::SCA_NOISE_SUM:* noise expressed as sum of all the contributions

Using GTKWave for Analog signals

- Open the vcd file and load the analog waves
- Right click on the wave
  - Data format
  - Analog -> Interpolated
  - Insert analog height extension



Suggested Readings

- Alain Vachoux, Christoph Grimm, and Karsten Einwich, "*SystemC-AMS requirements, design objectives and rationale*" *Proceedings of IEEE/ACM DATE* 2003.

- Alain Vachoux, Christoph Grimm, and Karsten Einwich, "*Analog and mixed signal modelling with SystemC-AMS*" Proceedings of IEEE ISCAS 2003

- Grimm, C., Barnasconi, M., Vachoux, A., & Einwich, K., "*An introduction to modeling embedded analog/mixed-signal systems using SystemC AMS extensions*" Proceedings of IEEE/ACM DAC 2008

- Franco Fummi, Michele Lora, Francesco Stefanni, Sara Vinco, "*Code generation alternatives to reduce heterogeneous embedded systems to homogeneity*", Proceedings of IEEE/ECSI FDL 2013

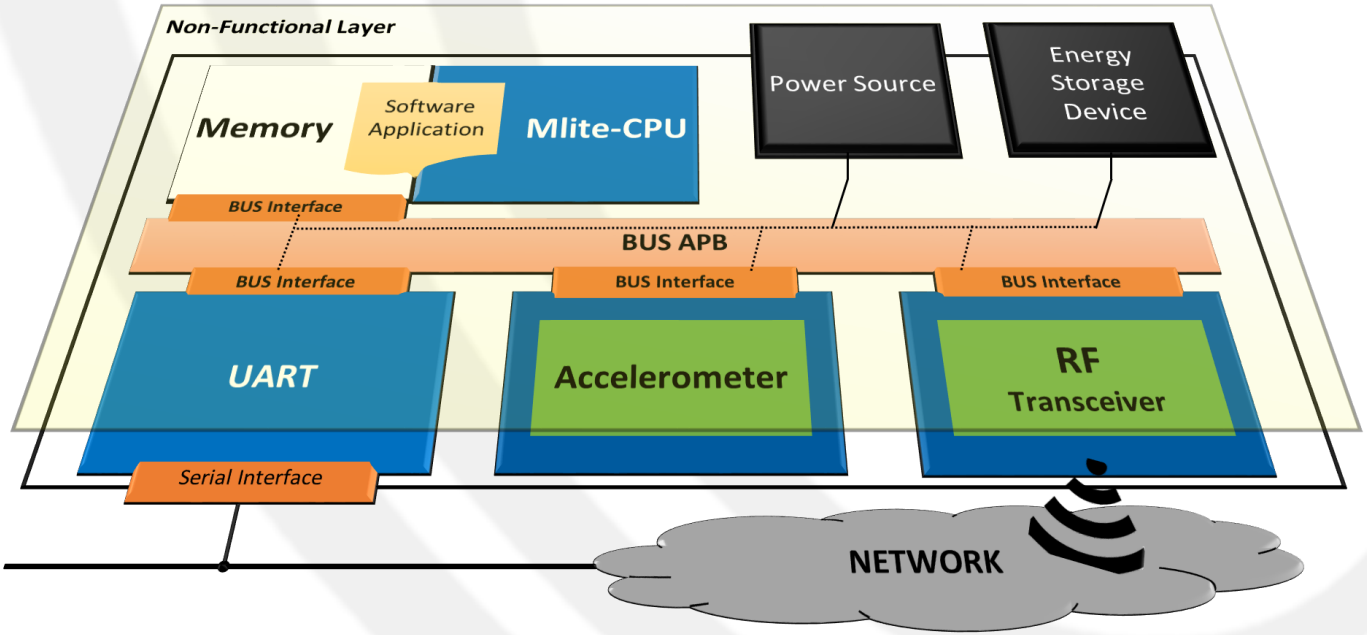
- François Pêcheux, Christophe Lallement, and Alain Vachoux. "*VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems.*" IEEE transactions on Computer-Aided design of integrated Circuits and Systems 24.2 (2005): 204-225.

Case studies

Some new examples

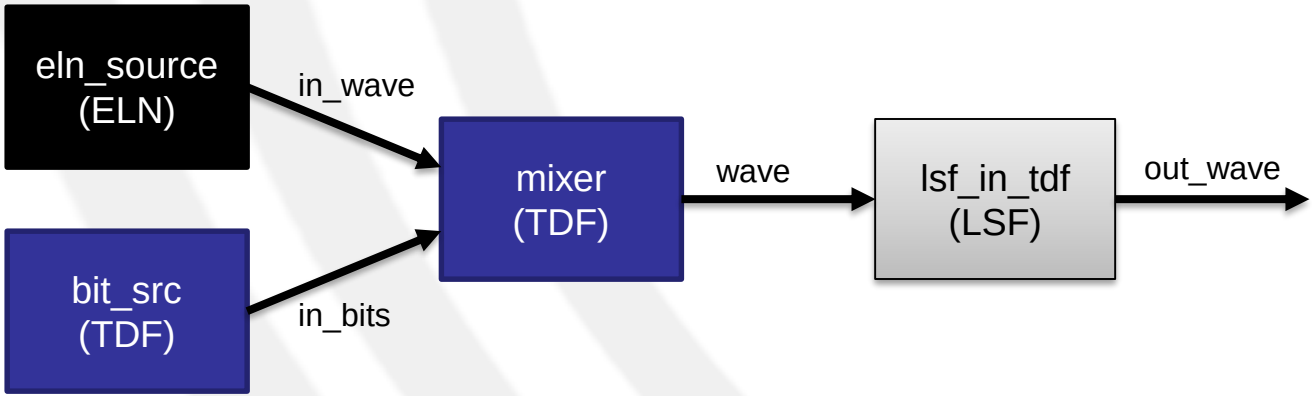
Heterogeneous Systems

- The Open-Source Test Case
  - Industrial-level Smart-System
  - Multi-domain
  - Developed within the SMAC FP7 European Project
    - SMArt systems Codesign



User Guide examples

- Example made assembling components described in the user guide
  - Using all the computational models
    - ELN, LSF and TDF
  - Purely didactic example



Thanks to Enrico Fraccaroli for providing this example.

Language Resources

- Accellera website
  - Very reactive support forum
  - Standard Definition
  - Official **User Guide**
    - Most of the material of this lesson is taken from there!
- The fundamental material is reported in the e-learning site
- Implementation:
  - http://www.coseda-tech.com/systemc-ams-proof-of-concept COSEDA technologies website:
    - 2.1-beta-PoC is the suggested implementation to use

SystemC-AMS Installation (tested only on Linux)

- Download the library from the e-learning, decompress it and setup the environment
  - `$> mkdir pse_libraries/systemc-ams`
  - `$> tar xzf systemc-ams-2.1.tar.gz`
  - `$> cd systemc-ams-2.1`
  - `$> mkdir objdir && cd objdir`
  - `$> ../configure --prefix=/home/user/pse_libraries/systemc-ams --with-systemc=/home/user/pse_libraries/systemc`
    - User is **YOUR USERNAME!**
    - **The last path is YOUR SYSTEMC INSTALLATION DIRECTORY!**
      - Look back at **Lesson 1!**
  - `$> make && make install`
    - Make provide the parameter –jN, where N is the number of parallel threads you want to use

- The procedure **should** be similar also for Mac OSX and Bash on Windows
  - Known issues with Cygwin

- **Export a environment variable SCAMS_HOME**
  - `$> export SCAMS_HOME=/home/user/pse_libraries/systemc-ams`
  - Add the command to the .bashrc file (as in Lesson 1)

SystemC-AMS into action

- Download the source code from the e-learning
  - $> tar xzf 05_systemc_ams.tar.gz
  - $> cd 05_systemc_ams
- You will find two examples:
  - The BASK model (TDF)
  - The Designer Guide example (Mixed)
- Compile and execute them
  - The directory structure of the Designer Guide will be slightly different
  - The BASK directory structure will be as in the previous lectures
- Analyze the traces using GTKWave