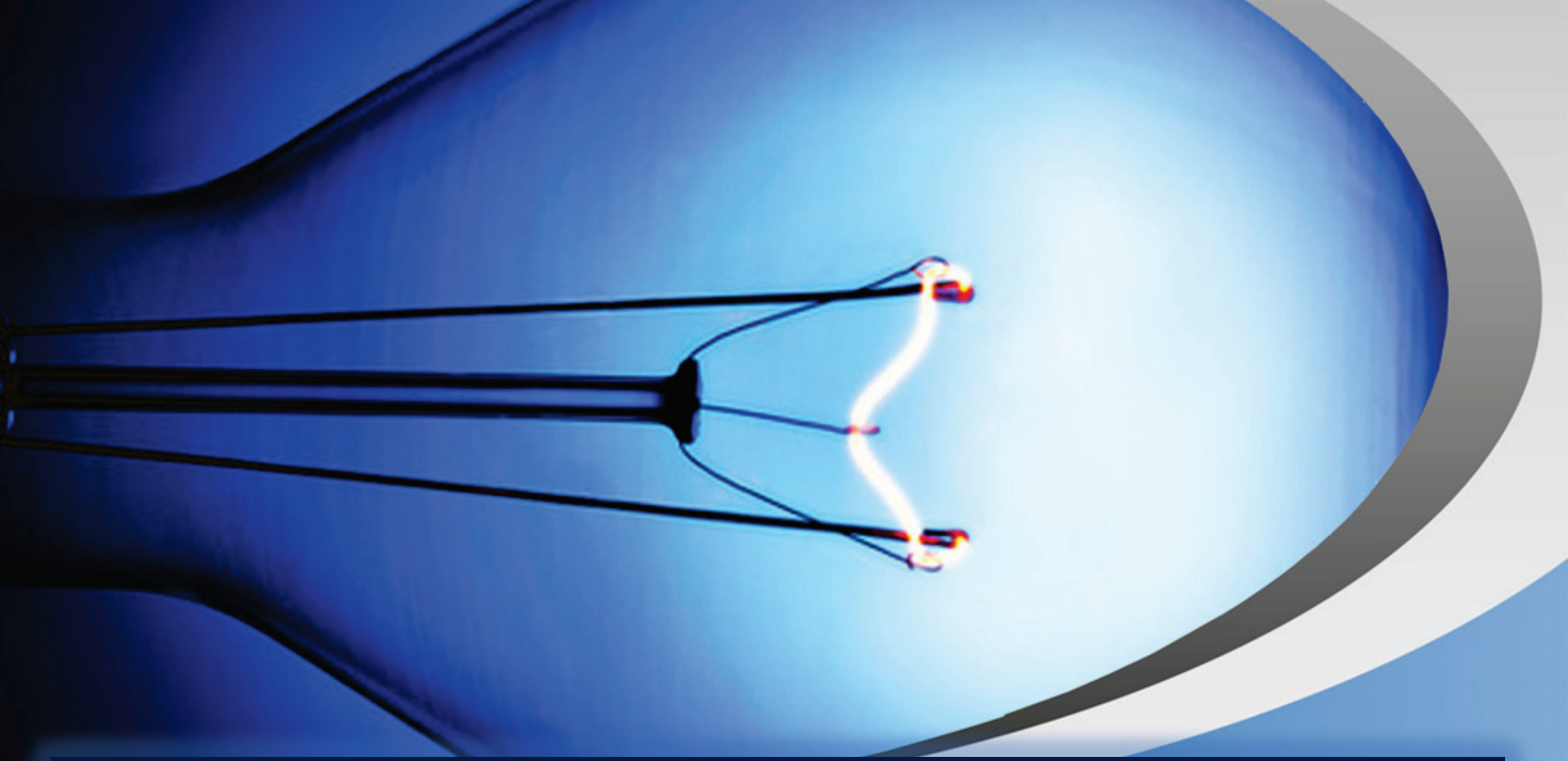


Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Tutorial Introduction

Martin Barnasconi, NXP Semiconductors

accellera SYSTEMS INITIATIVE

Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Martin Barnasconi – NXP Semiconductors
Karsten Einwich – Fraunhofer IIS/EAS
François Pêcheux – Université Pierre et Marie Curie
Torsten Mähne – Université Pierre et Marie Curie

Introduction

- Welcome!
- History
- SystemC AMS objectives
- Positioning
- Summary
- Tutorial overview
- Labs

SystemC AMS – History

- **1999:** Open SystemC Initiative (OSCI) announced
- **2000:** SystemC 1.0 released (sourceforge.net)
- **2002:** OSCI SystemC 1.0.2
- **2005:** IEEE Std 1666-2005 LRM
- **2005:** SystemC Transaction level modeling (TLM) 1.0 released
- **2007:** SystemC 2.2 released
- **2009:** SystemC TLM 2.0 standard
- **2009:** SystemC Synthesizable Subset Draft 1.3
- **2011:** IEEE Std 1666-2011 LRM
- **2012:** SystemC 2.3 PoC released by Accellera Systems Initiative

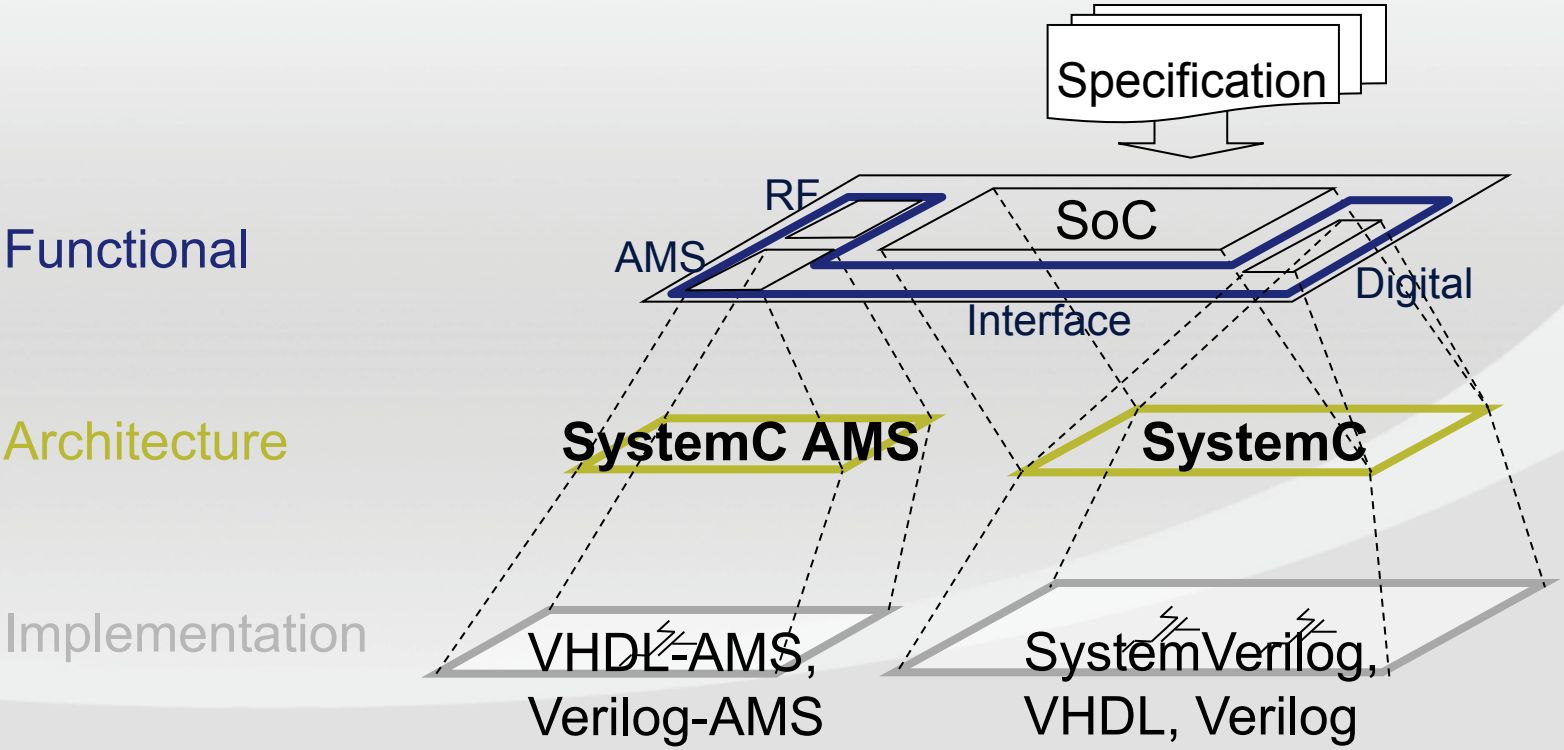
- **~2000:** First C-based AMS initiatives (AVSL, MixSigC)
- **2002:** SystemC-AMS study group started
- **2005:** First SystemC-AMS PoC released by Fraunhofer
- **2006:** OSCI AMSWG installed
- **2008:** SystemC AMS Draft 1 LRM
- **2010:** SystemC AMS 1.0 LRM standard
- **2010:** SystemC AMS 1.0 PoC released released by Fraunhofer IIS/EAS
- **2012:** SystemC AMS 2.0 draft standard
- **2013:** SystemC AMS 2.0 LRM standard
- **2013:** SystemC AMS 2.0 PoC test version
- **2014:** IEEE 1666.1 (SystemC AMS) started

accellera SYSTEMS INITIATIVE

SystemC AMS objectives

- System-level modeling standard and methodology for analog/mixed-signal systems
- An architecture design language for AMS system-level design and verification
- A platform that facilitates AMS model exchange and IP reuse
- SystemC-centric methodology to integrate (abstract) AMS and digital HW/SW descriptions
- Efficient abstraction concepts and modeling formalisms using dedicated models of computation (MoC)

Positioning SystemC AMS



accellera SYSTEMS INITIATIVE

Tutorial overview

- **Tutorial introduction**
- **Session 1:**
  - SystemC AMS introduction
  - Lab 1: Sine source connected to a sink
- **Session 2:**
  - Models of computation
  - Lab 2: Filtering and A/D conversion
- **Session 3:**
  - SystemC AMS 2.0 and applications
  - System example lab: Vibration sensor

Material for the “Labs”

- **Examples**

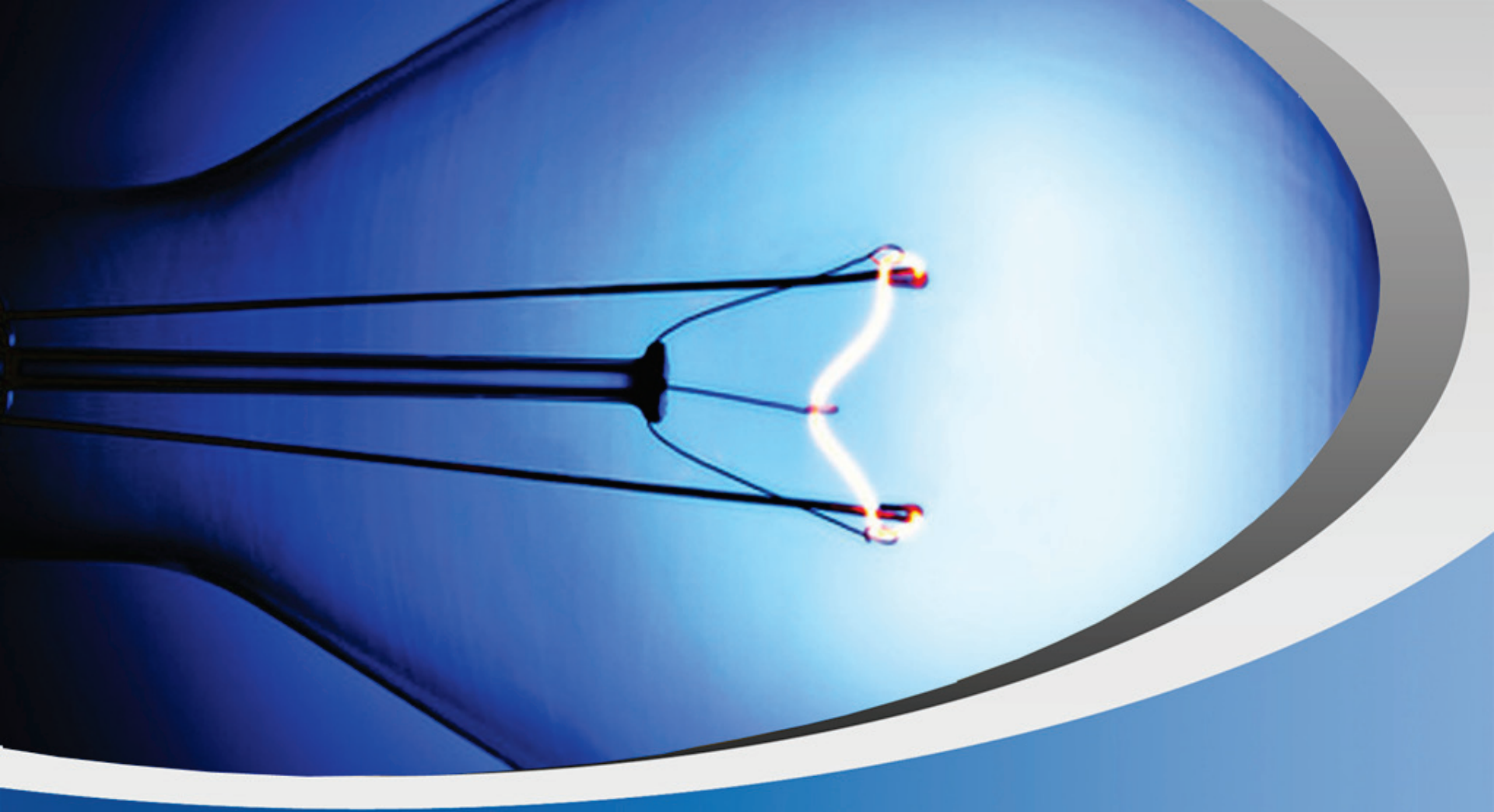
  http://videos.accellera.org/amsnextwave/labs/dvcon_labs_ws.zip

- **Solutions**

  http://videos.accellera.org/amsnextwave/labs/dvcon_labs_solutions.zip

- **SystemC AMS 2.0 Standard (LRM, User’s Guide)**

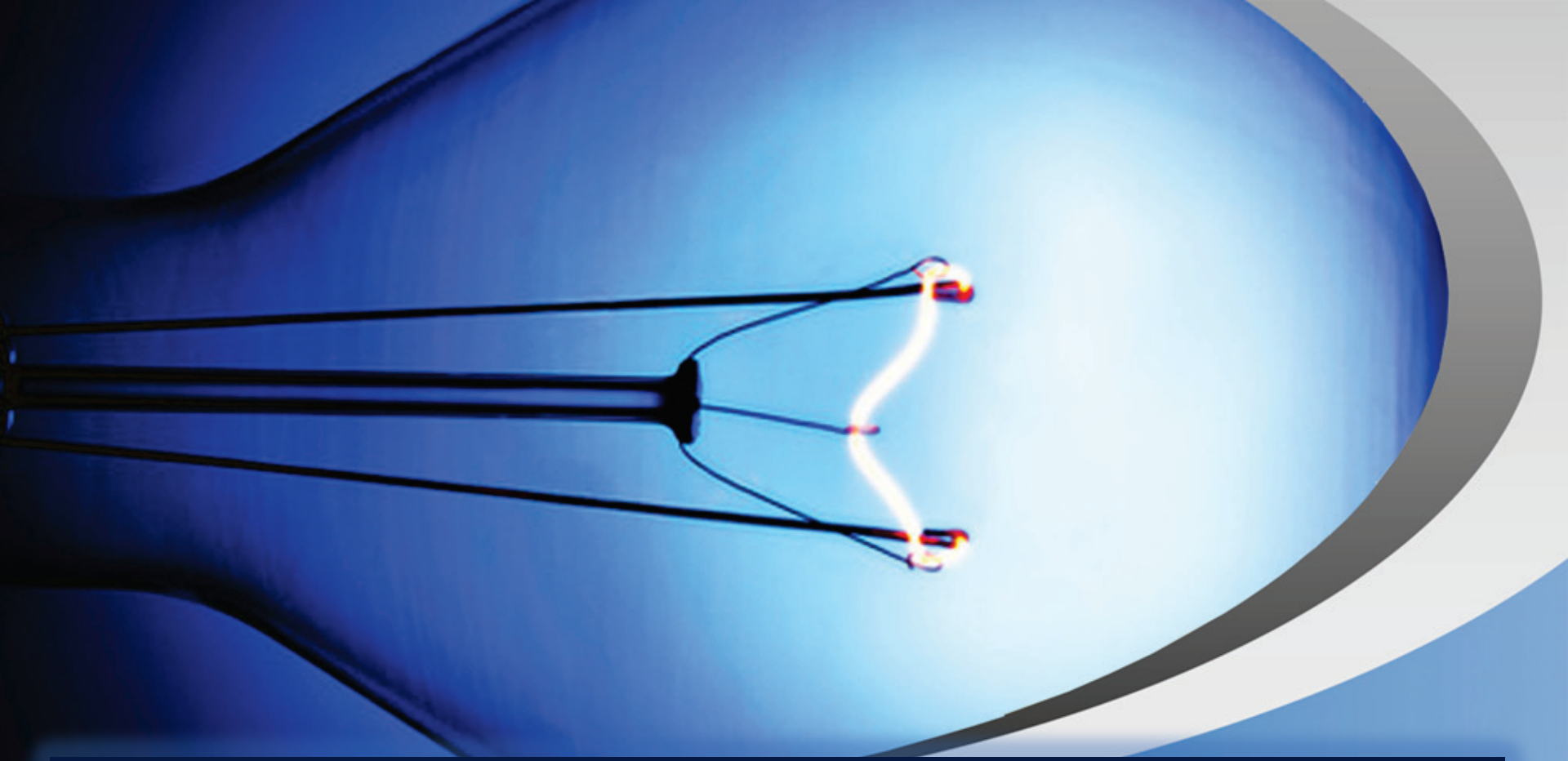
  http://www.accellera.org/downloads/standards/systemc



Thank you

Continue with Session 1: SystemC AMS Introduction

accellera
SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Session 1: SystemC AMS Introduction

Karsten Einwich, Fraunhofer IIS/EAS

accellera SYSTEMS INITIATIVE

Session 1 - Outline

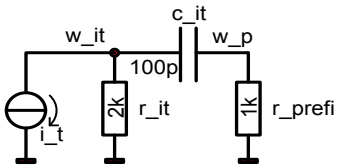
- **Session 1: SystemC AMS Introduction**
  - SystemC AMS language composition
  - Timed Data Flow (TDF) Model of Computation
  - Simulation control and debugging
- **Lab 1:**
  - Sine source connected to a sink
  - Q&A

SystemC AMS Language Composition

What's different between analog and digital?

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

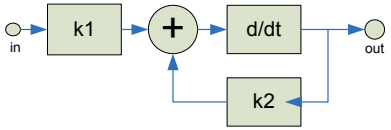
conservative (Kirchhoff)



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

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

non-conservative



$$out = \frac{d}{dt}(k1 \bullet in + k2 \bullet out)$$

➔ 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)

SystemC AMS language basics

- **A primitive module represents a contribution of equations to a Model of Computation (MoC)**
  - Primitives of each MoC must be derived from a specific base class
- **A channel represents in general an edge or variable of the equation system**
  - thus not necessarily a communication channel
- **SystemC AMS modules/channels are derived from the SystemC base classes**
  - sc_core::sc_module, sc_core::sc_prim_channel/sc_core::sc_interface
- **There is no difference compared to SystemC for hierarchical descriptions**
  - they are using SC_MODULE / SC_CTOR

SystemC AMS models of computation

- **Timed Data Flow (TDF)**
  - Data flow semantics annotated with time
  - User-defined primitives to encapsulate any signal processing functionality (algorithm)
- **Linear Signal Flow (LSF)**
  - Modeling of continuous-time non-conservative behavior
  - Pre-defined LSF primitive modules for adders, integrators, differentiators, transfer functions, etc.
- **Electrical Linear Networks (ELN)**
  - Modeling of continuous-time conservative behavior
  - Pre-defined ELN primitive modules for linear components (e.g., resistors, capacitors) and switches

Symbol names and namespaces

- All SystemC AMS symbols have the prefix **sca_** and macros the prefix **SCA_**
- All SystemC AMS symbols are embedded in a namespace – the concept permits extensibility
- Symbols assigned to a certain MoC are in the corresponding namespace (**sca_tdf**, **sca_lsf**, **sca_eln**)
- Symbols relating to core functionality or general base classes embedded in the namespace **sca_core**
- Symbols of utilities like tracing and data types are in the namespace **sca_util**
- Symbols related to small-signal frequency-domain analysis are in the namespace **sca_ac_analysis**

SystemC AMS language composition - Summary

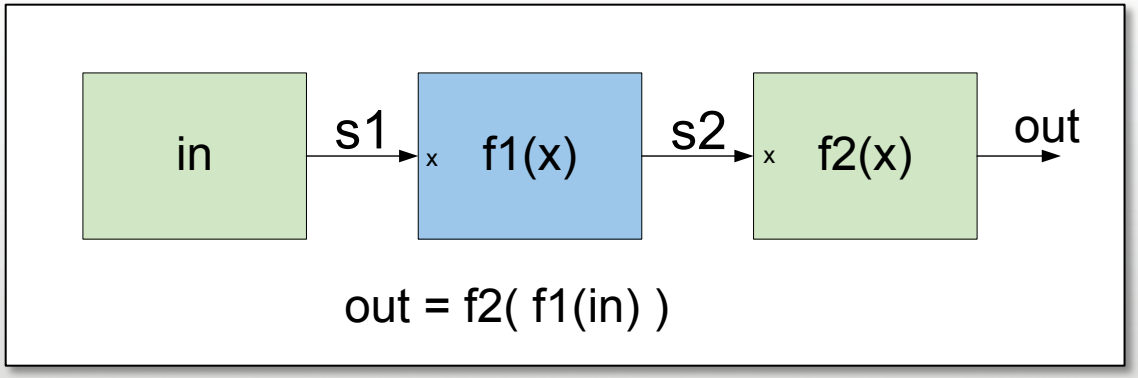
- **Basic keywords:**
  - **sca_module** – base class for SystemC AMS primitive
  - **sca_in / sca_out** – non-conservative (directed in / out port)
  - **sca_terminal** – conservative terminal
  - **sca_signal** – non-conservative (directed) signal
  - **sca_node / sca_node_ref** – conservative node / ground reference
- **The model of computation is assigned by the namespace e.g.:**
  - **sca_tdf::sca_module** – base class for timed data flow modules
  - **sca_lsf::sca_in** – a linear signal flow input port
  - **sca_tdf:sca_in<T>** – a TDF input port of type **T**
  - **sca_eln::sca_terminal** – an electrical linear network terminal
  - **sca_eln::sca_node** – an electrical linear network node

SystemC AMS language element composition – converter elements

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

SystemC AMS
Timed Data Flow (TDF)

Dataflow fundamentals

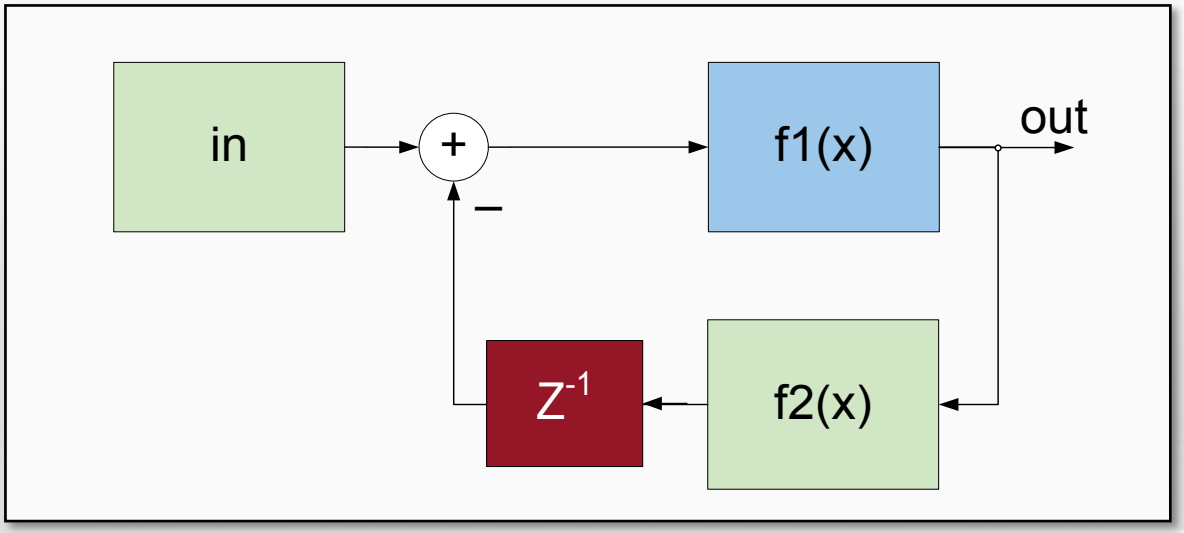


equation system:

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

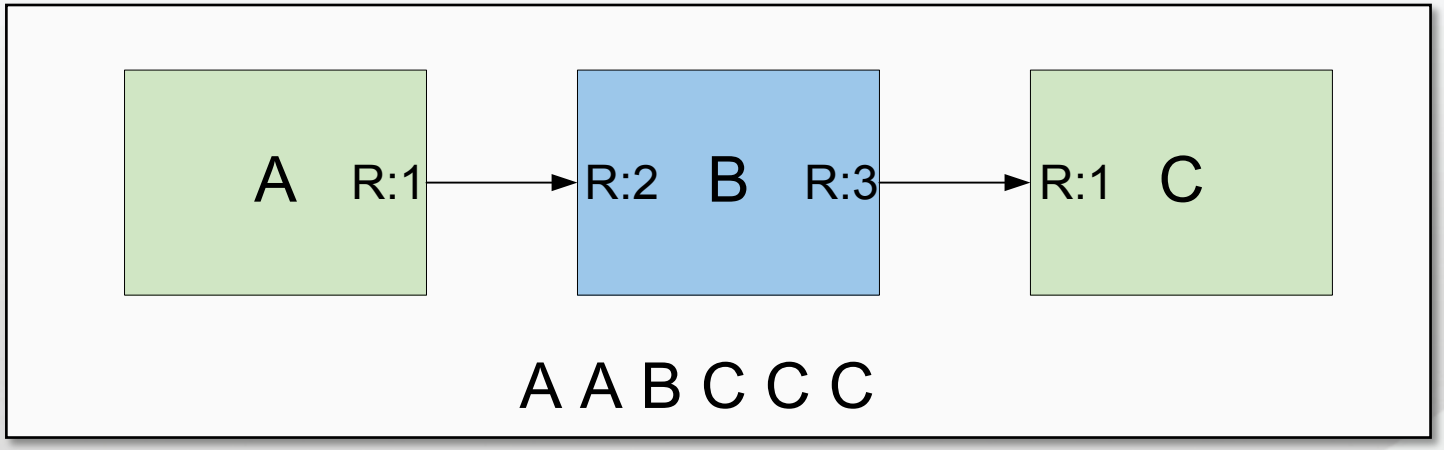
- **Simple firing rule: A module is executed if enough samples are available at its input ports**
- **The function of a module is performed by:**
  - reading from the input ports (thus consuming samples),
  - processing the calculations, and
  - writing the results to the output ports.
- **For synchronous dataflow (SDF), the numbers of read/written samples are constant for each module activation.**
- **The scheduling order follows the signal flow direction.**

Loops in dataflow graphs



- Graphs with loops require a delay to become schedulable
- A delay inserts a sample in the initialization phase

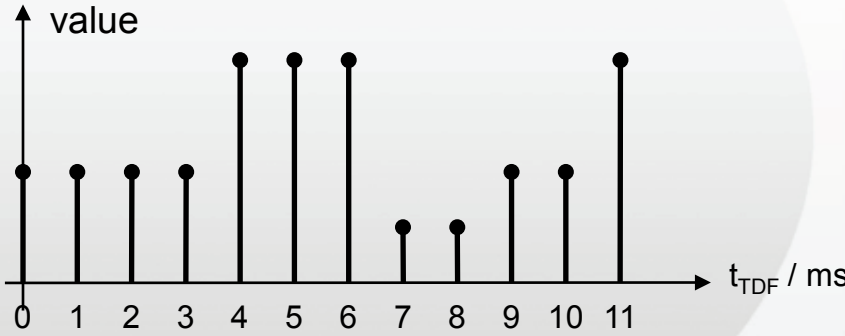
Multi-rate dataflow graphs



- **If the number of read/write sample (rate) for at least one port is >1**
  ➡ multi-rate

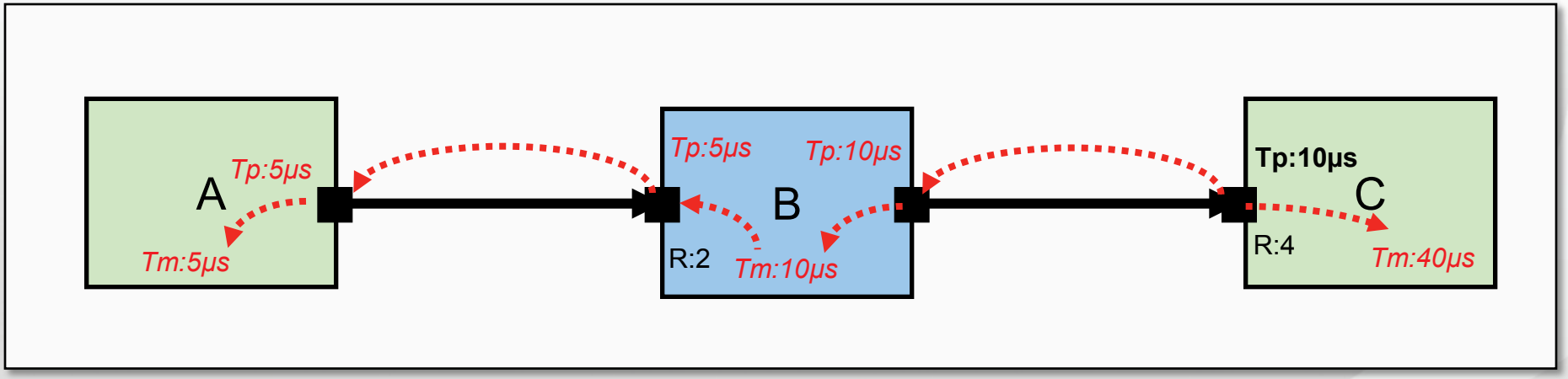
- **The rates in loops must be consistent**

Timed Data Flow (TDF)



- Data flow is an untimed MoC
- Timed Data Flow tags each sample and each module execution with an absolute time point
- Therefore, the time distance (time step) between two samples / two executions is assumed as constant
- This time distance has to be specified
- Enables synchronization with time-driven MoCs like SystemC discrete event and embedding of time-dependent functions like a continuous time transfer function

TDF – Time step propagation



- If more than one time step assigned consistency will be checked

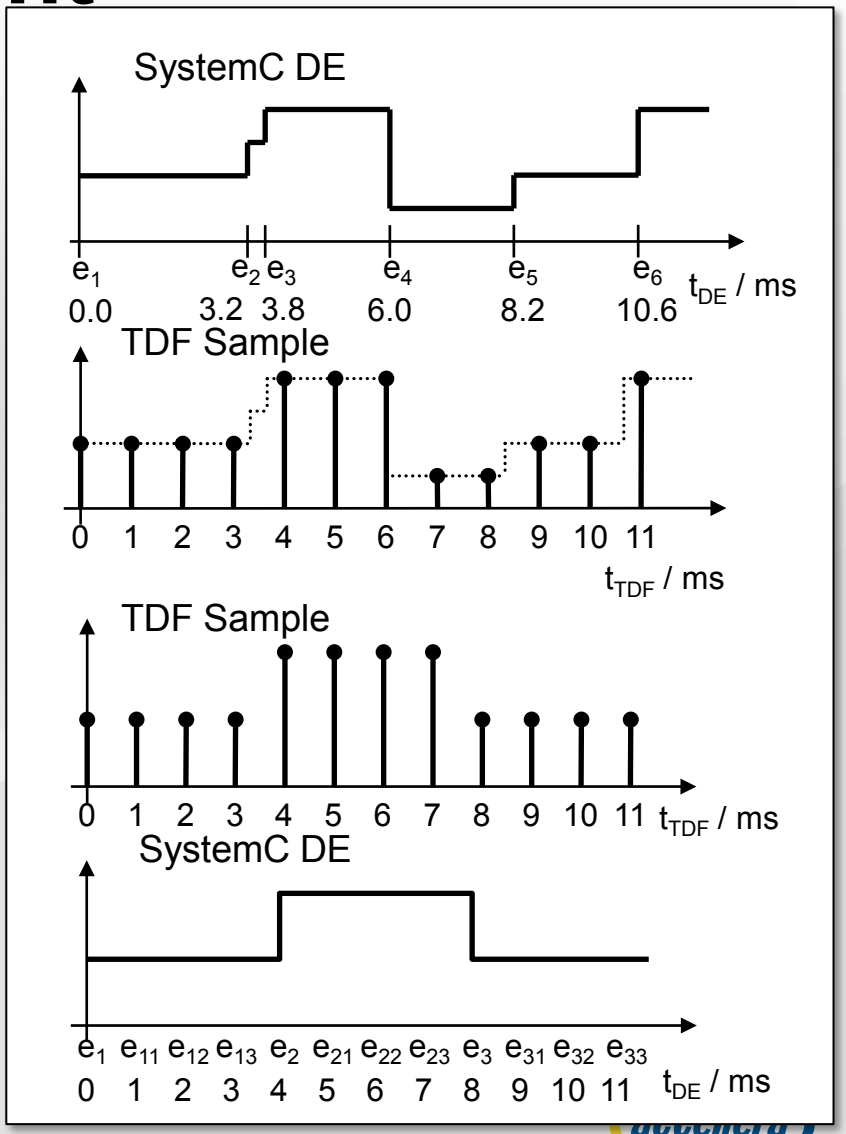
TDF attributes - summary

- rate
- delay
- timestep

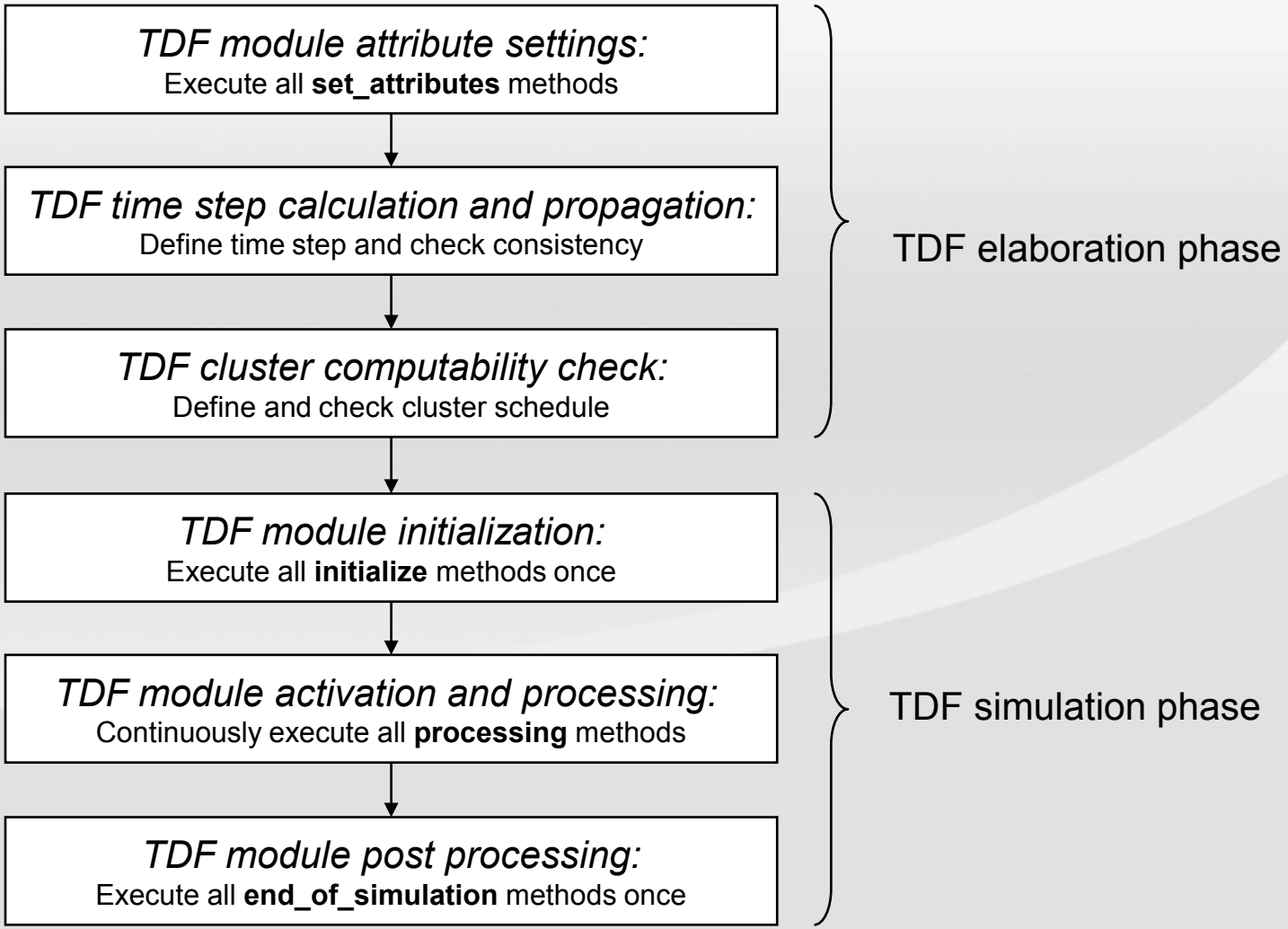
- Port attribute – number of sample for reading / writing during one module execution
- Port attribute – number of sample delay, number of samples to be inserted while initializing
- Port and module attribute – time distance between two samples or two module activations

Synchronization between TDF and SystemC discrete event

- Synchronization between SystemC discrete event (DE) is done by converter ports
- They have the same attributes and access methods like usual TDF ports
- SystemC (DE) signals are sampled at the first Δ of the tagged TDF time point
- TDF samples are scheduled at the first Δ of the tagged TDF time (and thus valid at least at Δ=1)



TDF elaboration and simulation



accellera SYSTEMS INITIATIVE

Timed Data Flow (TDF) primitive module

- Module declaration
- Module declaration macro
- Port declaration: data flow ports
- Port declaration: converter ports (for TDF primitives only)
- Virtual primitive methods called by the simulation kernel – overloaded by the user-defined TDF primitive
- Methods for setting/getting module activation time step
- Constructor macro / constructor

```
struct name:
  public sca_tdf::sca_module {
SCA_TDF_MODULE(name) {

sca_tdf::sca_in< type > port_name;
sca_tdf::sca_out< type > port_name;

sca_tdf::sc_in< type > port_name;
sca_tdf::sc_out< type > port_name;

void set_attributes();
void initialize();
void processing();
void ac_processing();



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

SCA_CTOR(name);
name(sc_core::sc_module_name nm);
```

Structure of Timed Data Flow user-defined primitive module

```
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) {} // constructor
};
```

accellera SYSTEMS INITIATIVE

Set and get TDF port attributes

- Set methods can only be called in set_attributes()
- Get methods can be called in initialize() and processing()
- Sets / gets port rate (number of samples read/write per execution)
- Sets / gets number of sample delays
- Sets time distance of samples / gets calculated/propagated time distance
- Get absolute sample time

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

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

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


sca_time get_time(unsigned long sample);
```

TDF port read and write methods

- **Writes initial value to delay buffer**
  - only allowed in **initialize**()
  - sample_id must be smaller than the number of delays
  - available for all in ports and out ports
- **Reads value from input port**
  - only allowed in **processing**()
  - **sca_tdf::sca_in<T>** or **sca_tdf::sca_de::sca_in<T>**
- **Writes value to output port**
  - only allowed in **processing**()
  - **sca_tdf::sca_out<T>** or **sca_tdf::sca_de::sca_out<T>**

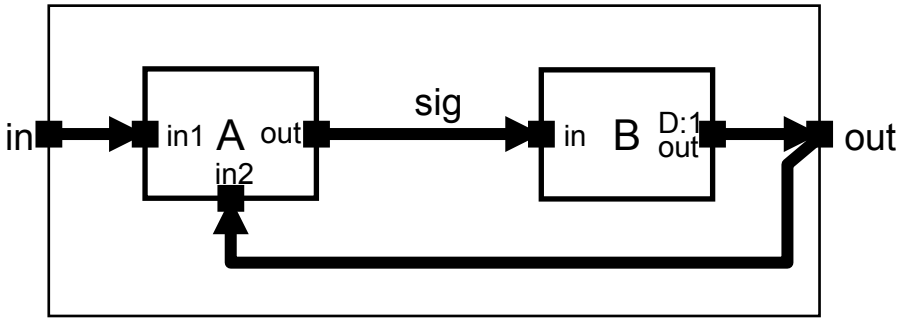
```
void initialize(
  const T& value,
  unsigned long sample_id = 0 )



const T& read(
  unsigned long sample_id = 0 )
operator const T&() const
const T& operator[](
    unsigned long sample_id ) const

void write( const T& value,
  unsigned long sample_id = 0 )
… operator= (const T&)
… operator[](
    unsigned long sample_id )
```

First complete TDF primitive module

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

  void set_attributes()
  {
    set_timestep(1.0, SC_US); // time between activations
    if_out.set_delay(5);      // 5 sample delay at port if_out
  }

  void initialize()
  {  //initialize delay buffer (first 5 sample read by the
     //following connected module input port)
     for(unsigned int i = 0; i < 5; ++i) if_out.initialize(0.0,i);
  }

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

  SCA_CTOR(mixer) : rf_in("rf_in"), lo_in("lo_in"), if_out("if_out") {}
};
```

Hierarchical module example

```
SC_MODULE(hierarchical_module)
{
  sca_tdf::sca_in<double> in;
  sca_tdf::sca_out<double> out;

  mod_a* a; // TDF module
  mod_b* b; // TDF module

  sca_tdf::sca_signal<double> sig;

  SC_CTOR(hierarchical_module) : in("in"), out("out"), sig("sig")
  {
    a = new mod_a ("a");
      a->in1(in);
      a->in2(out);
      a->out(sig);

    b = new mod_b("b");
      b->in(sig);
      b->out(out);
  }
};
```



Simulation Control and Debugging

Tracing of analog signals

- **SystemC AMS has its own trace mechanism:**
  - Analog and digital time scales are not always synchronized
  - **Note:** The VCD file format is in general inefficient for analog
- **Traceable are:**
  - all objects of type **sca_MoC**::**sca_signal**, **sca_eln**::**sca_node** (voltage) and **sc_core**::**sc_signal**
  - Most ELN modules – the current through the module
  - Ports and terminals (traces the connected node or signal)
  - For TDF a traceable variable to trace internal module 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

Viewing wave files

- **Simple Tabular Format:**

```
%time name1 name2  ...
0.0     1         2.1    ...
0.1     1e2       0.3    ...
 :       :         :       :
```

- **A lot of tools like gwave, gaw, Eclipse Impulse can read this format**
- **Can be loaded directly into Matlab/Octave by the load command:**

```
load result.dat
plot(result(:,1), result(:,2));      % plot the first trace versus time
plot(result(:,1), result(:,2:end));  % plot all waves versus time
```

- **For compatibility with SystemC the VCD format is available**
  - However, it is not well suited to store analog waves.
  - VCD waveform viewers usually badly handle analog waves.

Simulation control

- **Time domain simulation – no difference to SystemC**
  - **sc_start(10.0, SC_MS);** *// run simulation for 10 ms*
  - **sc_start();** *// run simulation forever or until sc_stop() is called*
  - **sc_pause();** *// pause simulation for resume with sc_start()*
    // (state of TDF/LSF/ELN implementation-defined)
  - **sc_stop();** *// stop simulation*

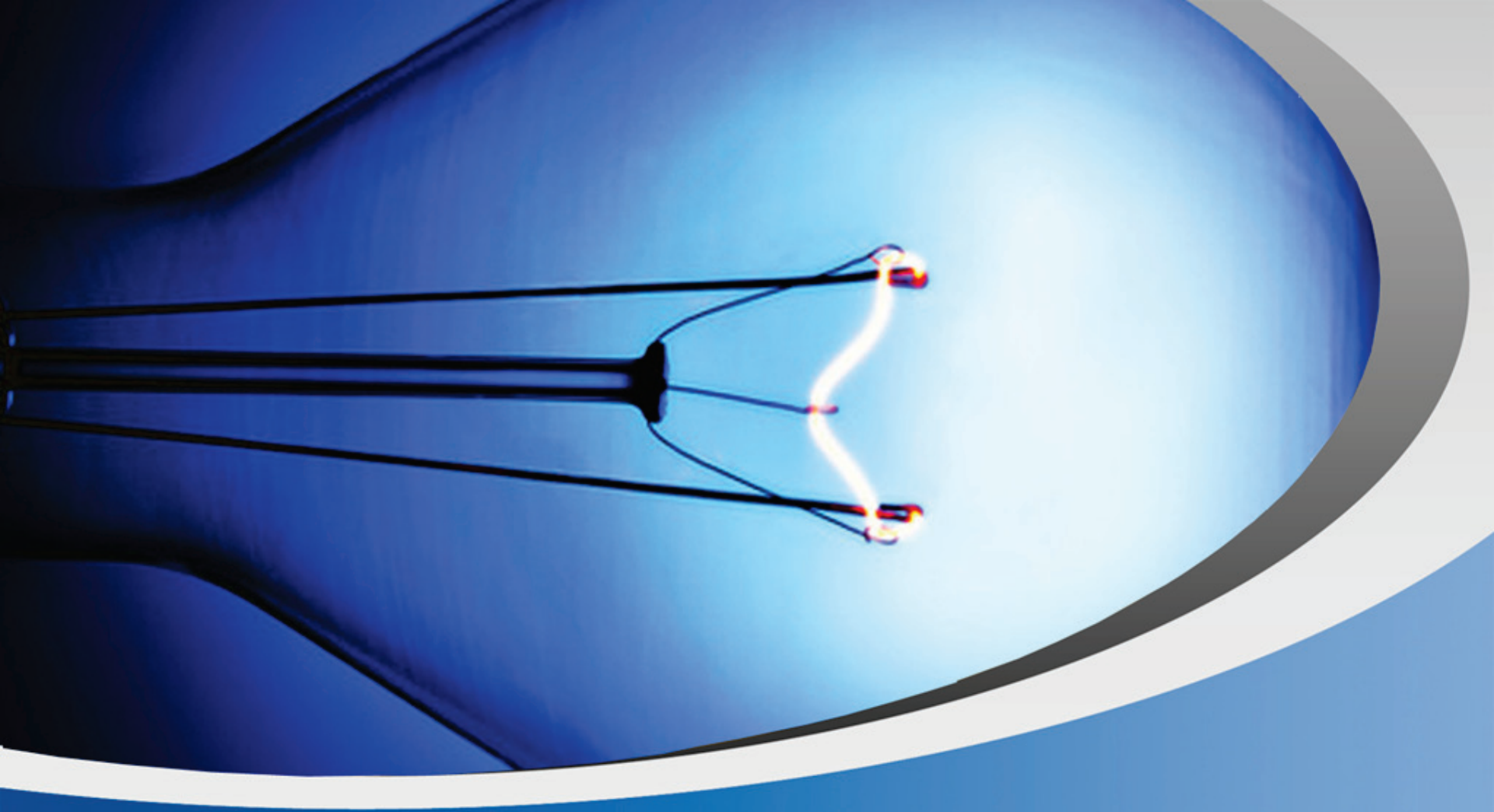
- **AC-domain / AC-noise-domain**
  - **sca_ac_start(1.0,100e3,1000,SCA_LOG);** // ac-domain
  - **sca_ac_noise_start(1.0,100e3,1000,SCA_LOG);** // ac-noise domain

SystemC AMS testbench 1/2

```
#include <systemc-ams.h>

...

int sc_main(int argn, char* argc)
{
  // Instantiate signals, modules, … from arbitrary domains e.g.:
  sca_tdf::sca_signal<double>  s1;     // SystemC AMS TDF MoC
  sca_eln::sca_node            n1;     // SystemC AMS ELN MoC
  sca_lsf::sca_signal          slsf1;  // SystemC AMS LSF MoC
  sc_core::sc_signal<bool>     scsig1; // SystemC discrete-event

  ...
  dut i_dut("i_dut");                  // Instantiate DUT
    i_dut.inp(s1);
    i_dut.ctrl(scsig1);

  // Open VCD and tabular trace files
  sc_trace_file* sctf = sc_create_vcd_trace_file("sctr");
    sc_trace(sctf, scsig1, "scsig1");
  sca_trace_file* satf = sca_create_tabular_trace_file("tr.dat");
    sca_trace(satf, n1, "n1");
  ...
```

SystemC AMS testbench 2/2

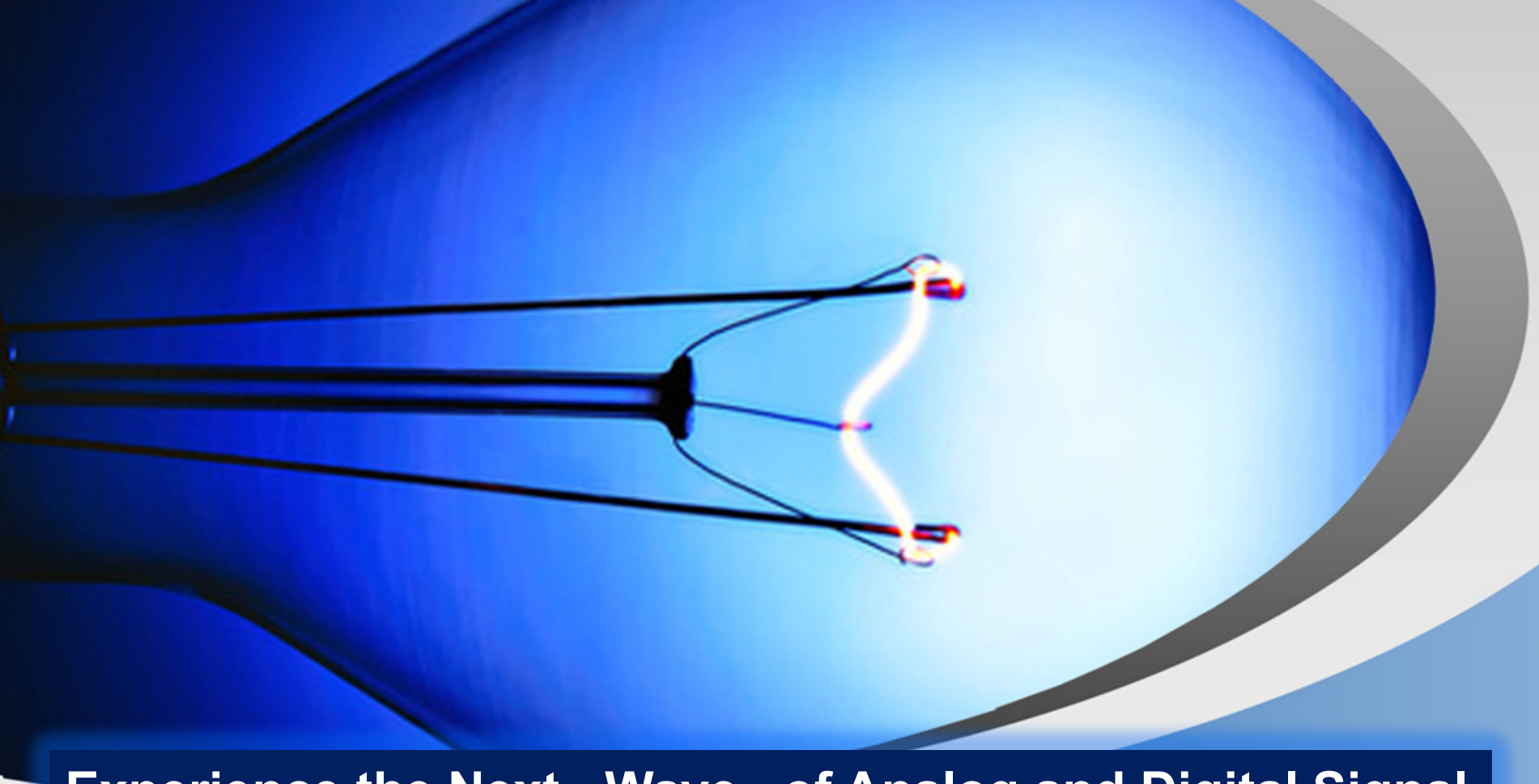
```
  sc_start(2.0, SC_MS); // start time domain simulation for 2ms

  satf->disable();      // stop writing to trace file

  sc_start(2.0, SC_MS); // continue time domain simulation 2ms

  satf->enable();       // continue writing to trace file

  sc_start(2.0, SC_MS); // continue time domain simulation with 2ms

  satf->set_mode(sca_sampling(1.0, SC_US)); // sample results with
                                            // 1us time distance

  sc_start(100.0, SC_MS);               // continue time domain simulation

  sc_close_vcd_trace_file(sctf);        // close SystemC vcd trace file

  sca_close_tabular_trace_file(satf);  // close tabular trace file
}
```



Thank you

Continue with Lab 1: Sine Source Connected to a Sink

accellera SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

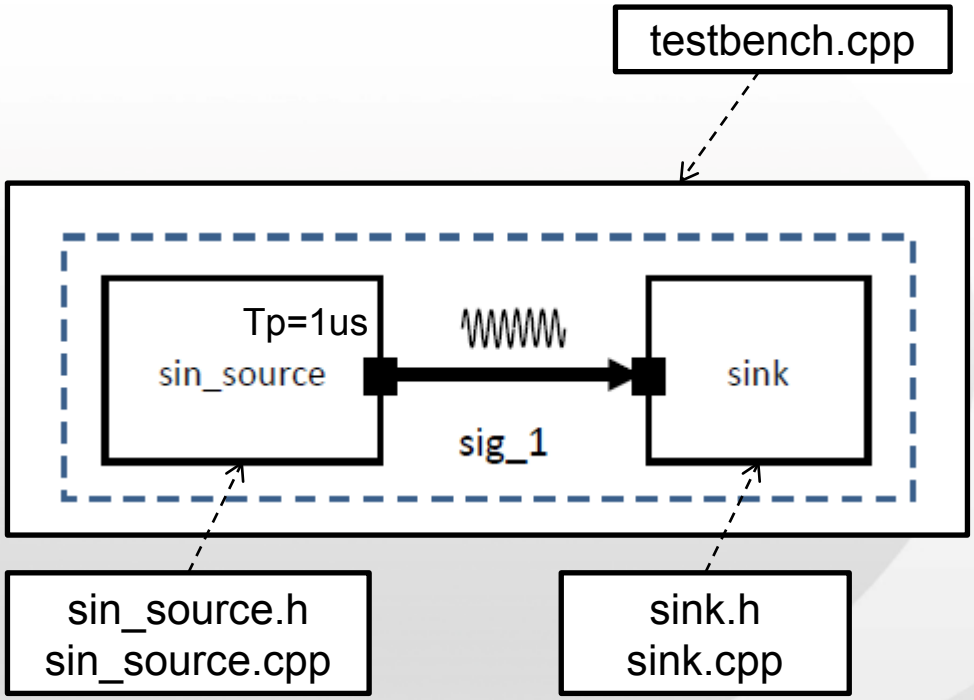
Lab 1: Sine Source Connected to a Sink

accellera
SYSTEMS INITIATIVE

Lab 1

- **Lab1 has 3 incomplete AMS modules you have to complete and correct ("fill in the blanks" approach, mini-tasks)**
  - Lab 1a: Build a sinusoidal source (use sin_source_with_noise.h and sin_source_with_noise.cpp)
  - Lab 2b: prefilter (prefilter.h and prefilter.cpp)
  - Lab 2c: adc_sd (adc_sd.h and adc_sd.cpp)
  - Lab 2d: comb_filter (comb_filter.h and comb_filter.cpp)

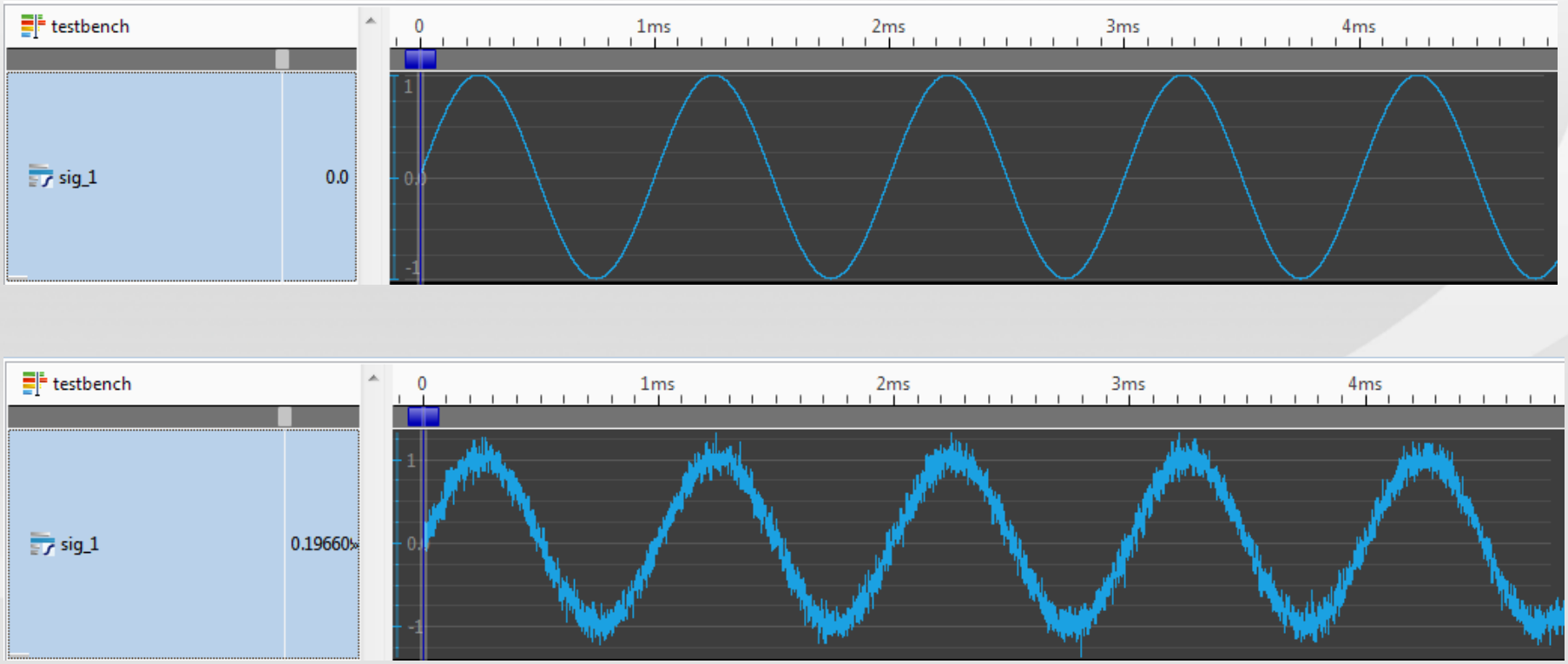
Lab 1



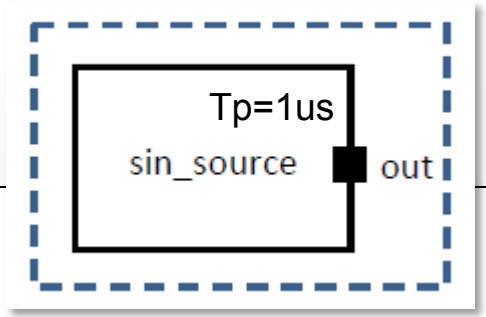
Lab1 exercises

- **lab1a:**
  - Implement sinusoidal source (in sin_source.cpp)
  - Store sinusoidal signals in tabular trace file (in testbench.cpp)
- **lab1b**
  - Print sinusoidal values to cout for each time step (in sink.cpp)
- **lab1c (optional)**
  - Add Gaussian noise to signal (e.g., using Marsaglia Polar Method)
- **Execute simulation and examine the results**
  - Study the values via the console
  - Look at the waves using the tabular trace file

Lab1 results

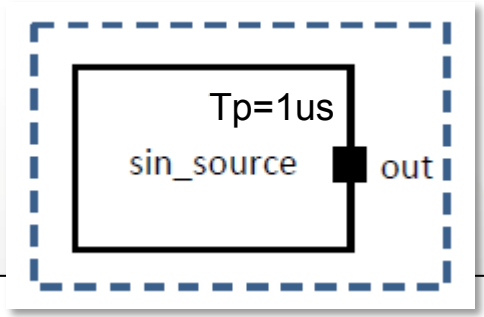


sin_source.h



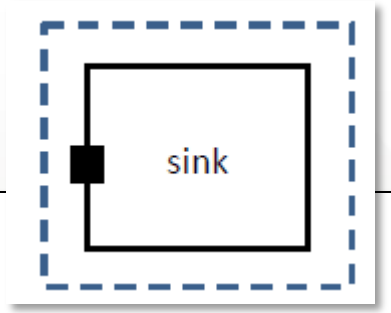

```
#ifndef SIN_SOURCE_H
#define SIN_SOURCE_H

#include <systemc-ams.h>

SCA_TDF_MODULE(sin_source)                  // Declare a TDF module
{
  sca_tdf::sca_out<double> out;             // TDF output port

  double ampl;                              //   amplitude
  double freq;                              //   frequency

  void set_attributes();                    // Set TDF attributes

  void processing();                        // Describe time-domain behavior

  SCA_CTOR(sin_source)                      // Constructor of the TDF module
  : out("out"),                             // Name the port(s)
    ampl(1.0), freq(1e3)                    // Initial values for ampl and freq
  {}
};                                          // Never forget ";" at end of
                                            // class definition!

#endif // SIN_SOURCE_H
```

sin_source.cpp



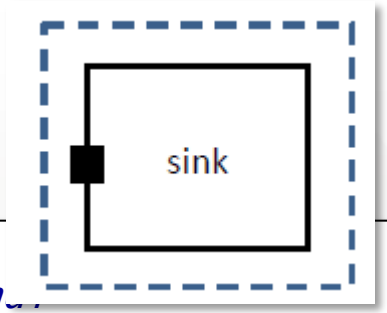

```
#include "sin_source.h"
#include <cmath>                    // for M_PI and std::sin

void sin_source::set_attributes() // Set TDF attributes
{
  out.set_timestep(1.0, SC_US);   // Set time step of output port
}

void sin_source::processing()     // Describe time-domain behavior
{
  double t = out.get_time().to_seconds(); // Get current time of the sample
  double x = ampl * std::sin(2.0 * M_PI * freq * t); // Calc. the sine wave
  out.write(x);                           // Write sample to the output
}
```

sink.h



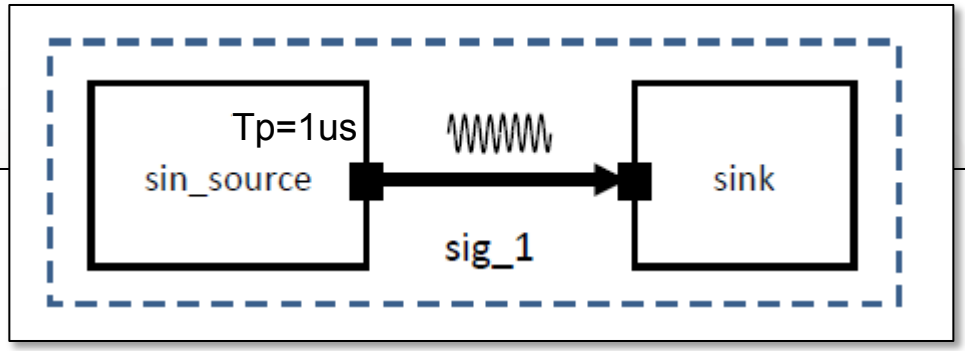

```
#ifndef SINK_H
#define SINK_H

#include <systemc-ams>

SCA_TDF_MODULE(sink)
{
  sca_tdf::sca_in<double> in;      // TDF input port

  SCA_CTOR(sink)                   // Constructor of the TDF module
  : in("in")                       // Name the port(s)
  {}

  void processing();               // Describe time-domain behaviour
};

#endif // SINK_H
```

sink.cpp




```
#include "sink.h"
#include <iostream>                  // for std::cout and std::endl

void sink::processing()              // Describe time-domain behavior
{
  std::cout << this->name()          // Display name of module
            << " @ "
            << this->get_time()      // Show module time
            << ": "
            << in.read()             // Read value from input port
            << std::endl;            // and display it
}
```

testbench.cpp

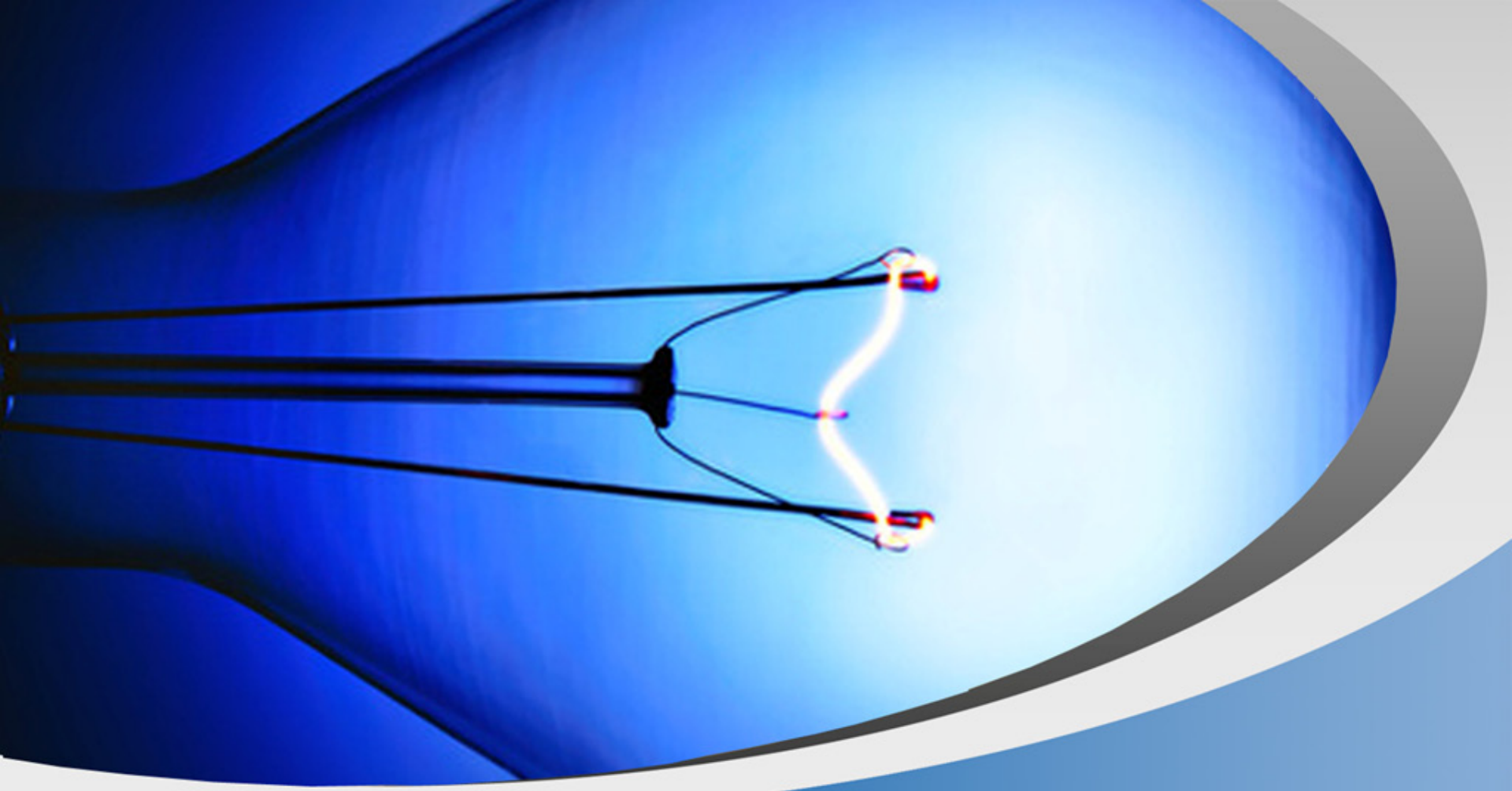



```
#include <systemc-ams.h>

#include "sin_source.h"
#include "sink.h"

int sc_main(int argc, char *argv[]) {          // SystemC main program

  sca_tdf::sca_signal<double> sig_1("sig_1"); // Signal between source and sink

  sin_source src_1("src_1");                   // Instantiate source
     src_1.out(sig_1);                         // Connect (bind) with signal

  sink sink_1("sink_1");                       // Instantiate sink
     sink_1.in(sig_1);                         // Connect (bind) with signal

  sca_trace_file* tfp =                        // Open trace file
    sca_create_tabular_trace_file("testbench");

    sca_trace(tfp, sig_1, "sig_1");            // Define which signal to trace

  sc_start(10.0, SC_MS);                       // Start simulation for 10 ms

  sca_close_tabular_trace_file(tfp);           // Close trace file

  return 0;                                    // Exit with return code 0
}
```

Gaussian noise [1]

```
#include <cstdlib> // for std::rand
#include <cmath>   // for std::sqrt and std::log

double gauss_rand(double variance)
{
  double rnd1, rnd2, Q, Q1, Q2;

  do
  {
    rnd1 = static_cast<double>(std::rand()) / RAND_MAX;
    rnd2 = static_cast<double>(std::rand()) / RAND_MAX;

    Q1 = 2.0 * rnd1 - 1.0;
    Q2 = 2.0 * rnd2 - 1.0;
    Q = Q1 * Q1 + Q2 * Q2;
  }
  while (Q > 1.0);

  return std::sqrt(variance) * ( std::sqrt( -2.0 * std::log(Q) / Q ) * Q1 );
}
```

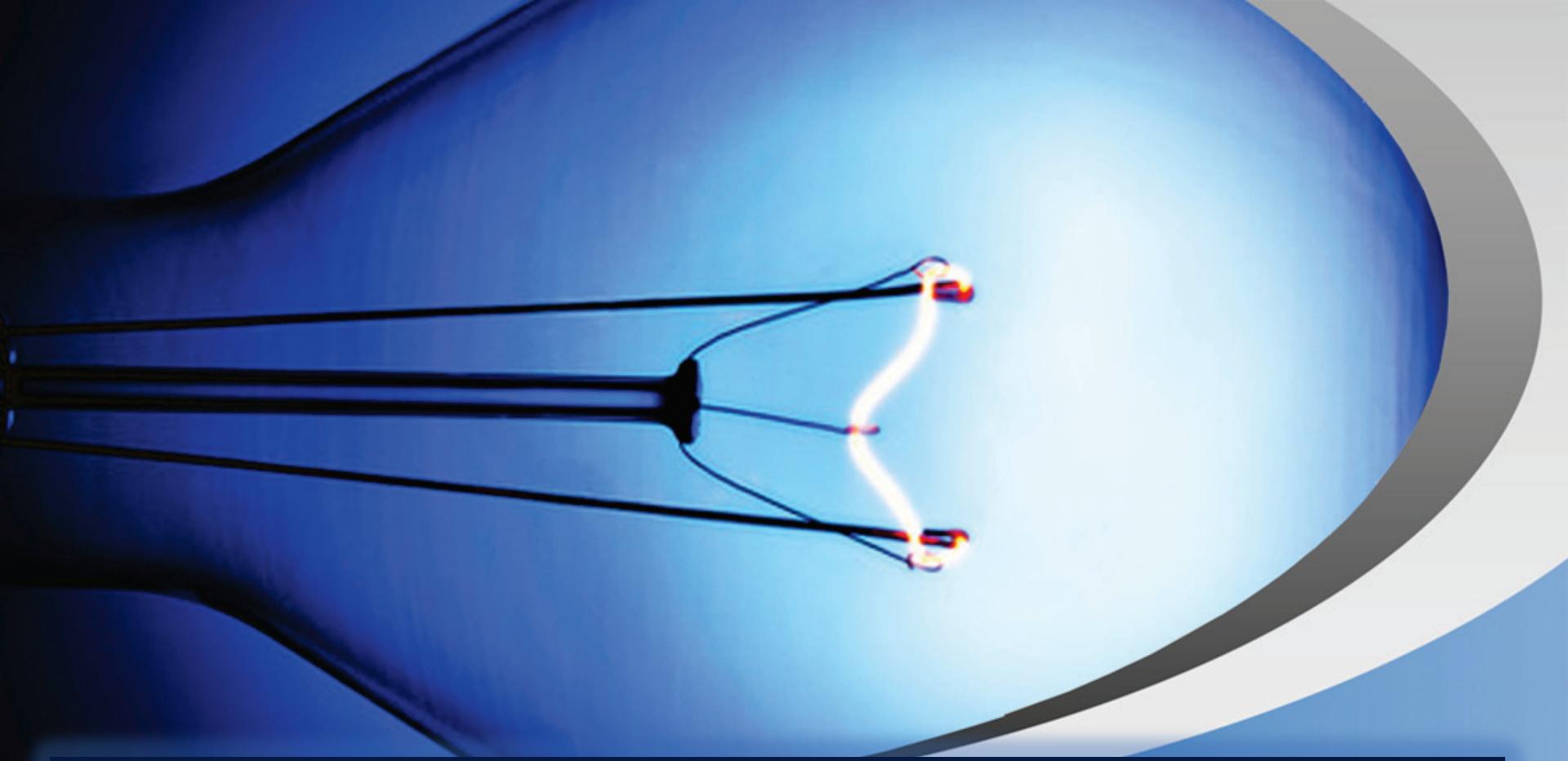
[1] http://en.wikipedia.org/wiki/Marsaglia_polar_method



Thank you

Continue with Session 2: Models of Computation

accellera
SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Session 2: Models of Computation

François Pêcheux, Université Pierre et Marie Curie
Torsten Mähne, Université Pierre et Marie Curie

accellera
SYSTEMS INITIATIVE

Outline

- **Models of Computation and Analysis types**
  - Linear dynamic behavior in TDF
  - Linear Signal Flow (LSF)
  - Electrical Linear Networks (ELN)
  - Small-signal frequency-domain analysis (AC, AC-noise)

Linear Dynamic Behavior in TDF

Linear dynamic behavior in TDF 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 / denominator 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**

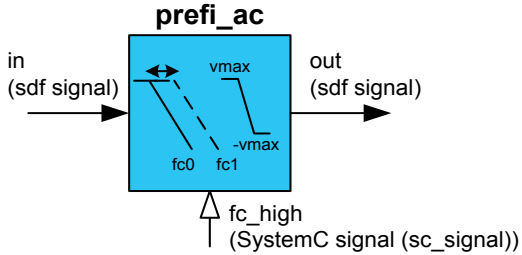
Linear dynamic behavior in TDF 2/2

- **The equation systems will be represented and calculated by dedicated SystemC AMS objects:**
  - **sca_tdf::sca_ltf_nd** - Numerator / denominator 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 an "artificial" object (sca_tdf::sca_ct_proxy or sca_tdf::sca_ct_vector_proxy)**
  - This object performs the time discretization (sampling) in dependency of the context – this makes the usage more comfortable and increases the accuracy.
  - It additionally permits a very fast calculation for multi-rate systems.

TDF module with Linear Transfer Function (LTF)

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

  // control signal from SystemC
  // (connected to sc_signal<bool>)
  sca_tdf::sc_in<bool> fc_high;
  double fc0, fc1, 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_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 cutoff 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;
  }
};
```

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



Linear Signal Flow (LSF)

Linear Signal Flow (LSF) modeling

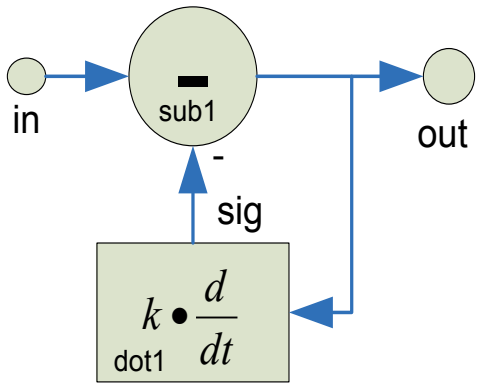
- **Library of predefined elements**
- **Allows for the description of arbitrary linear equation systems**
- **Several converter modules to/from TDF and SystemC are provided**
- **Models for switching behavior like mux / demux**
- **LSF models are always hierarchical models**
- **Ports:**
  - **sca_lsf::sca_in** - LSF input port
  - **sca_lsf::sca_out** - LSF output port
- **Channel / Signal:**
  - **sca_lsf::sca_signal**

List of 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::sca_de::sca_source (sca_lsf::sca_de_source)
- sca_lsf::sca_de::sca_gain (sca_lsf::sca_de_gain)
- sca_lsf::sca_de::sca_mux (sca_lsf::sca_de_mux)
- sca_lsf::sca_de::sca_demux (sca_lsf::sca_de_demux)
- sca_lsf::sca_de::sca_sink (sca_lsf::sca_de_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

  sca_lsf::sca_dot dot1;    // module instances
  sca_lsf::sca_sub sub1;

  mylsfmodel( sc_core::sc_module_name nm, double fc = 1.0e3 )
  : in("in"), out("out"), sig("sig"), dot1("dot1"), sub1("sub1")
  {
    dot1.x(out);
    dot1.y(sig);

    sub1.x1(in);
    sub1.x2(sig);
    sub1.y(out);
  }
};
```



Electrical Linear Networks (ELN)

Electrical Linear Networks (ELN)

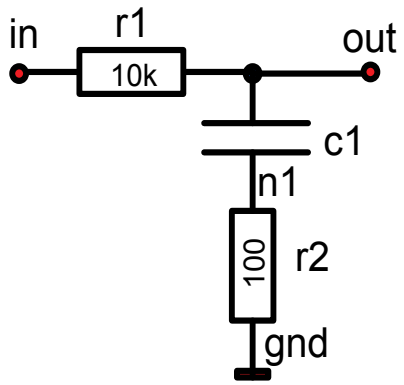
- **Library of predefined elements**
- **Allows for the description of arbitrary conservative linear electrical networks**
- **Several converter modules to/from TDF and SystemC are provided (connecting to *sca_tdf::sca_signal* and *sc_core::sc_signal*)**
- **Models for switching behavior like switches**
- **ELN models are always hierarchical models**
- **Ports:**
  - ***sca_eln::sca_terminal*** – conservative terminal
- **Channel / Node:**
  - ***sca_eln::sca_node*** – conservative node
  - ***sca_eln::sca_node_ref*** – reference node, node voltage is always zero

List of 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::sca_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::sca_de::sca_vsource (sca_eln::sca_de_vsource)
- sca_eln::sca_de::sca_isource …
- sca_eln::sca_tdf::sca_r …
- sca_eln::sca_tdf::sca_l …
- sca_eln::sca_tdf::sca_c …
- sca_eln::sca_de::sca_r …
- sca_eln::sca_de::sca_l …
- sca_eln::sca_de::sca_c …

Example: ELN language constructs

```
SC_MODULE(eln_filter)
{
  sca_eln::sca_terminal in;
  sca_eln::sca_terminal out;
  sca_eln::sca_r r1, r2;
  sca_eln::sca_c c1;
  SC_CTOR(eln_filter)
    : in("in"), out("out"), r1("r1", 10e3), r2("r2", 100.0),
     c1("c1", 100e-6), n1("n1"), gnd("gnd")
  {
    r1.p(in);
    r1.n(out);
    c1.p(out);
    c1.n(n1);
    r2.p(n1);
    r2.n(gnd);
  }
 private:
  sca_eln::sca_node n1;
  sca_eln::sca_node_ref gnd;
};
```



Solvability of analog equations
(applicable for ELN and LSF MoC)

Not all analog systems that can be described are **solvable**	• Do not connect voltage sources in parallel or current sources in series • Zero-valued resistors represent a short cut (voltage source with value zero) • Do not define floating nodes • Do always have a path to a reference node
Not all **theoretically solvable** analog systems **can be solved** by the applied numerical algorithm	• The time constants of the network should be at least ~5 times larger than the simulation step width • Prevent the use of extremely high/low values and large differences in the dimensions

Small-signal Frequency-domain Analysis (AC, AC-noise)

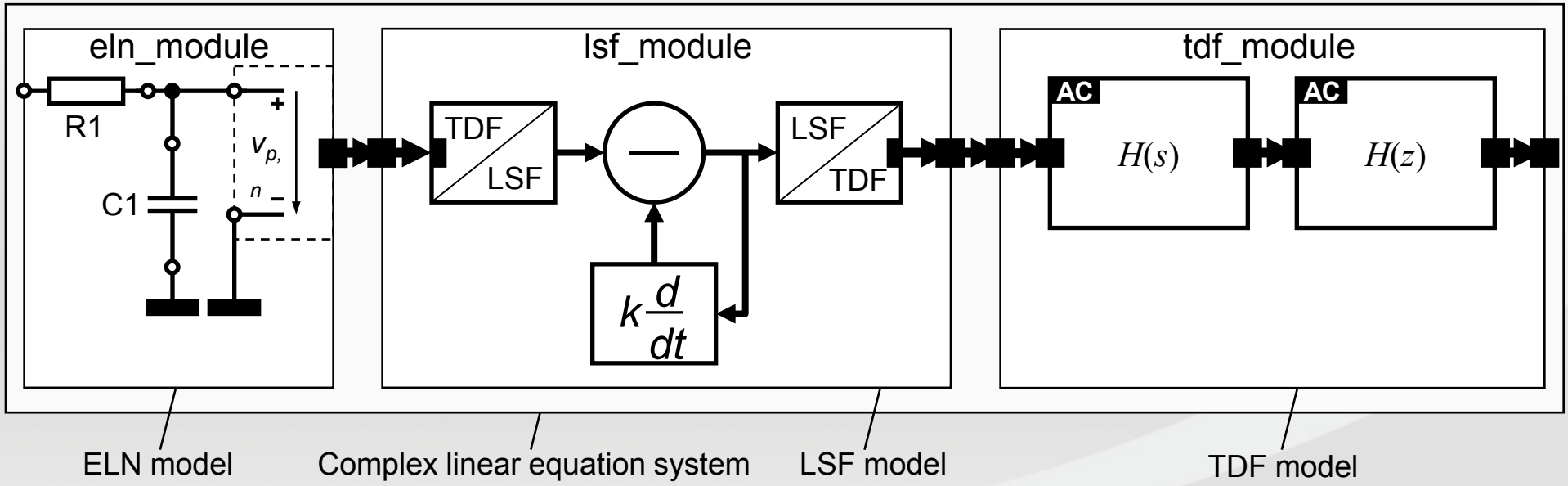
Analysis types

- **Transient time domain is driven by the SystemC kernel**
  - Thus, the SystemC **sc_core::sc_start** function call controls the simulation.
- **Two different kinds of small-signal frequency-domain analysis (AC analysis) are available**
  - AC-analysis
  - AC-noise-analysis

Small-signal frequency-domain analysis (AC Analysis)

- **AC analysis**
  - Calculates linear complex equation system stimulated by AC-sources
- **AC noise domain**
  - solves the linear complex equation system 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 inside the method ac_processing (otherwise, the output values are assumed as zero)**
- **This transfer function can depend on the current time domain state (e.g., the setting of a control signal)**

Small-signal frequency-domain analysis



- **Linear equation system contribution for LSF/ELN:**

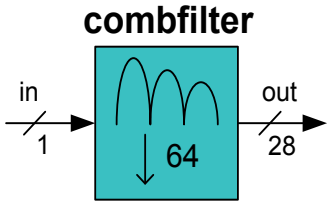
  q(t) = Adx + Bx -> q(f) = Ajwx(f) + Bx(f)

  Sources

Small-signal frequency-domain description for TDF models

```
SCA_TDF_MODULE(combfilter)
{
  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(combfilter) : ...
  {
    ...
  }
};
```



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

Simulation control

- **Time domain – no difference to SystemC**
  - **sc_start(10.0,SC_MS);** *// run simulation for 10 ms*
  - **sc_start();** *// run simulation forever or until sc_stop() is called*

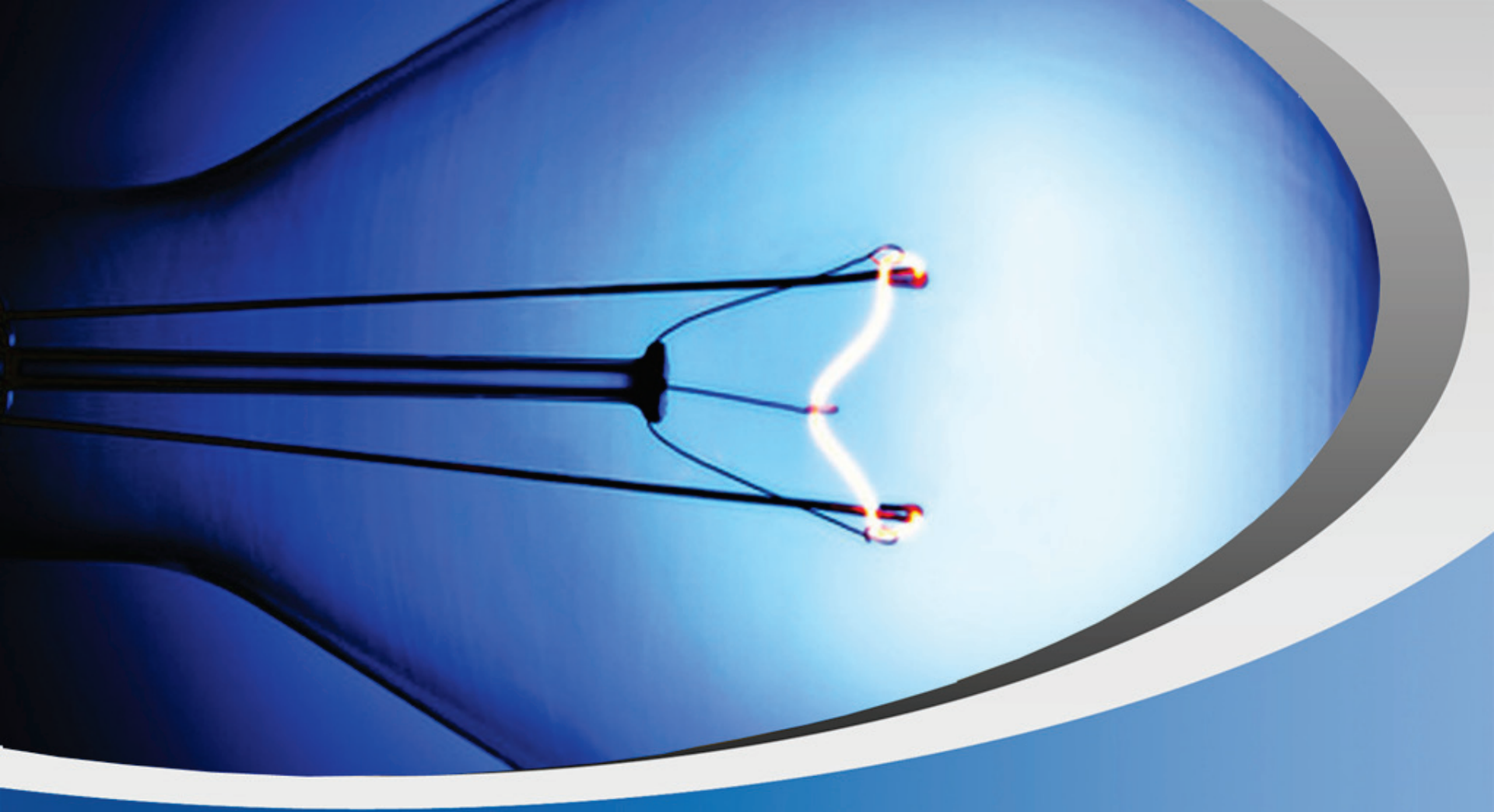
- **AC-domain / AC-noise-domain**
  - Run simulation from 1Hz to 100kHz, calculate 1000 points logarithmically spaced:
    - **sca_ac_start(1.0, 100e3, 1000, SCA_LOG);** // ac-domain
    - **sca_ac_noise_start(1.0, 100e3, 1000, SCA_LOG);** // ac-noise domain
  - Run simulation at frequency points given by a std::vector<double>:
    - **sca_ac_start(frequencies);** // ac-domain
    - **sca_ac_noise_start(frequencies);** // ac-noise domain

SystemC AMS testbench 1/2
(incl. AC tracing)

```
#include <systemc-ams.h>

...
int sc_main(int argc, char** argv)
{
  //instantiate signals, modules, … from arbitrary domains e.g.:
  sca_tdf::sca_signal<double>  s1;
  sca_eln::sca_node            n1;
  sca_lsf::sca_signal          slsf1;
  sc_core::sc_signal<bool>     scsig1;
  ...
  dut i_dut("i_dut");
    i_dut->inp(s1);
    u_dut->ctrl(scsig1);
  ...
  sc_trace_file* sctf = sc_create_vcd_trace_file("sctr");
  sc_trace(sctf, scsig1, "scsig1");
  ...
  sca_trace_file* satf = sca_create_tabular_trace_file("tr.dat");
  sca_trace(satf, n1, "n1");
  ...
```

SystemC AMS testbench 2/2
(incl. AC tracing)

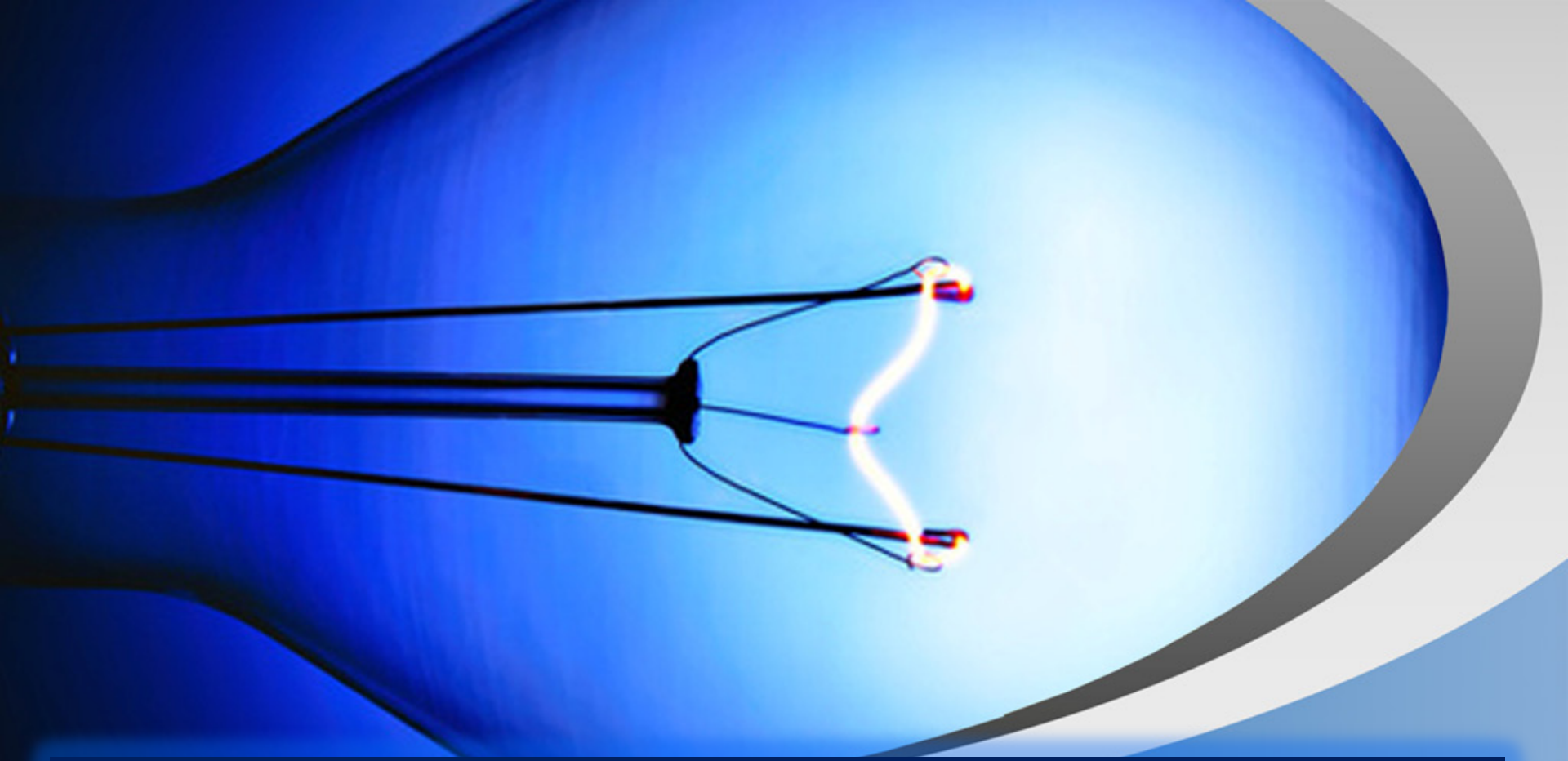
```
  sc_start(2.0, SC_MS); // start simulation for 2ms
  satf->disable();      // stop writing to trace file
  sc_start(2.0, SC_MS); // continue time domain simulation 2ms
  satf->enable();       // continue writing to trace file
  sc_start(2.0, SC_MS); // continue time domain simulation 2ms
  // close time domain file, open ac-file
  satf->reopen("my_tr_ac.dat");
  sca_ac_start(1.0, 1e6, 1000, SCA_LOG); // calculate ac at current
                                         // operating point
  // 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 time domain simulation
  sc_close_vcd_trace_file(sctf);       // close SystemC vcd trace file
  sca_close_tabular_trace_file(satf);  // close tabular trace file
}
```



Thank you

Continue with Lab 2: Filtering and A/D Conversion

accellera SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Lab 2: Filtering and A/D Conversion

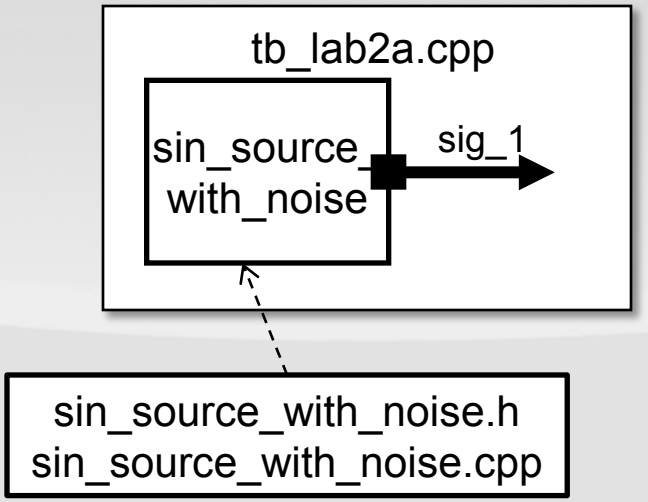
accellera
SYSTEMS INITIATIVE

Lab 2

- **Lab2 uses 4 incomplete AMS modules you have to complete and correct (“fill in the blanks” approach, mini tasks indicated by TODO)**
  - lab 2a: sin_source_with_noise (sin_source_with_noise.h and sin_source_with_noise.cpp)
  - lab 2b: prefilter (prefilter.h and prefilter.cpp)
  - lab 2c: adc_sd (adc_sd.h and adc_sd.cpp)
  - lab 2d: comb_filter (comb_filter.h and comb_filter.cpp)

Lab 2a

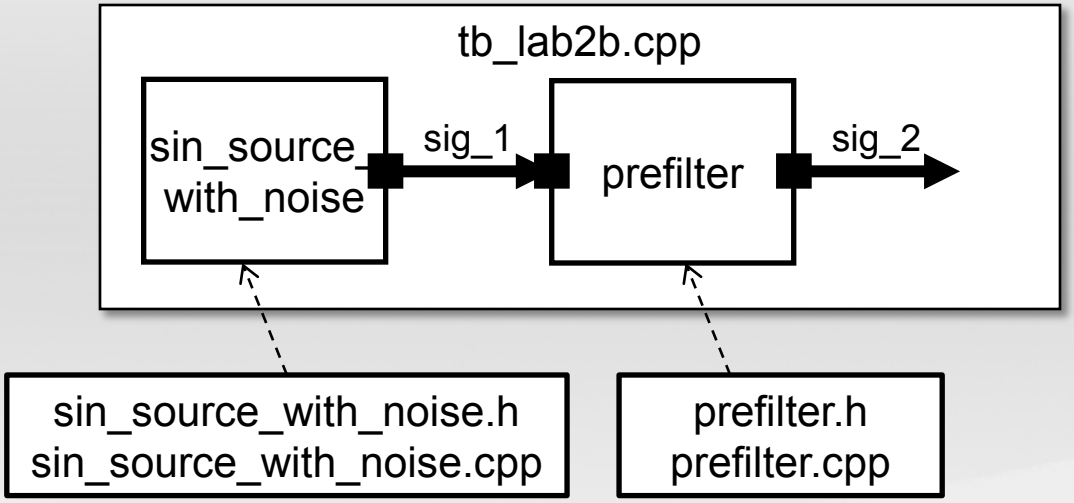
- **Lab2a contains 4 test benches to be run one after another, once assigned TODOs are completed.**
- **First testbench: Is Lab1 still working?**



4 TODOs

Lab 2b

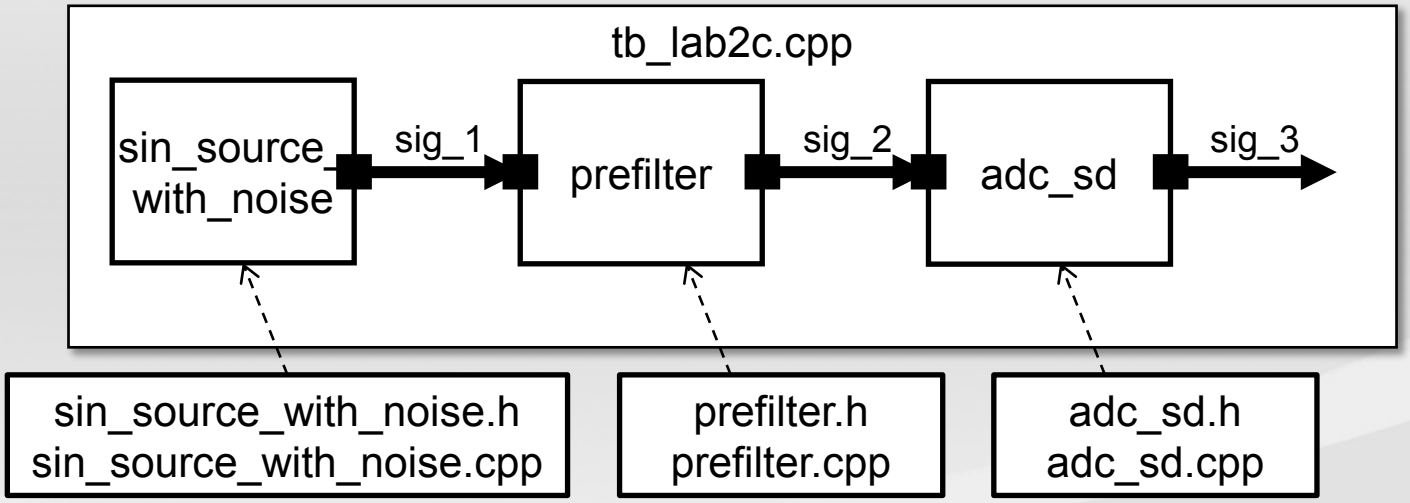
- **Lab 2b: testbench: Add a prefilter, use TDF LTF primitives**



8 TODOs

Lab 2c

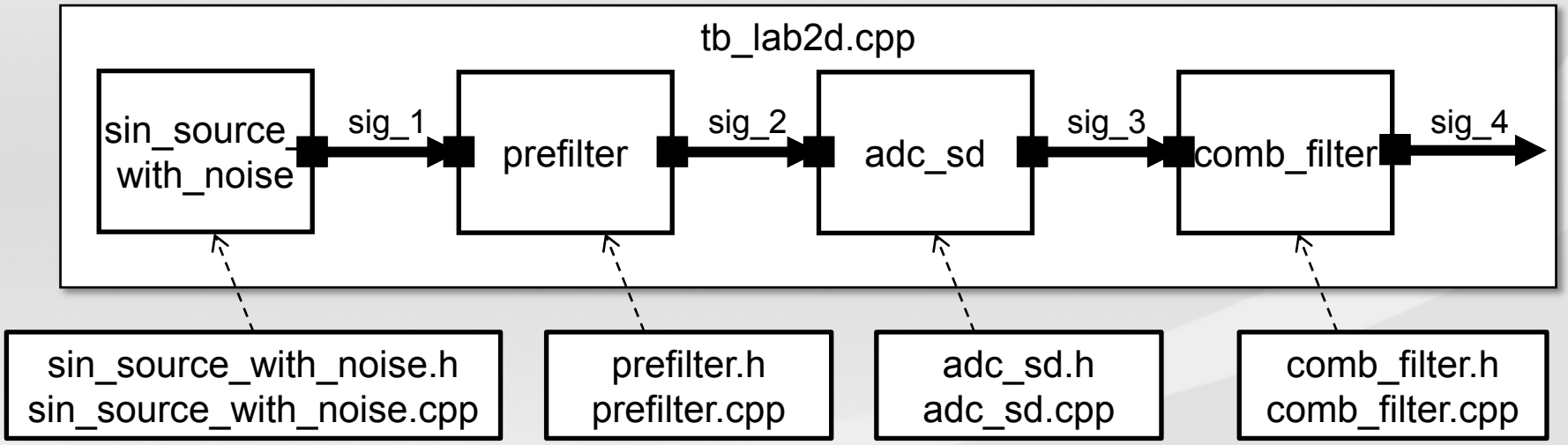
- **Lab 2c: Model and instantiate the ΣΔ ADC converter**



5 TODOs

Lab 2d

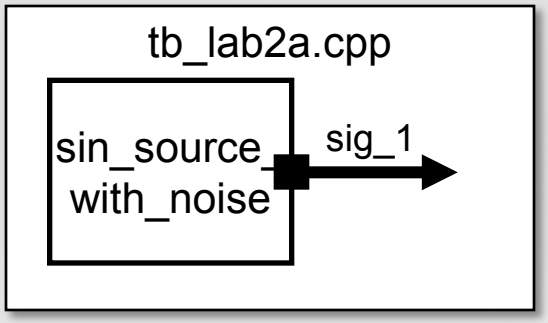
- **Lab 2d: Model and instantiate the comb filter**



5 TODOs

Lab 2a: First testbench

- **First testbench: Is Lab1 still working?**

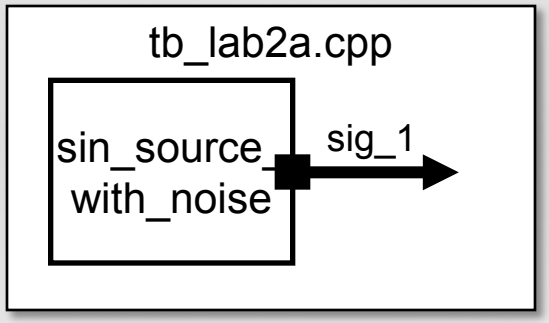


TODO 1.1

Check if the test bench correctly compiles.

Lab 2a: First testbench

- **First testbench: Is Lab1 still working?**

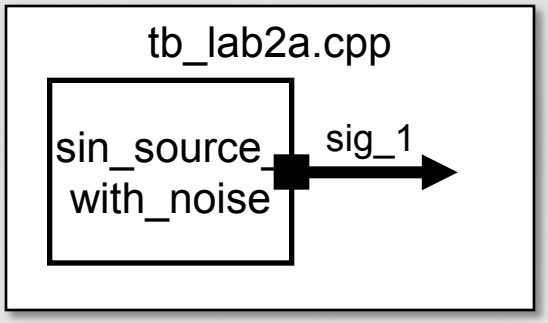


TODO 1.2

Perform a 10 ms time-domain simulation.

Lab 2a: First testbench

- **First testbench: Is Lab1 still working?**

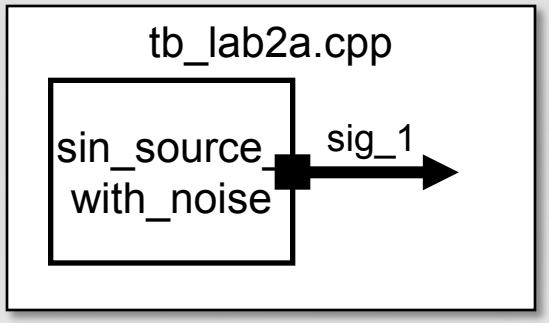


TODO 1.3

Create a tabular trace file and trace global signal **sig_1**. Run the simulator and view the results in the waveform viewer.

Lab 2a: First testbench

- **First testbench: Is Lab1 still working?**

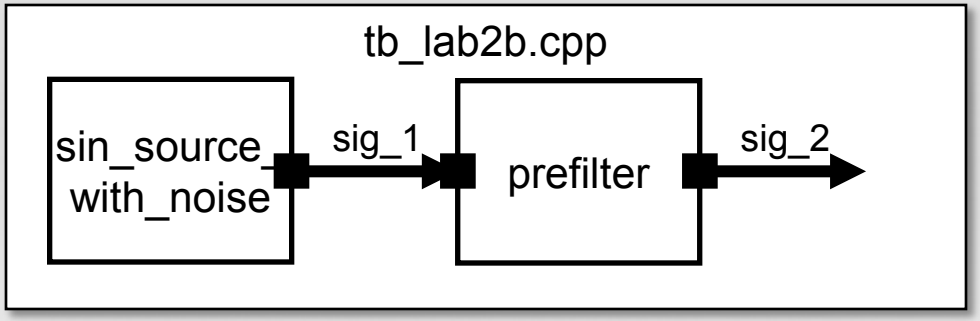


TODO 1.4

Create a new tabular trace file, perform an AC analysis (1Hz to 1MHz, 1000 points logarithmically spaced) and view the results in the waveform viewer.

Lab 2b: Second testbench

- **Second testbench, lab2b: Model and instantiate the prefilter**



TODO 2.1

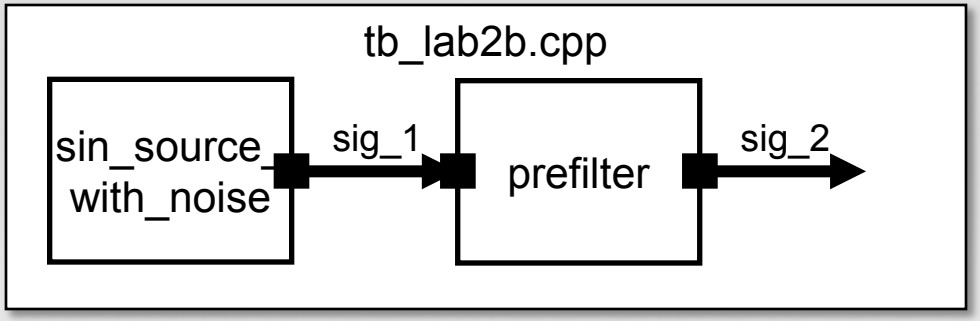
In the **prefilter.cpp** file, assign 1 kHz to the cut-off frequency **fc** member variable in the constructor.

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

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

Lab 2b: Second testbench

- **Second testbench, lab2b: Model and instantiate the prefilter**



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

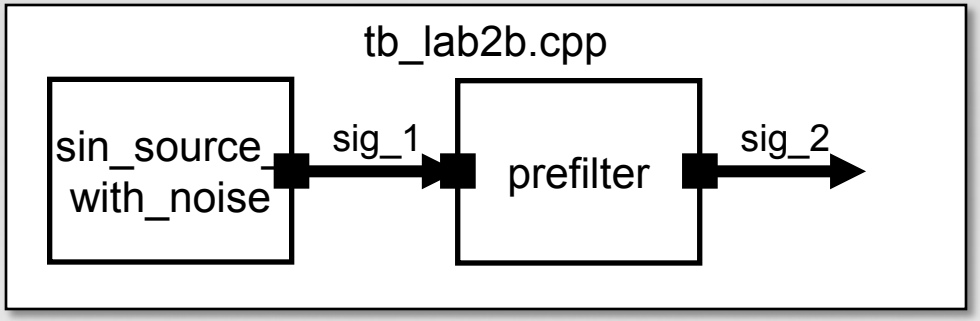
$$H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}$$

TODO 2.2

On paper, determine the appropriate numerator and denominator coefficients for the filter representation using ND Laplace Transfer Function (Hint given). Once this is done, open the **prefilter.h** file and complete MINITASK 2.3.

Lab 2b: Second testbench

- **Second testbench, lab2b: Model and instantiate the prefilter**



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

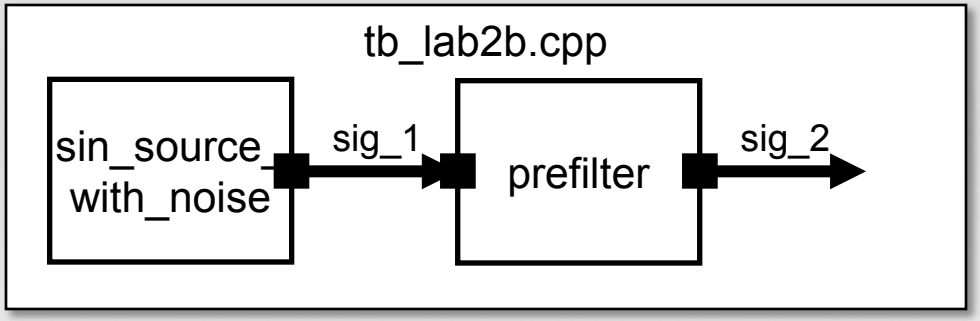
$$H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}$$

TODO 2.3

In the **prefilter.h** file, add double vectors named **num** and **den** as member variables. Also add a member variable **ltf1** of type **sca_tdf::sca_ltf_nd**. Then go to the **initialize()** function in **prefilter.cpp** and accordingly set the **num** and **den** coefficients.

Lab 2b: Second testbench

- **Second testbench, lab2b: Model and instantiate the prefilter**



TODO 2.4

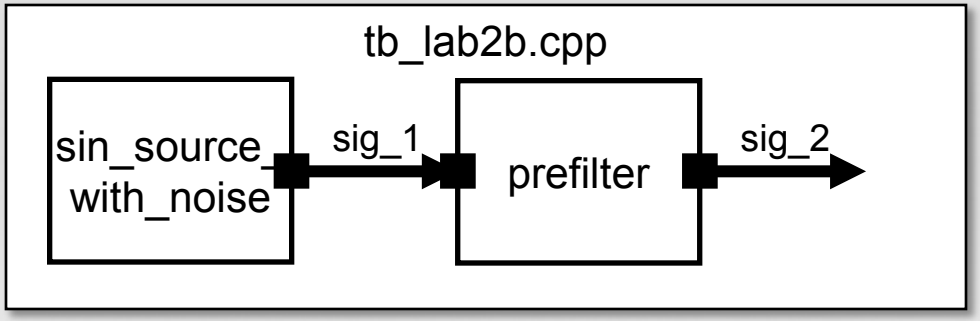
Then, go to the **initialize()** function in **prefilter.cpp** and accordingly set the **num** and **den** coefficients.

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

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

Lab 2b: Second testbench

- **Second testbench, lab2b: Add the prefilter**



TODO 2.5

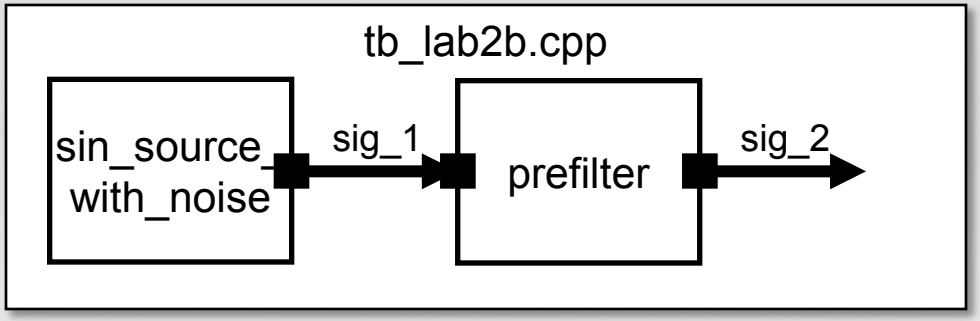
In the **prefilter.cpp** file, in the **processing()** function, write the appropriate LTF function call using **ltf1**, **num**, **den**, input sample and output sample.

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

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

Lab 2b: Second testbench

- **Second testbench, lab2b: Add the prefilter**



TODO 2.6

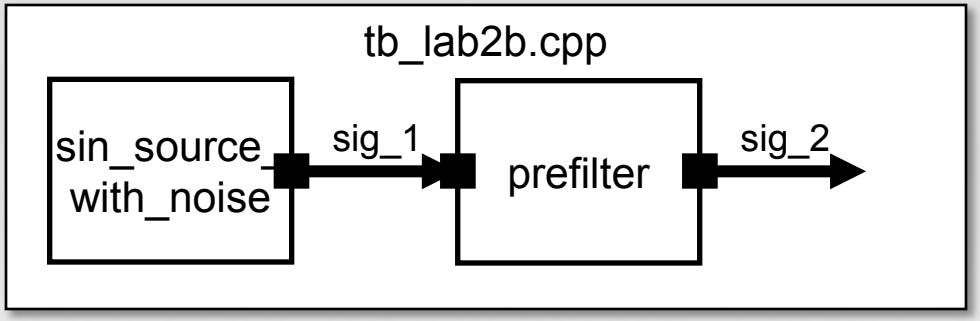
In the **tb_lab2b.cpp** file, trace signals **sig_1** and **sig_2**, run a 10 ms time-domain simulation and view the results in the waveform viewer.

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

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

Lab 2b: Second testbench

- **Second testbench, lab2b: Add the prefilter**



TODO 2.7

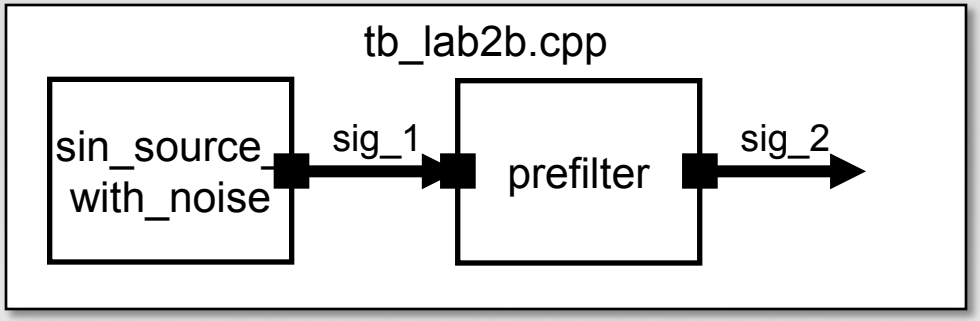
In the **prefilter.cpp** file, modify the **ac_processing()** function to perform a **sca_ac_ltf_nd()** function call.

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

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

Lab 2b: Second testbench

- **Second testbench, lab2b: Add the prefilter**



TODO 2.8

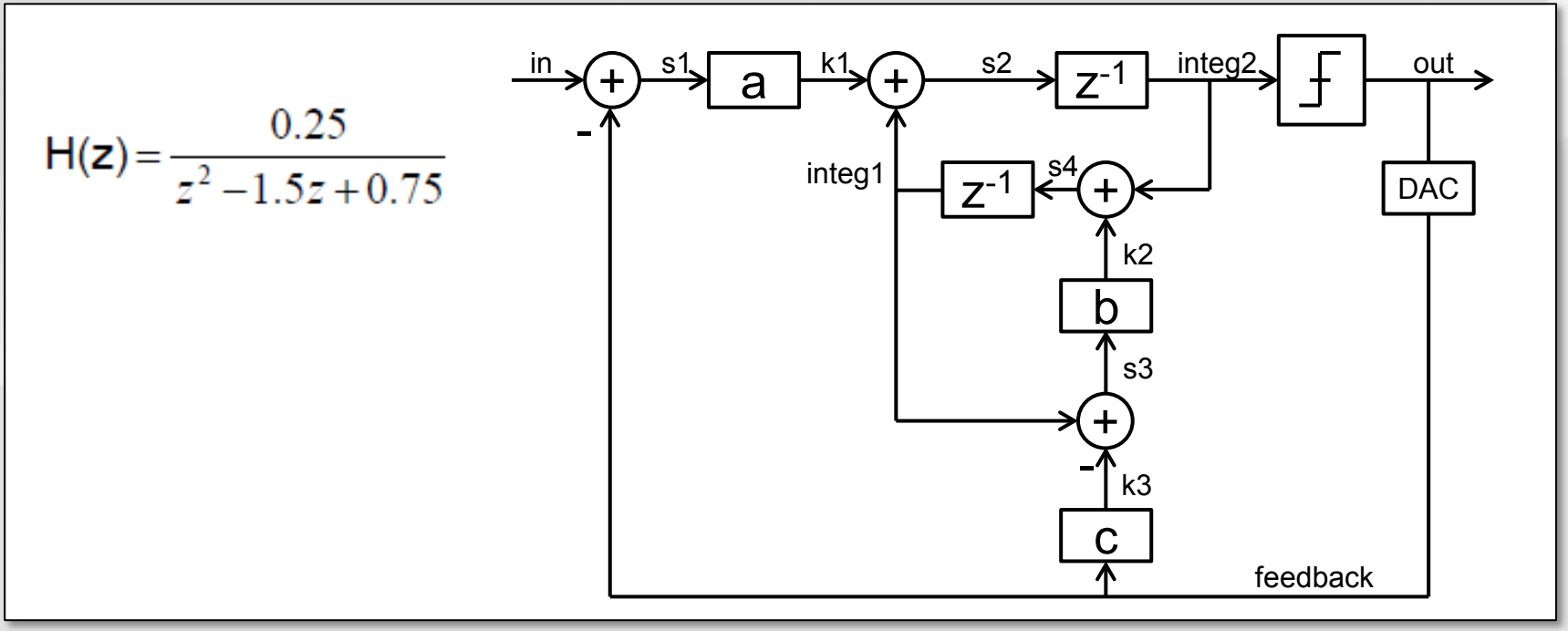
In the **tb_lab2b.cpp** file, perform an AC analysis (use previous parameters).

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

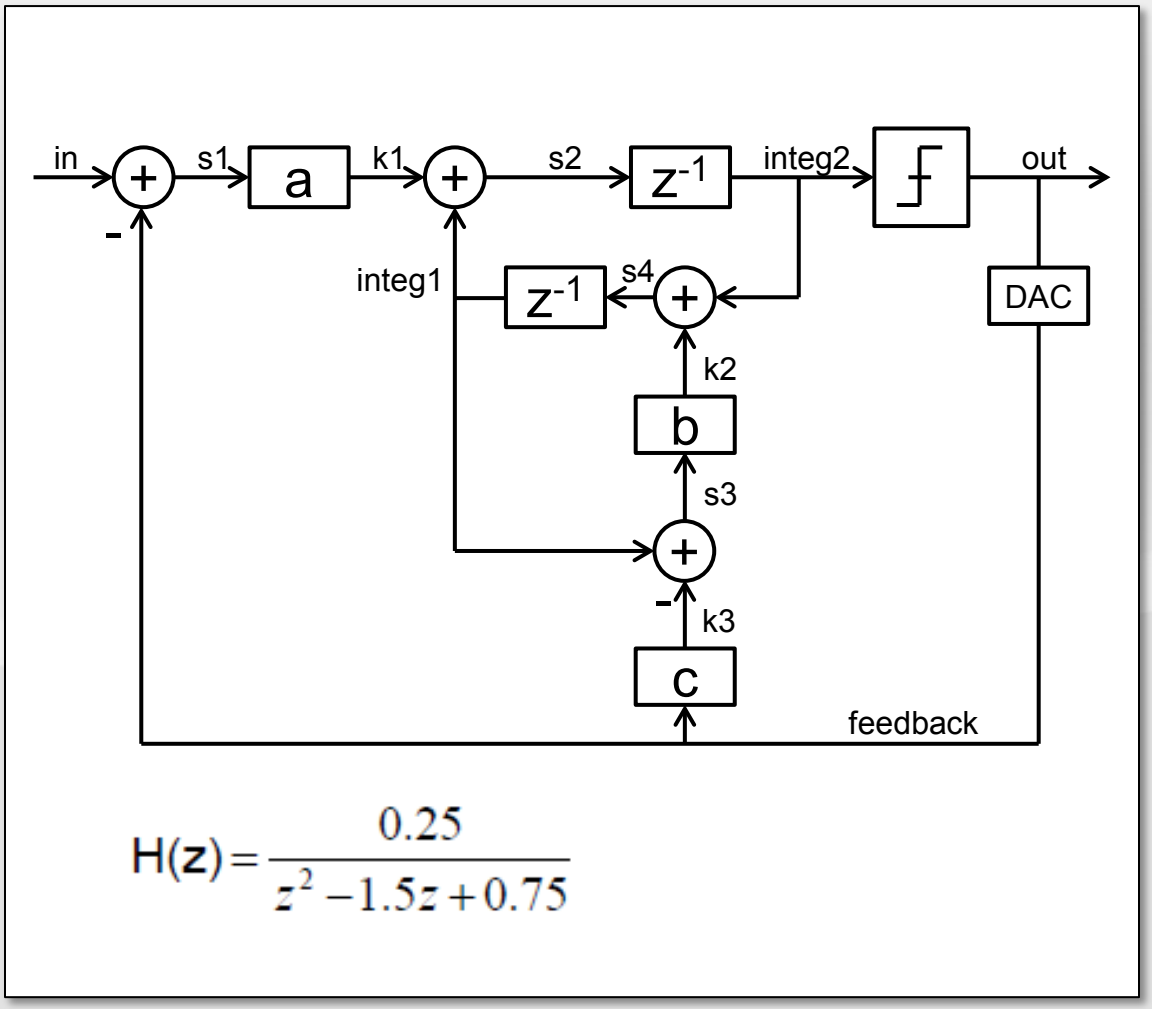
$$H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}$$

Lab 2c: Third testbench

- **Implement ADC_SD behavior in the TDF module adc_sd.h, adc_sd.cpp as C++-code**
  - Figure below shows implementation (with a=0.5, b=0.5, c=1.0)
- **TDF input port of type double, TDF output port of type bool**



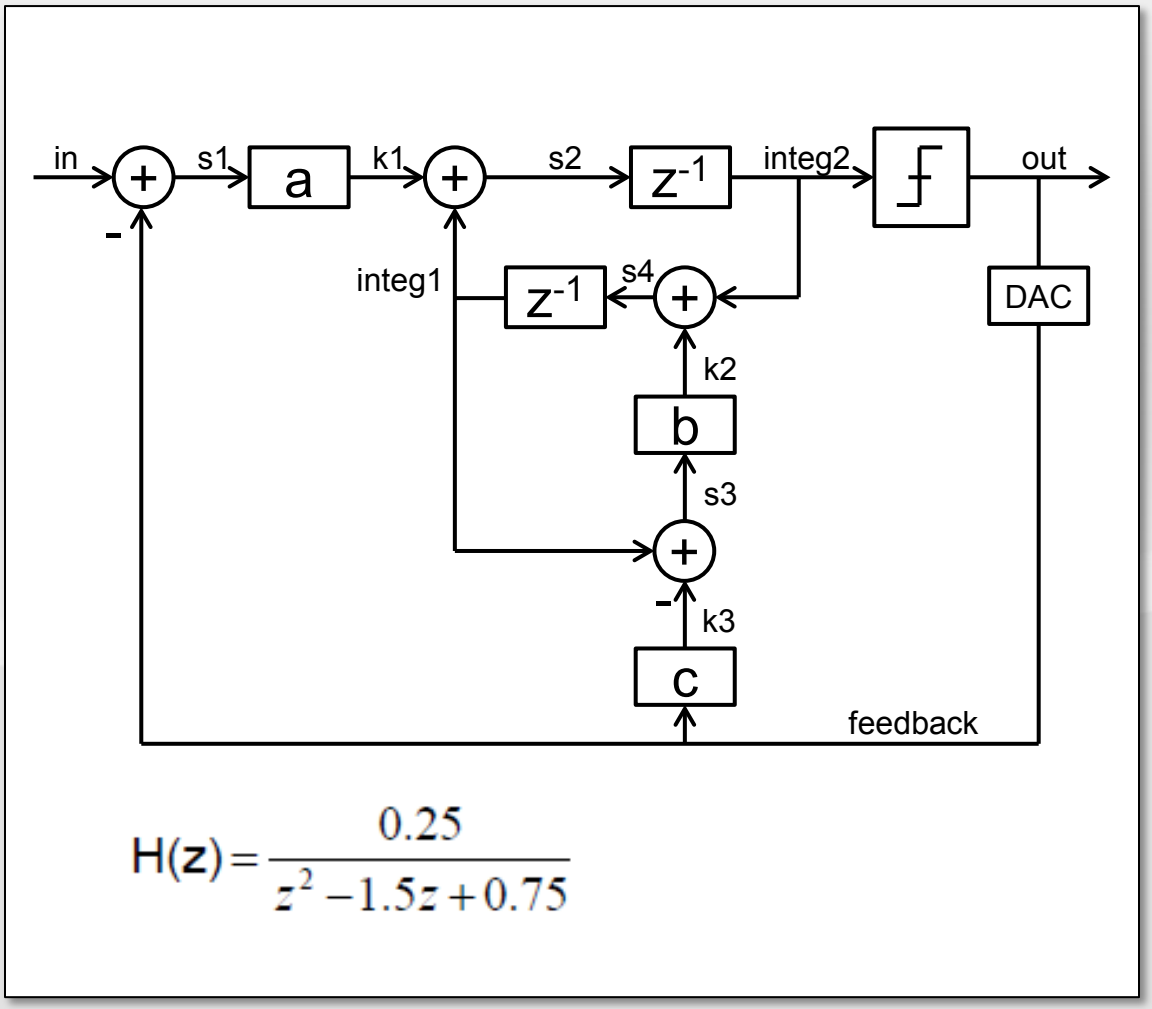
Lab 2c: Third testbench



TODO 3.1

In the **adc_sd.h** file, define two private double member variables, named **integ1** and **integ2**. These variables contain the previous/integrated values (cf. to Figure). They are read at the beginning of the **processing()** function and written at the end.

Lab 2c: Third testbench

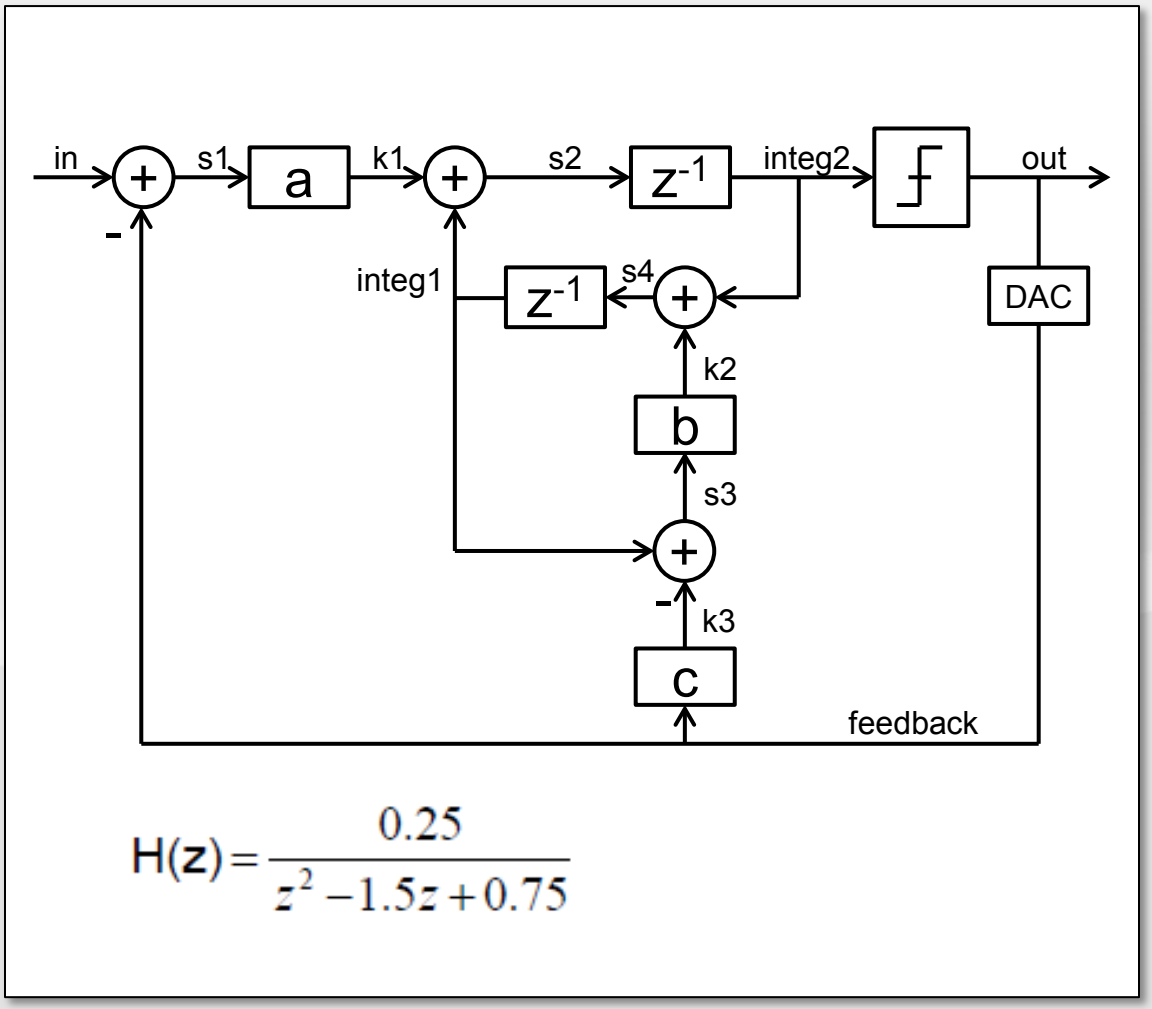


TODO 3.2

In the **adc_sd.cpp** file, in the **processing()** function, declare a local bool variable **output**. Assign it with true (if **integ2**>=0.0) or false (if **integ2**<0.0).

Also declare a local double variable **feedback**. Assign it with 1.0 (if **integ2**>=0.0) or -1.0 (if **integ2**<0.0). Write the output value to the out TDF Port.

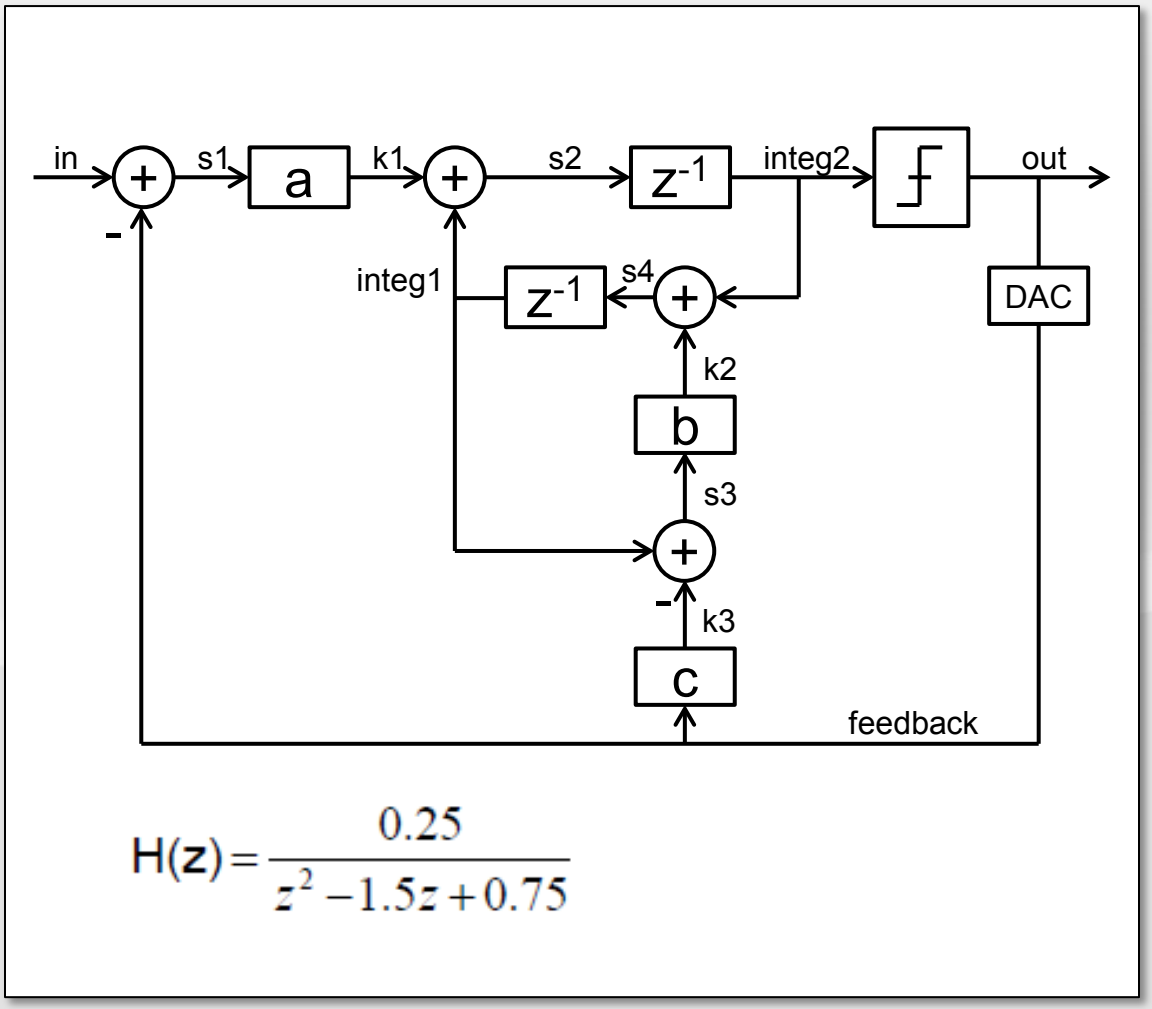
Lab 2c: Third testbench



TODO 3.3

In the **adc_sd.cpp** file, in the **processing()** function, with help of the Figure, declare and assign **s1**, **k1**, **s2**, **k2**, **k3**, **s3**, **s4** in the right order. Some assignments (i.e. **s2** or **s3**) depend on the previous/integrated values, **integ1** and **integ2** that are just used as RHS at this stage of the **processing()** function.

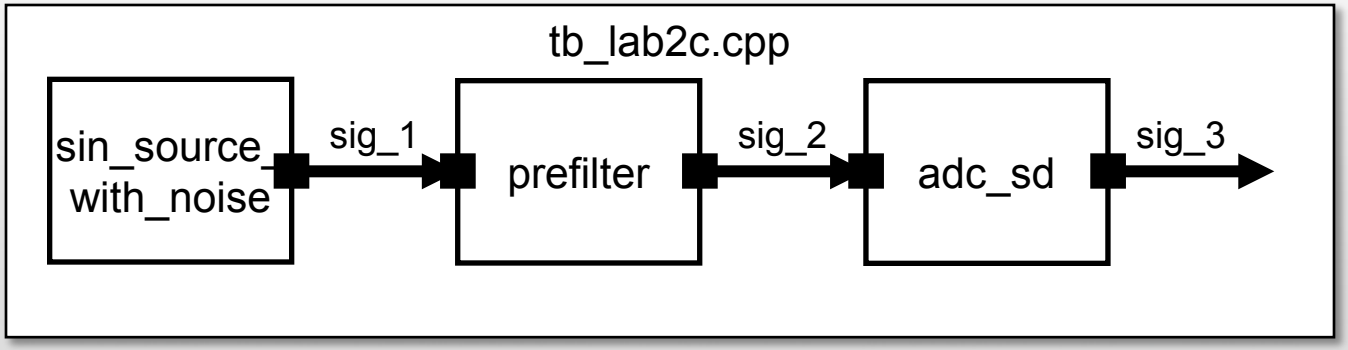
Lab 2c: Third testbench



TODO 3.4

In the **adc_sd.cpp** file, in the **processing()** function, complete/resolve the feedback loops with the respective assignments of the **integ1** and **integ2** member variables with the **s4** and **s2** values that have just been computed.

Lab 2c: Third testbench

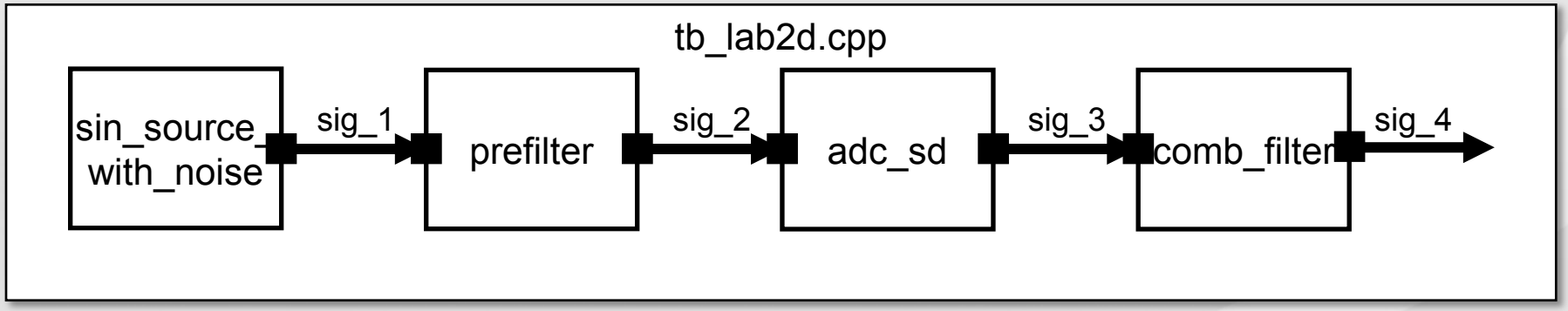


TODO 3.5

In the **tb_lab2c.cpp** test bench file, perform a 10 ms time-domain simulation and trace the signal **sig_3** coming from the **adc_sd**.

Lab 2d: Fourth testbench

- **Model and instantiate the comb filter**



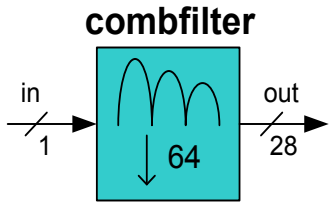
TODO 4.1

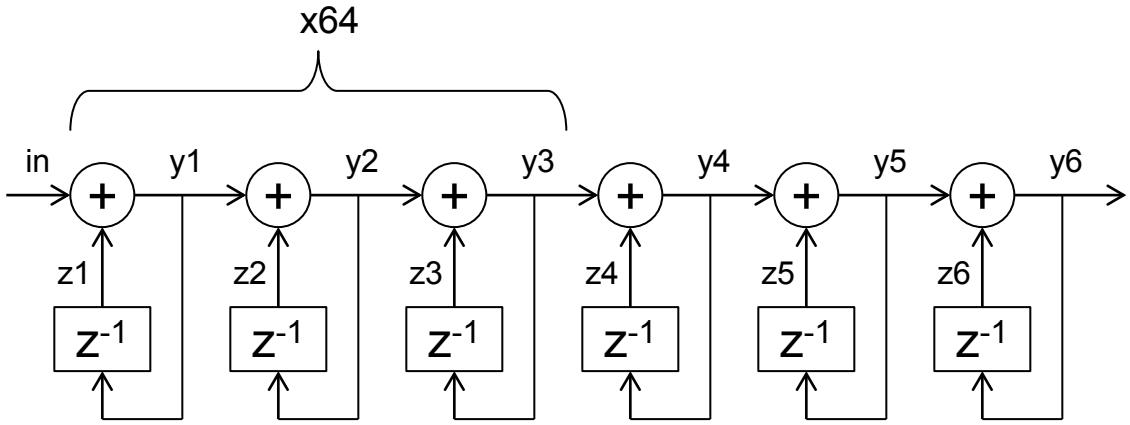
Examine the **comb_filter.h** file, and determine the size (in bits) of the data carried by the output port **out**.

Comb-filter principle

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

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



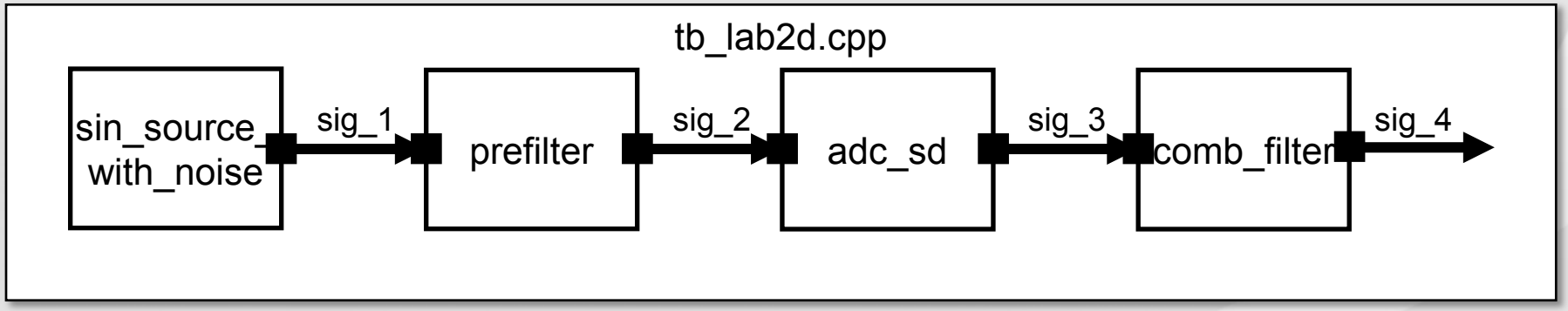


TODO 4.2

In the **comb_filter.h** file, declare 6 member variables z1 to z6 (19-bit long integers).

Lab 2d: Fourth testbench

- **Model and instantiate the comb filter**

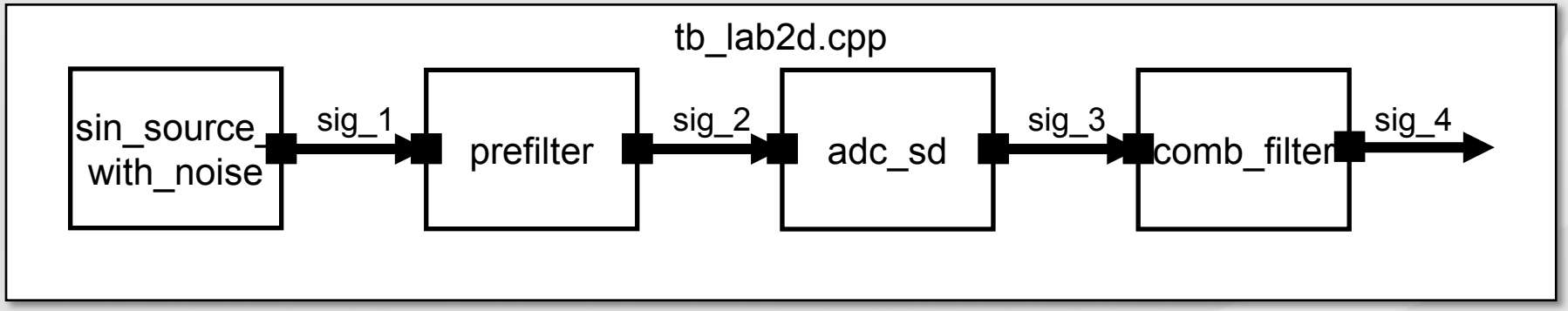


TODO 4.3

In the **comb_filter.h** file, complete the **set_attributes()** function so that the rate of the **in** input is 64 and keep the rate of the **out** output to 1. This way, **comb_filter** consumes 64 upstream samples to produce one downstream sample.

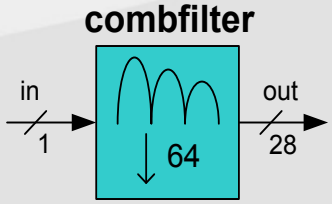
Lab 2d: Fourth testbench

- **Model and instantiate the comb filter**



TODO 4.4

In the **comb_filter.cpp** file, using the comb filter principle figure, complete the **processing()** function to perform comb filtering.

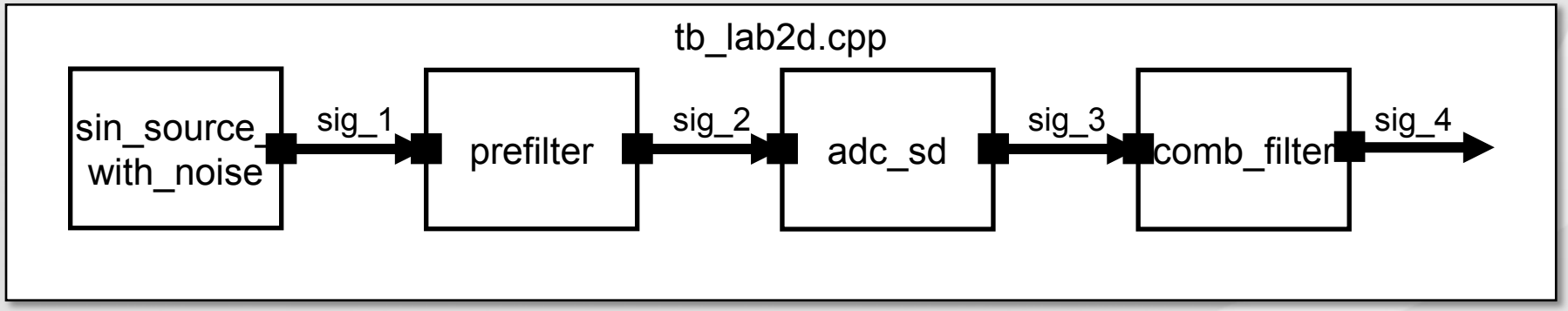


$$H(z) = \left(\frac{1 - z^{-k}}{1 - z^{-1}} \right)^{n}$$

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

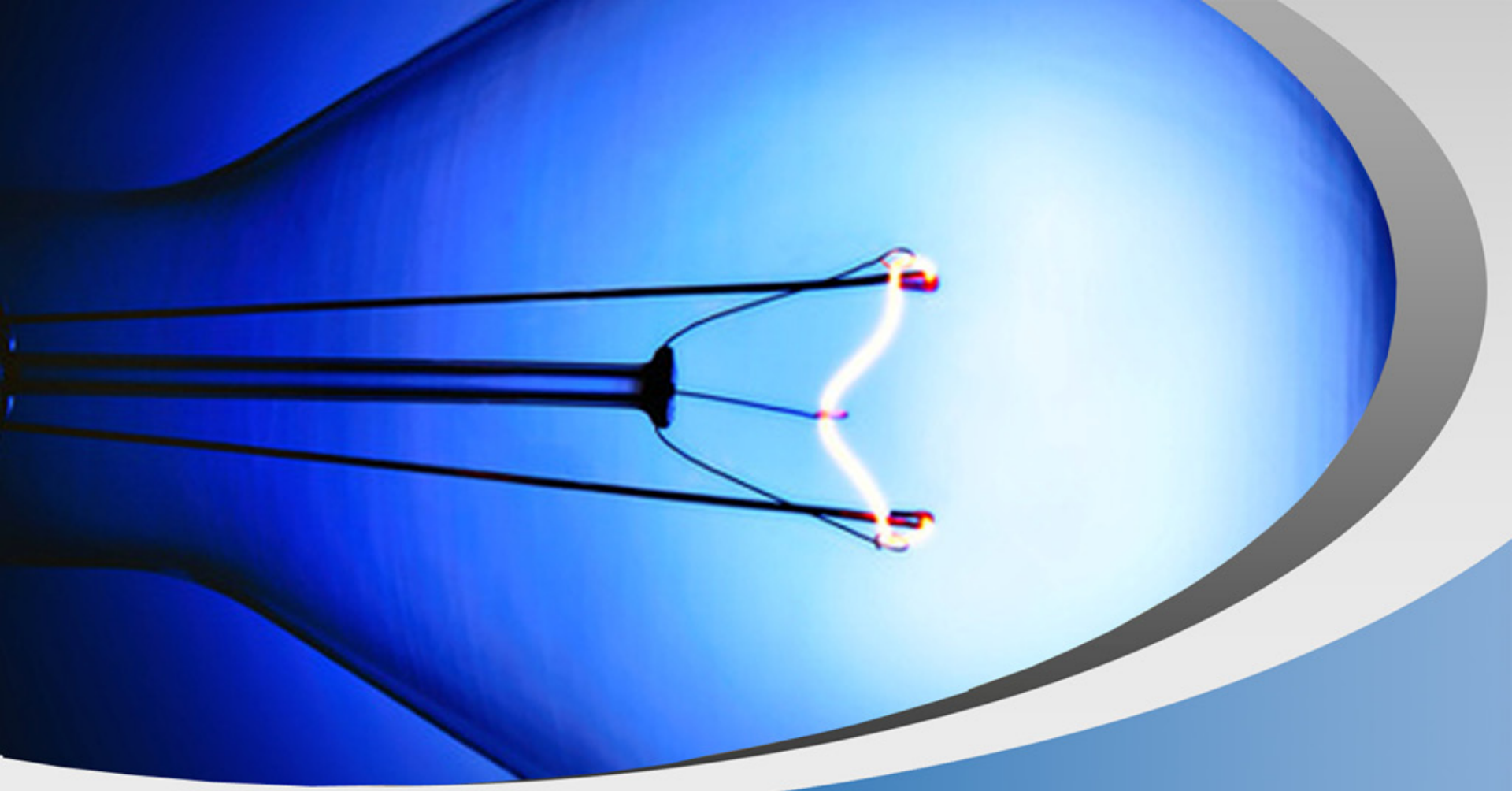
Lab 2d: Fourth testbench

- **Model and instantiate the comb filter**



TODO 4.5

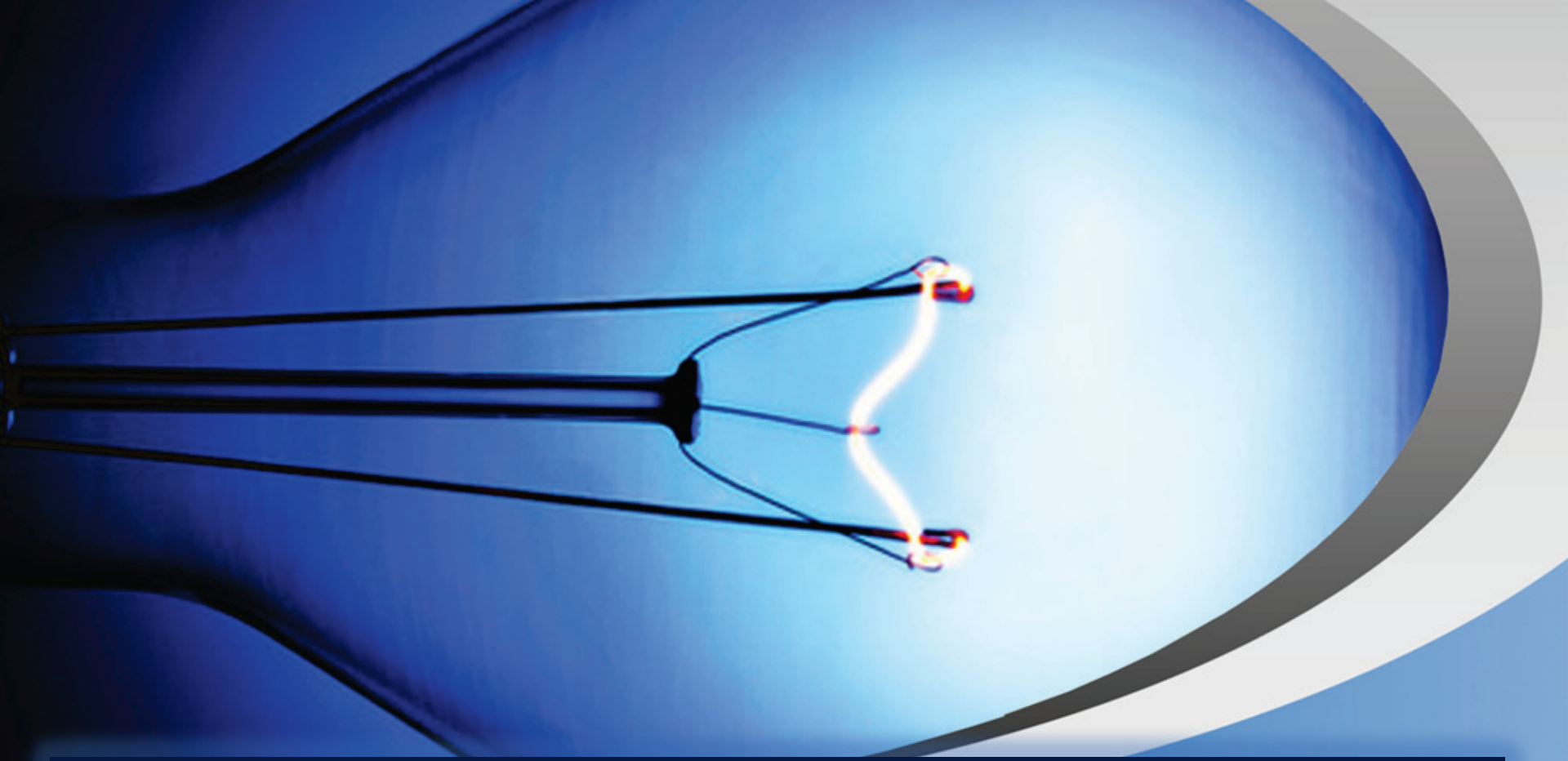
Perform a 10 ms time-domain simulation, and trace signal **sig_4**.



Thank you

Continue with Session 3: SystemC AMS 2.0 and Applications

accellera
SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Session 3: SystemC AMS 2.0 and Applications

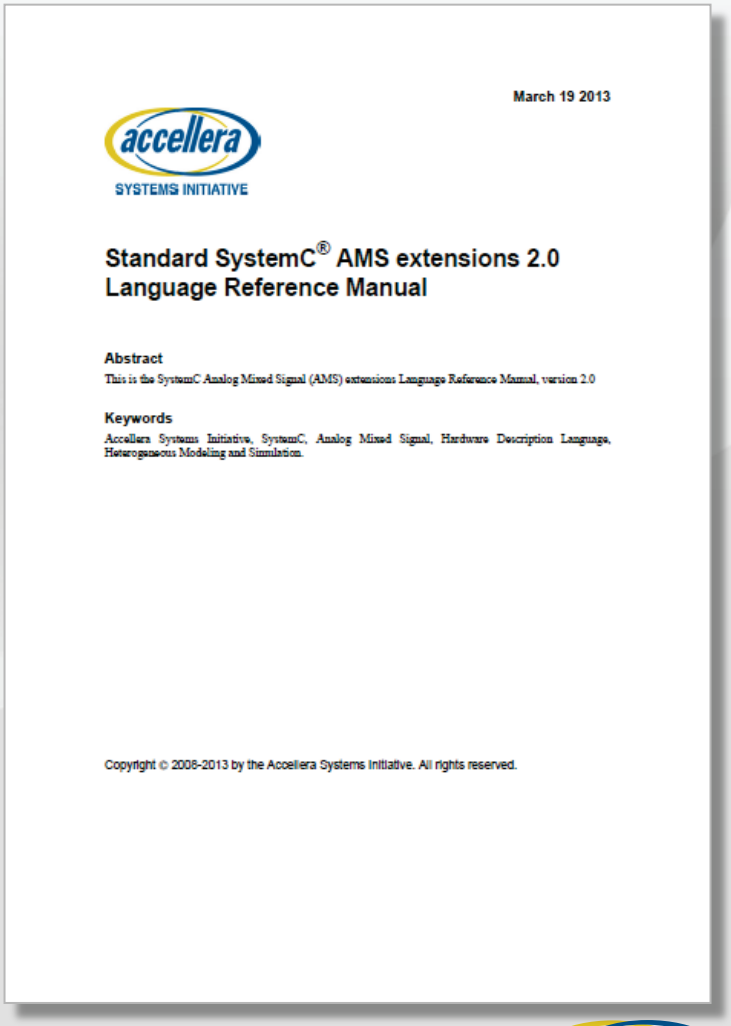
Torsten Mähne, Université Pierre et Marie Curie

accellera
SYSTEMS INITIATIVE

SystemC AMS 2.0 Overview

SystemC AMS 2.0 overview

- Dynamic time steps
- Changing time steps from TDF modules
- React on events – trigger calculation on an event
- Communicate between TDF modules belonging to different clusters with different non-multiple time steps
- Dynamic change of rate and delay attributes
- Repetition of time steps



March 19 2013

accellera SYSTEMS INITIATIVE

Standard SystemC® AMS extensions 2.0
Language Reference Manual

Abstract

This is the SystemC Analog Mixed Signal (AMS) extensions Language Reference Manual, version 2.0

Keywords

Accellera Systems Initiative, SystemC, Analog Mixed Signal, Hardware Description Language, Heterogeneous Modeling and Simulation.

Use cases and requirements

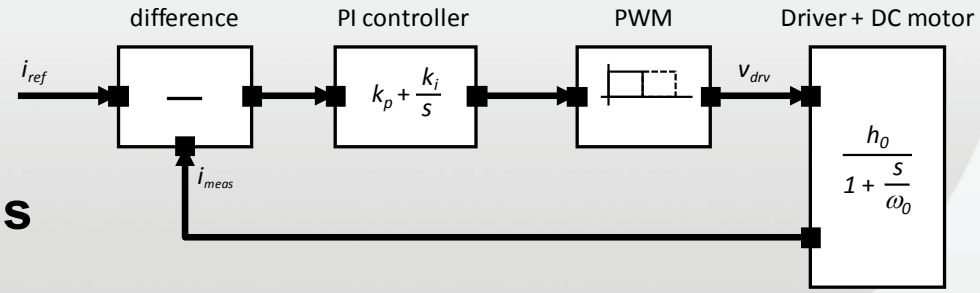
- **Abstract modeling of sporadically changing signals**
  - E.g., power management that switches on/off AMS subsystems
- **Abstract description of reactive behaviour**
  - AMS computations driven by events or transactions
- **Capture behavior where frequencies (and time steps) change dynamically**
  - Often the case for clock recovery circuits or for capturing jitter
- **Modeling systems with varying (data) rates**
  - E.g., multi-standard/software-defined radio (SDR) systems

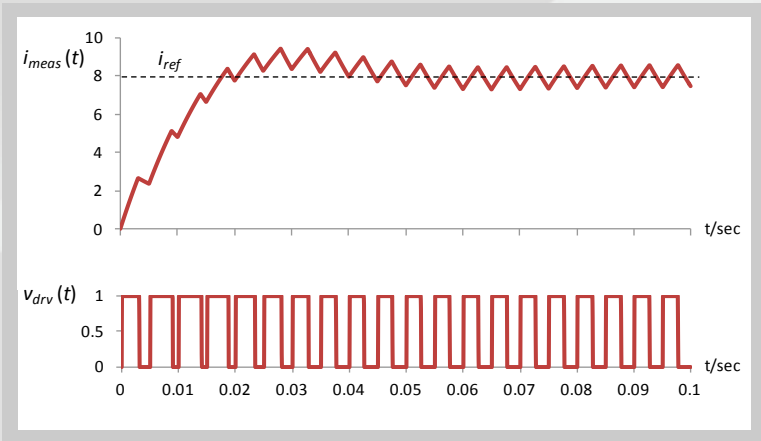
This requires a dynamic and reactive TDF modeling style

- **Basically introduce variable time step instead of fixed/constant time step**

Applications for dynamic TDF

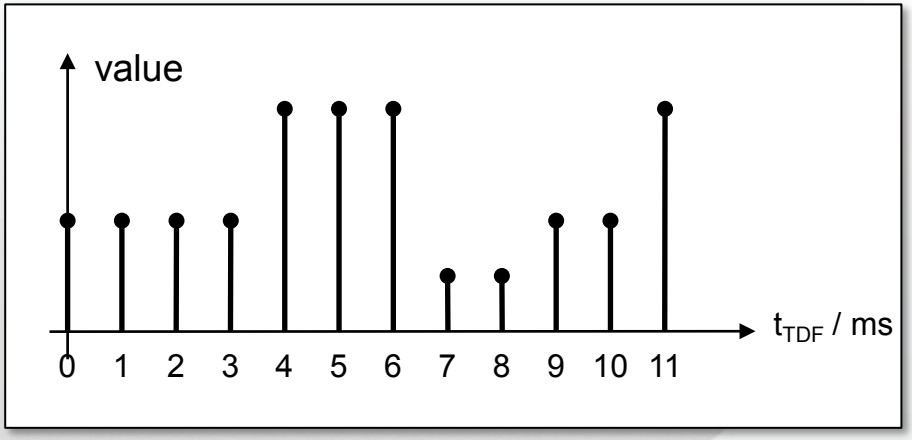
- **Automotive applications**
- **PWM stages**
- **Reaction to threshold crossings**
  - e.g., for modeling PLL's
- **Dataflow (DSP) algorithms**
  - Including clock recovery
- **Adaptive systems**
  - Systems supporting adjustable data rates or different/configurable algorithms
- **Non-linear dynamic systems**
  - Introducing macro modeling of, e.g., diodes, transistors, etc. in such systems





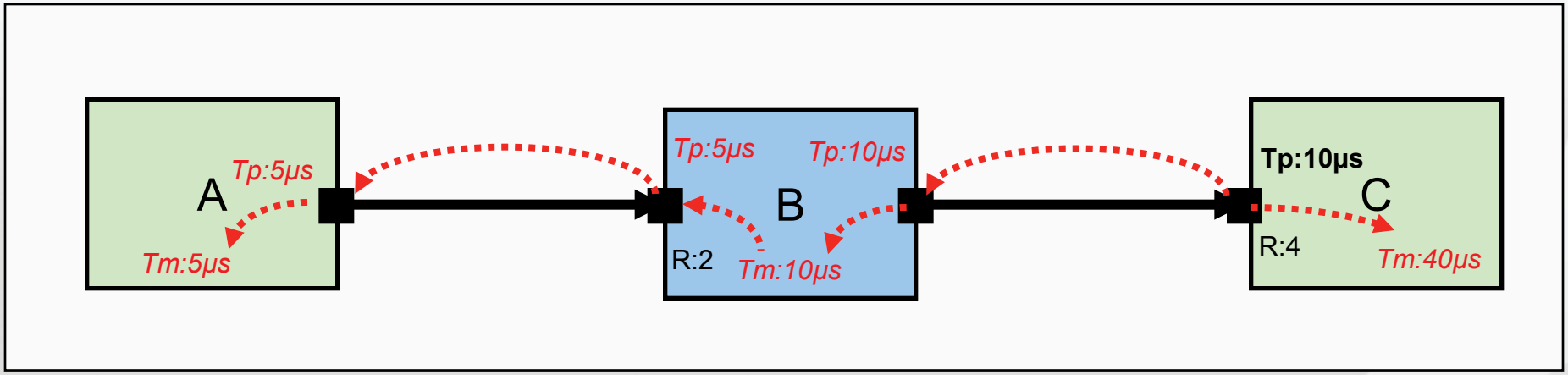
accellera SYSTEMS INITIATIVE

Timed Data Flow (TDF)



- Data Flow is an untimed MoC
- Timed Data Flow tags each sample and each module execution with an absolute time point
- Therefore, the time distance (time step) between two sample/two executions is assumed as constant
- This time distance has to be specified
- Enables synchronization with time-driven MoCs like SystemC discrete event and embedding of time-dependent functions like a continuous-time transfer function

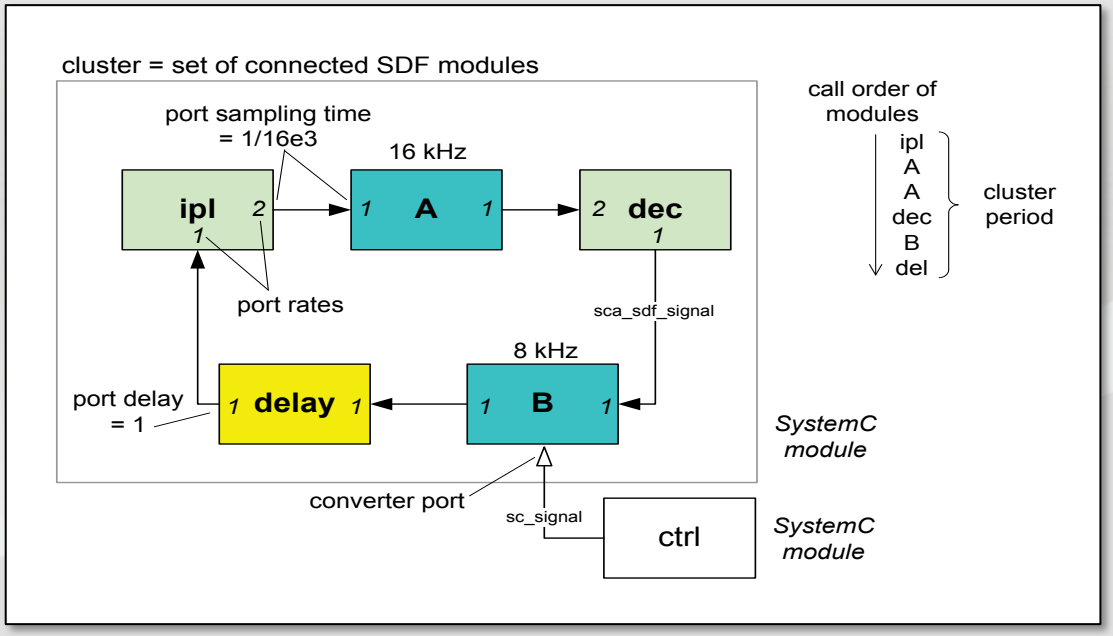
TDF – Time step propagation



- **If more than one time step is assigned, their consistency will be checked.**

Problems defining dynamic time steps

- **Each change requested by one module of the cluster influences the time step of all modules inside the cluster.**



- **Changes can only become valid after the whole cluster has been calculated.**
- **Attention: Run-time inconsistencies may occur!**

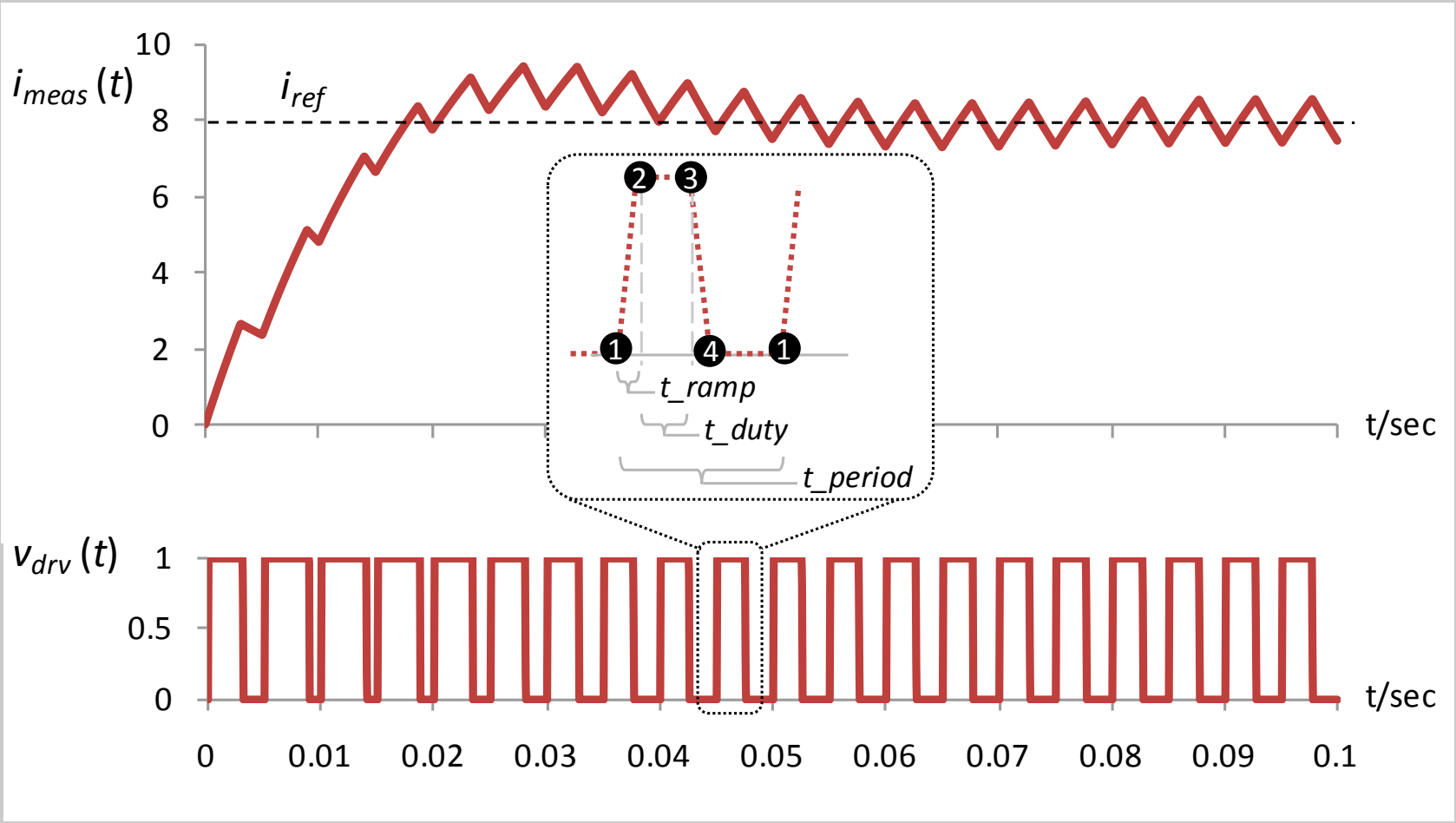
TDF dynamic mode of operation

- The TDF MoC now supports the *static* (fixed time step) and *dynamic* (variable time step) mode of operation.
- All modules in a cluster must support the dynamic mode of operation if one module requests dynamic features.
- Attributes (e.g., the time step) can only be changed before the cluster execution starts.
- An additional optional TDF module callback *change_attributes* is executed before the cluster starts. Only in the context of this callback, attributes can be changed.
- TDF modules can be marked to support the dynamic mode of operation by calling the method *accept_attribute_changes* or *does_attribute_changes* in the context of the constructor or *set_attributes.* This marking of the modules may be dynamically updated in the *change_attributes* callback.

New SystemC AMS 2.0 methods

- **New callback and member functions to support the TDF dynamic mode of operation:**
  - **change_attributes() / reinitialize()**
    - Callbacks provide a context, in which the time step, rate, or delay attributes of a TDF cluster may be changed and the delay samples may be reinitialized, respectively.
  - **request_next_activation(…)**
    - Member function to request a next cluster activation after a given time step, event, or event list.
  - **does_attribute_changes(), does_no_attribute_changes()**
    - Member functions to mark a TDF module to allow or disallow making attribute changes itself, respectively.
  - **accept_attribute_changes(), reject_attribute_changes()**
    - Member functions to mark a TDF module to accept or reject attribute changes caused by other TDF modules, respectively.

Application: DC motor control



Example of Pulse Width Modulator

Dynamic TDF features indicated in red

```
#include <systemc-ams>

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

  void set_attributes()
  {
    out.set_timestep(0.05, sc_core::SC_MS);
    does_attribute_changes();
    accept_attribute_changes();
  }

  void change_attributes()
  {
    out.set_timestep(duration, sc_core::SC_MS);
  }

  void processing()
  {
    double in_lim = in.read();
    ...
    if (in_lim > 255.0) in_lim = 255.0;
    if (in_lim < 0.0)   in_lim = 0.0;

    if (act1)
    {
      out.write(1.0);
      duration = in_lim;
      act1 = false;
    } else
    {
      out.write(0.0);
      duration = 255.0 - in_lim;
      act1 = true;
    }
  }

  SCA_CTOR(pwm_dtdf) : act1(true), duration(0.0) {}

 private:
  bool act1;
  double duration;
};
```

TDF vs. dynamic TDF comparison

TDF model of computation variant	*t_step* (ms)	*t_ramp* (ms)	*t_period* (ms)	Time accuracy (ms)	#activations per period
Conventional TDF	0.01 (fixed)	0.05	5.0	0.01 (= *t_step*)	500
Dynamic TDF	variable	0.05	5.0	defined by `sc_set_time_resolution()`	4

- **Comparison of the two variants of the TDF model of computation**
  - Conventional PWM TDF model uses a fixed time step that triggers too many unnecessary computations.
  - When using Dynamic TDF, the PWM model is only activated if necessary.

Methods for reactivity and dynamic time steps

```
set_timestep(...), set_rate(...),
  set_delay(...)



void request_next_activation(const sc_event&);
void request_next_activation
              ( const sca_core::sca_time& );
            :


void set_max_timestep
            ( const sca_core::sca_time& );
```

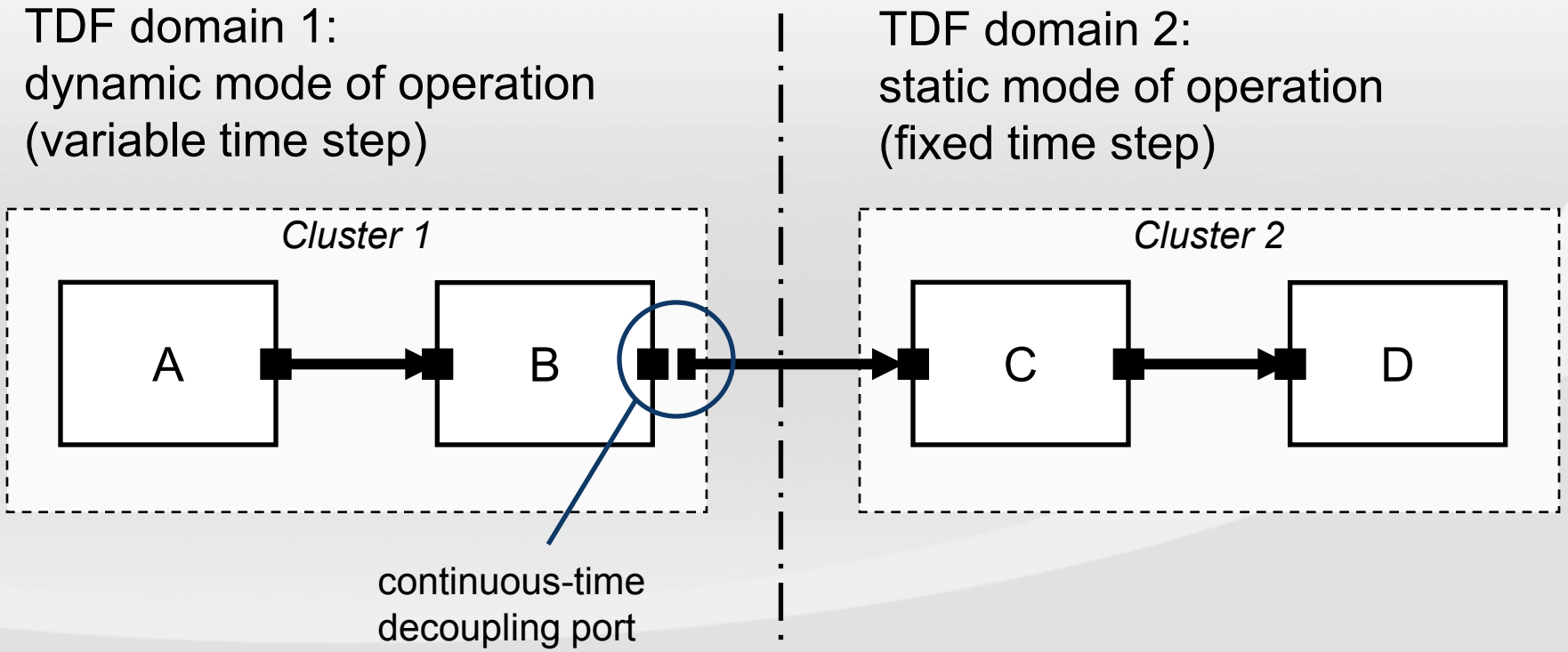
Same methods like for **set_attributes** – consistency will be checked.

Requests an additional activation, same argument combinations like SystemC **next_trigger**.

Limits the next time step

Note: *Zero time steps are allowed – analog solver will invalidate the last time step and re-calculate.*

Decoupling TDF clusters 1/2



Decoupling TDF clusters 2/2

- **Connecting static and dynamic TDF modules with non-multiple time steps**
- **Connecting static and dynamic TDF clusters**
- **Problem: Analog continuous-time semantic requires interpolation**
  - Interpolation requires the next resolution point.
- **We will use two special kinds of output ports for decoupling:**
  - One output port, which is able to interpolate. It must have at least one sample time step delay. A user-defined interpolation method may be provided.
  - One output port, which holds the value. It uses the evaluate-update semantic (if the read and write time is equal, the old value will be read).

Decoupling port example

```
SCA_TDF_MODULE(decoup)
{
  // The signal (sca_tdf::sca_signal) connected to
  // this port will be decoupled from the cluster.
  sca_tdf::sca_out<double, SCA_CUT_CT> out;

  void set_attributes()
  {
    // This delay will limit the maximum
    // possible time step.
    out.set_ct_delay(0.05, sc_core::SC_MS);
  }

  ...

};
```

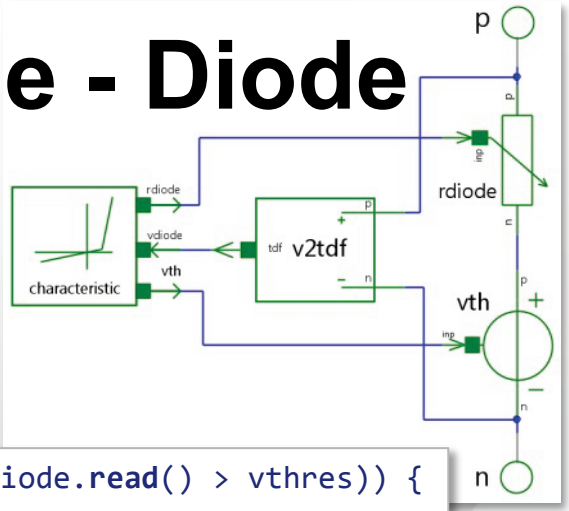
- **The decoupling port uses a 2nd template argument**
  - **SCA_CUT_CT**: continuous-time decoupling
  - **SCA_CUT_DT**: discrete-time decoupling (sample-and-hold)
- **New member function to define the continuous-time delay at the output port**
  - Essential for continuous time decoupling ports to be able to perform interpolation.

Repetition of time steps

- LSF and ELN networks derive the time step from the connected TDF cluster.
- When using the dynamic SystemC-AMS 2.0 feature (e.g., with request_next_activation), zero time steps are allowed.
- A zero time step for a linear equation solver (the ELN, LSF or the embedded sca_ltf_nd / zp, sca_ss) repeats the last time step with the new input values (resets to the old state and recomputes the time step).
- This enables solver iterations and modeling of non-linear dynamic behavior.

Non-linear modeling example - Diode

```
SCA_TDF_MODULE(characteristic)
{
  sca_tdf::sca_in<double>  vdiode;
  sca_tdf::sca_out<double> rdiode, vth;

  void set_attributes() {
    rdiode.set_delay(1);
    vth.set_delay(1);
    does_attribute_changes();
  }

  void initialize() {
    rdiode.initialize(roff);
    vth.initialize(0.0);
    recalculate = ron_state = false;
  }

  void change_attributes() {
    if(recalculate)
      request_next_activation(SC_ZERO_TIME);
  }

  void processing() {
    recalculate = false;

    if(ron_state && (vdiode.read() < vthres))
    {
      recalculate = true; ron_state = false;
    }
    if(!ron_state && (vdiode.read() > vthres)) {
      recalculate = true; ron_state = true;
    }

    if(ron_state) {
      rdiode.write(ron); vth.write(vthres);
    } else {
      rout.write(roff); vth.write(0.0);
    }
  }

  double ron, roff, vthres;

  SCA_CTOR(characteristic) {
    ron = 1e-3; roff = 1e7; vthres = 0.7;
  }

private:
  bool ron_state, recalculate;
};
```



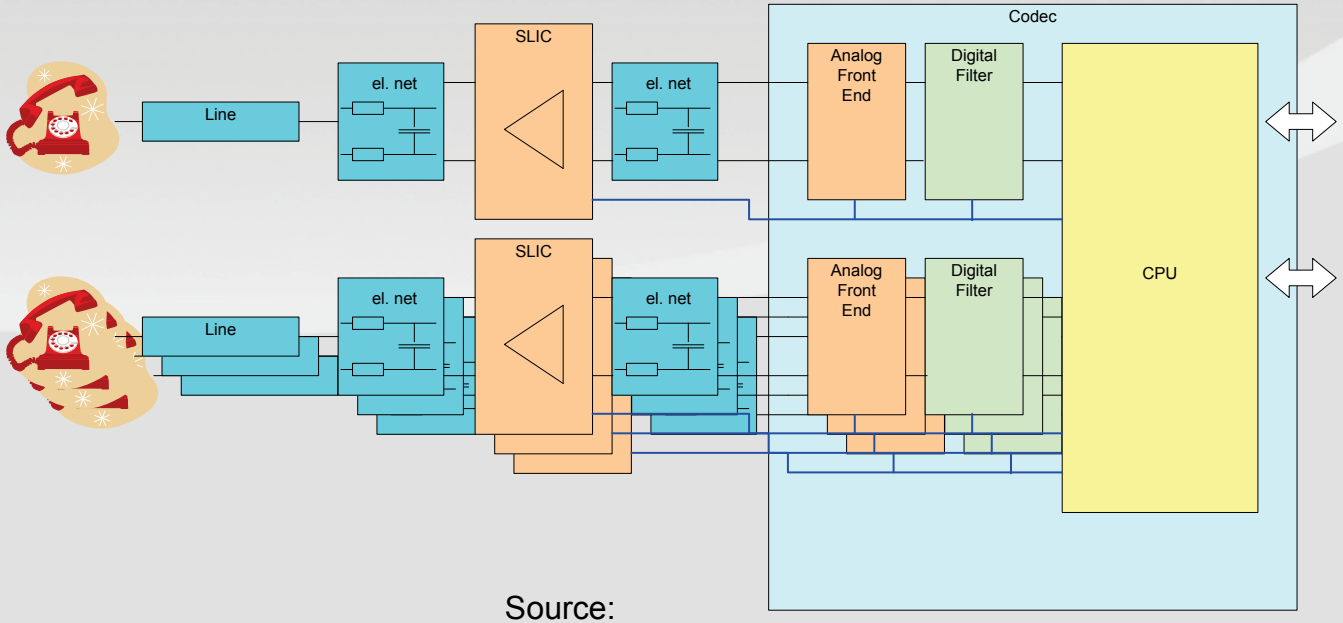
SystemC AMS Applications

Applications for SystemC AMS

- In-vehicle networks
- POTS and xDSL systems
- WLAN chipsets
- Imaging sensors
- Braking systems
- Airbag systems
- Powertrain
- Sensor circuits (magnetic, pressure, gyro, …)
- Smart Cards
- Driving systems
- Power supply

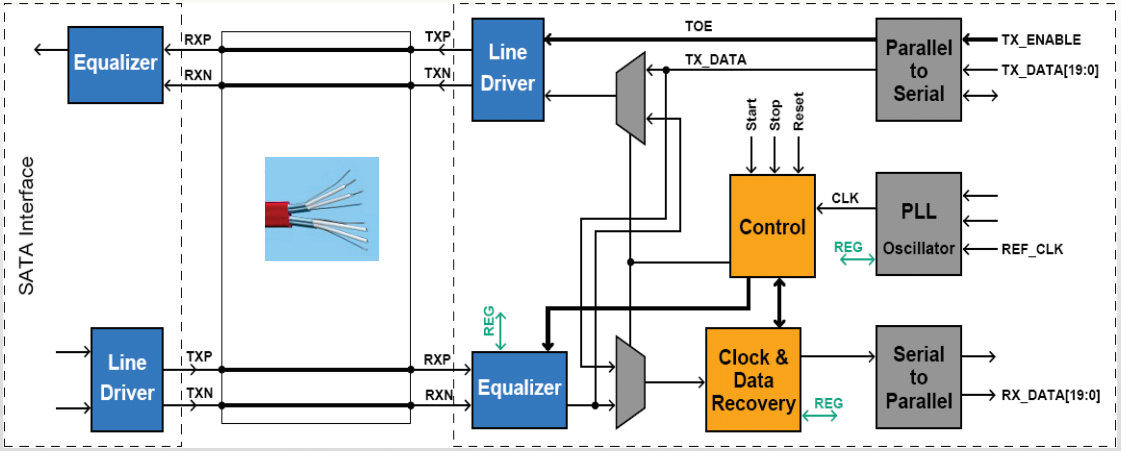
POTS System Design

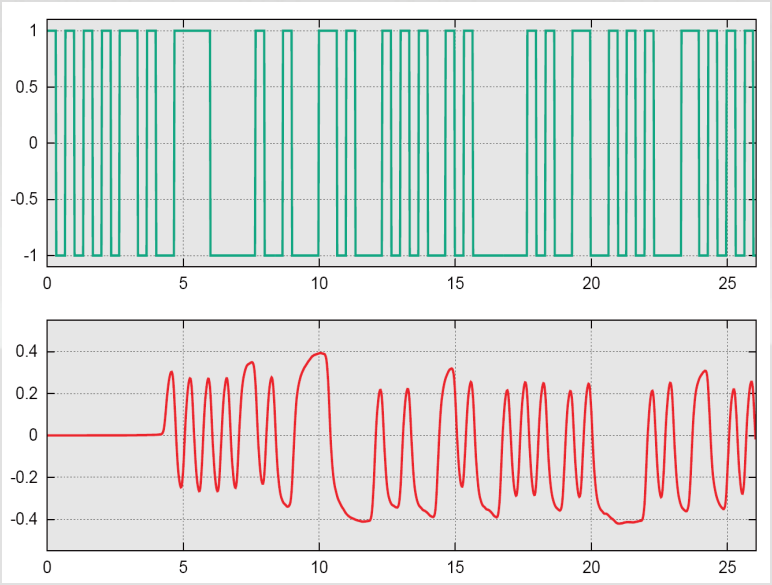
- Complete System functionality modeled
- All relevant analogue effects
- Digital parts “bit-true”, original code of embedded software
- Hundreds of simulation scenarios as regression tests available
- Simulation scenarios partially re-used for silicon verification
- Embedded software debugged before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



Source:
Gerhard Nössing, LANTIQ

Mixed-Signal Embedded Core - SATA



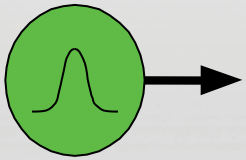


- **Serial ATA physical layer chip set for 3 / 6 / 8 GBit serial data transmission, e.g., to/from hard discs**
- **Concept engineering model of transmitter/receiver including the PLLs**
- **Goal: Estimation of bit error rates, simulation of PLL locking behavior**
- **Integration into a VHDL model into Modelsim as reference model and stimuli generator for the digital components**

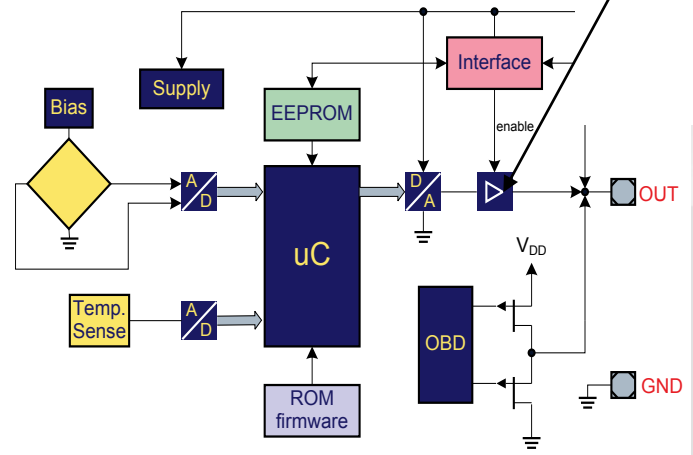
Source:
Infineon Technologies

Automotive Sensor Applications

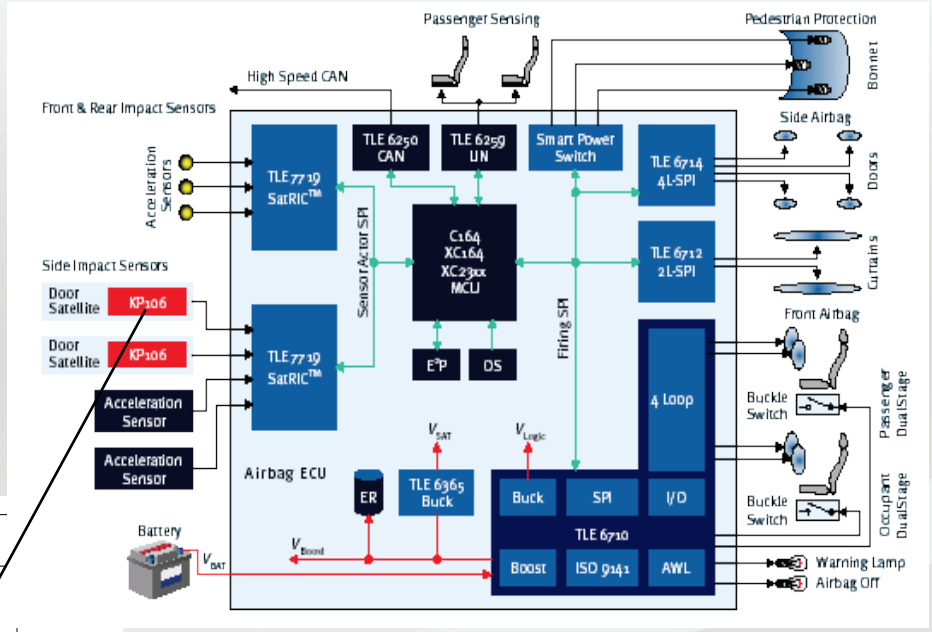
TIER2



pressure pulse



TIER1

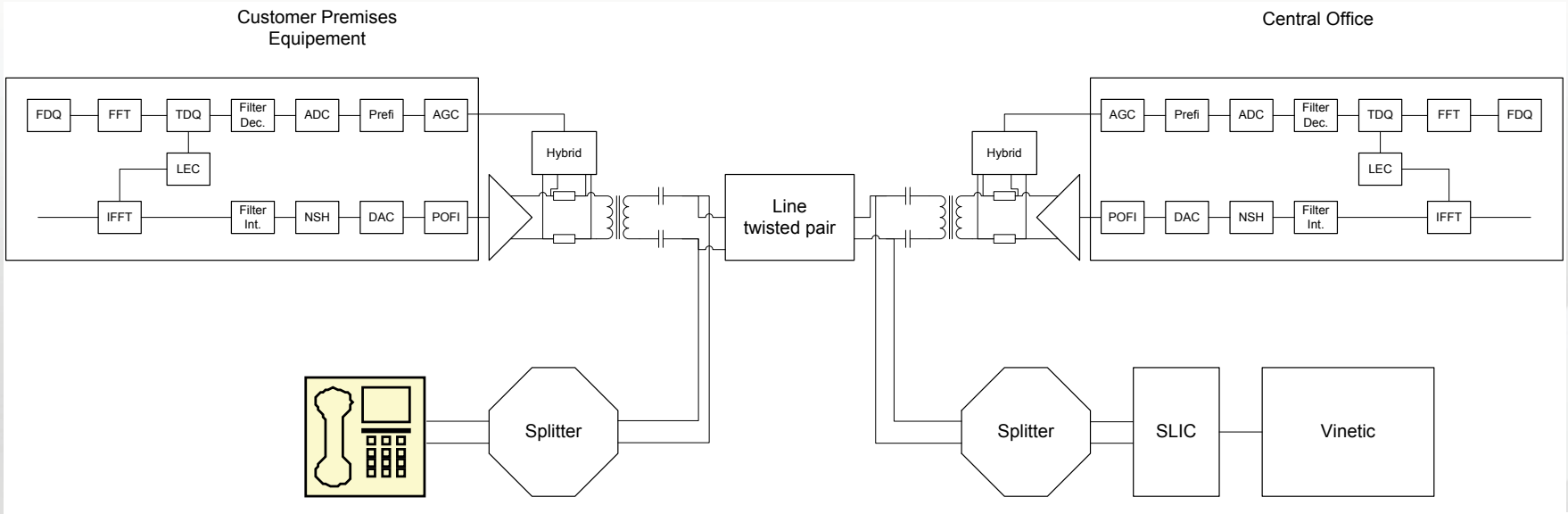


OEM

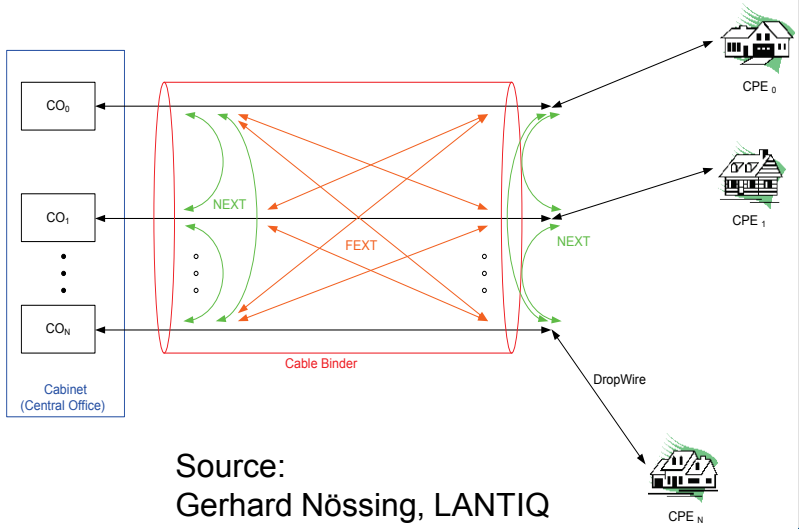
Source:
Wolfgang Scherr, Infineon

accellera SYSTEMS INITIATIVE

ADSL / VDSL Systems



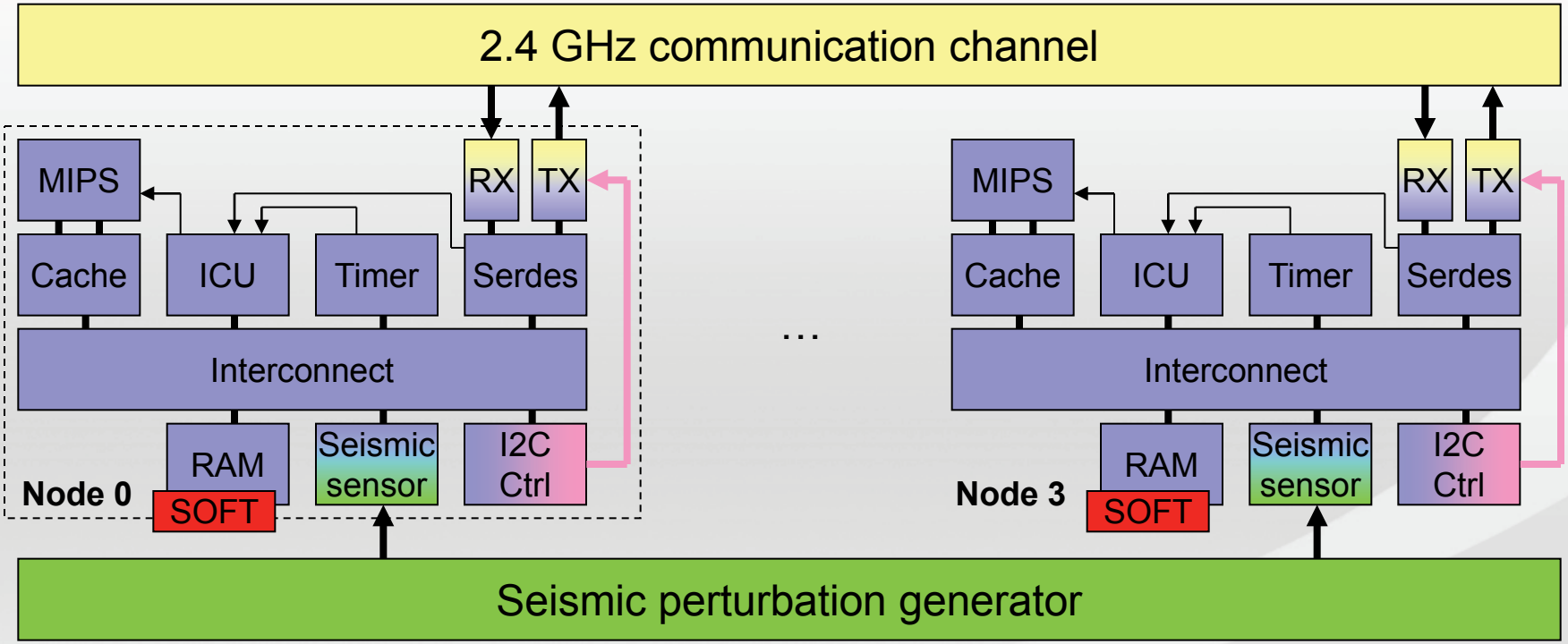
- **Transient settling behavior**
- **Interaction Voice / Data transmission**
- **Training algorithm**
- **BER estimations**
- **Numerous of use scenarios**
- **Interaction of different lines**
- **Multi-level simulation environment essential**



Source:
Gerhard Nössing, LANTIQ

accellera SYSTEMS INITIATIVE

Seismic Perturbation Wireless Sensor Network



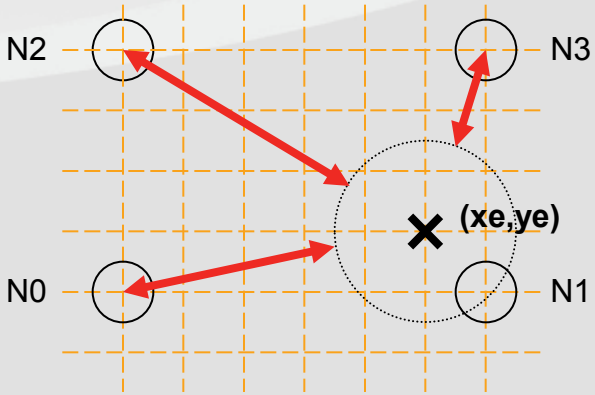
Digital, BCA, SocLib

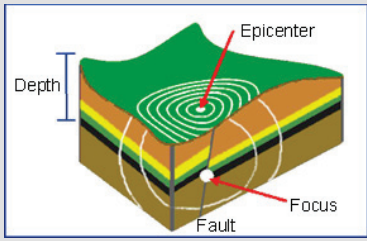
Analog, SystemC-AMS TDF, RF

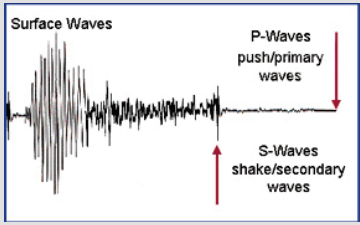
Analog, SystemC-AMS TDF, Physics, ΣΔ

Analog, SystemC-AMS ELN, Electrical, Bus

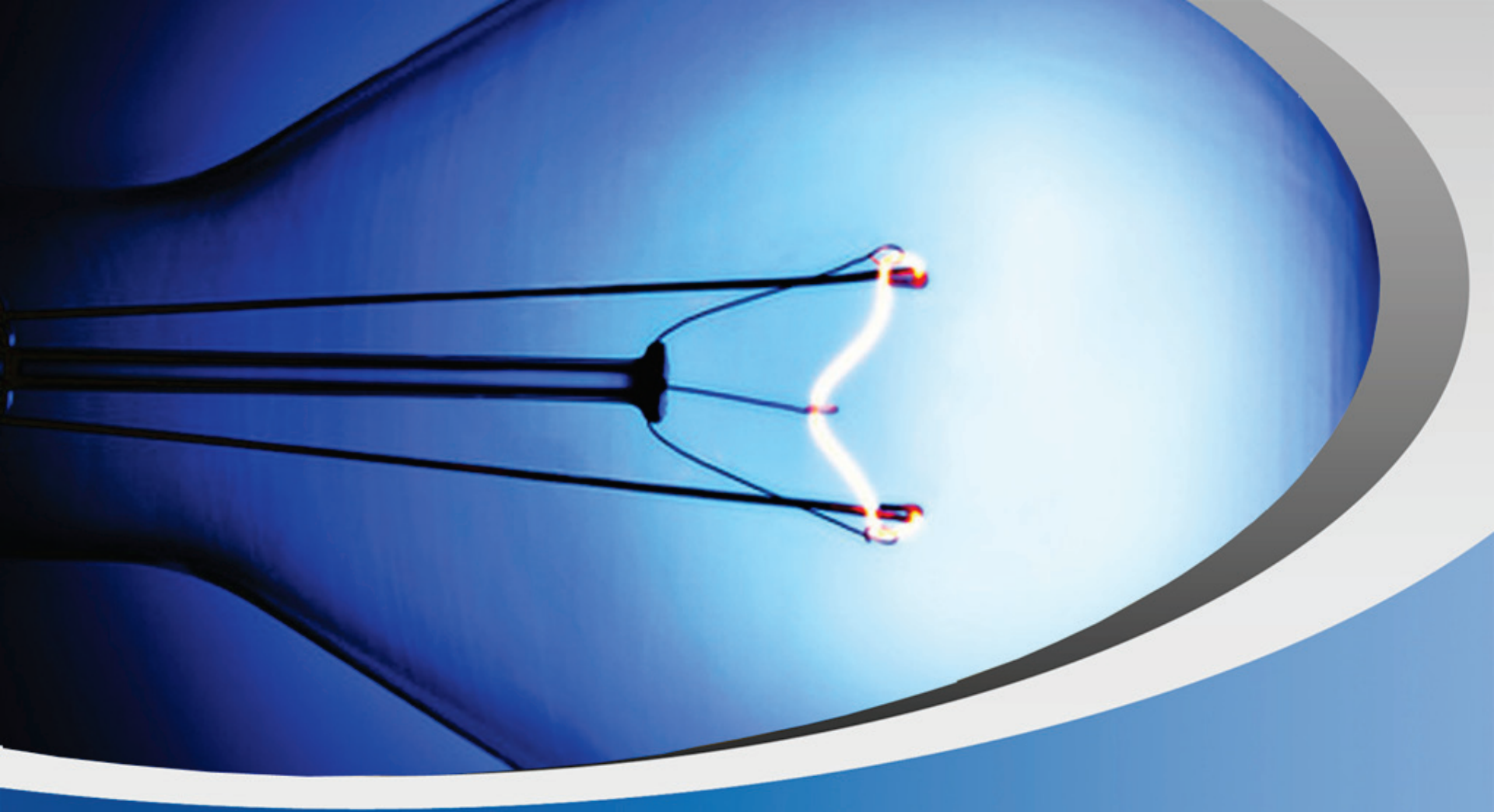
Embedded software







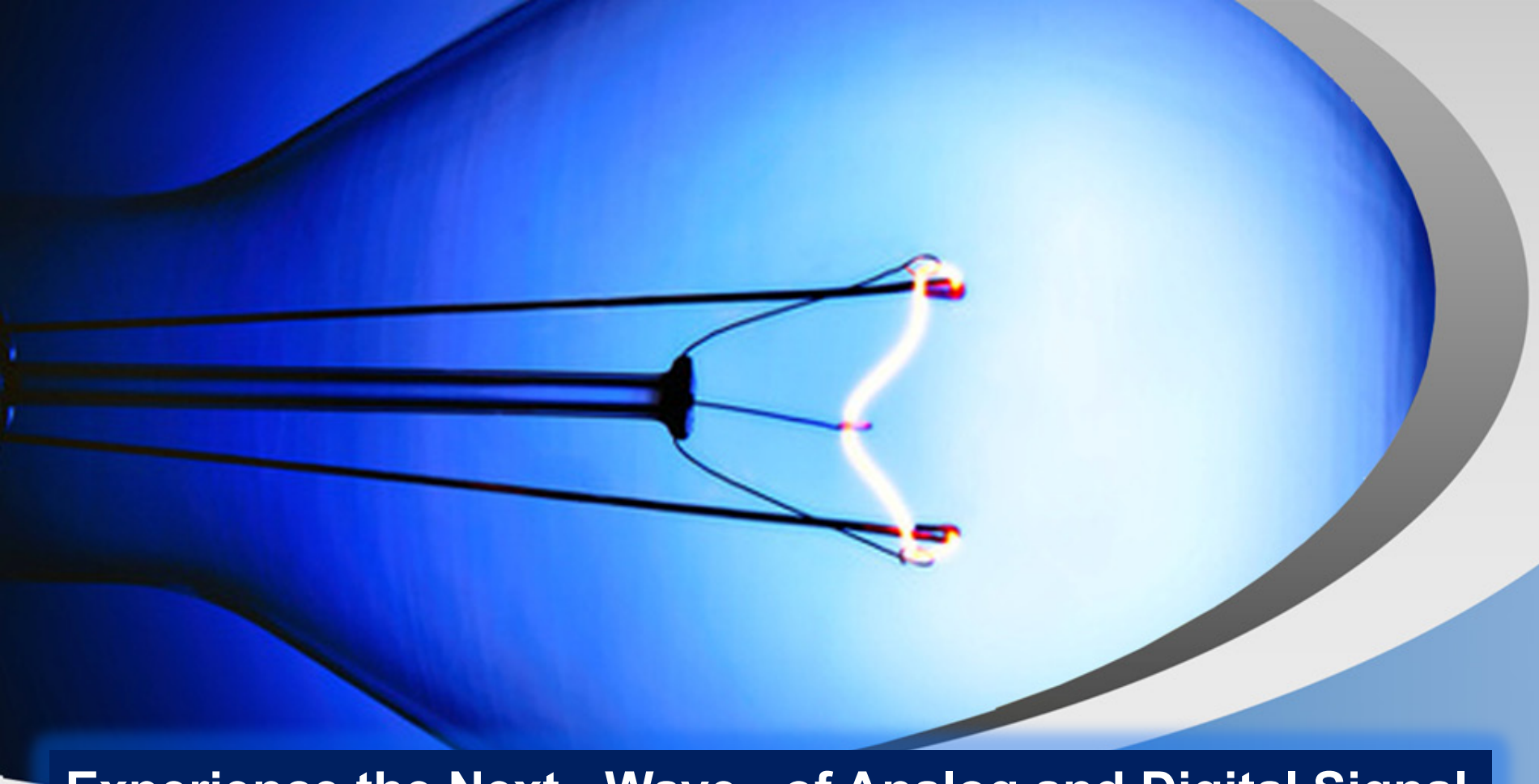
Source: Université Pierre et Marie Curie



Thank you

Continue with Lab 3: System Modeling — Vibration Sensor

accellera SYSTEMS INITIATIVE

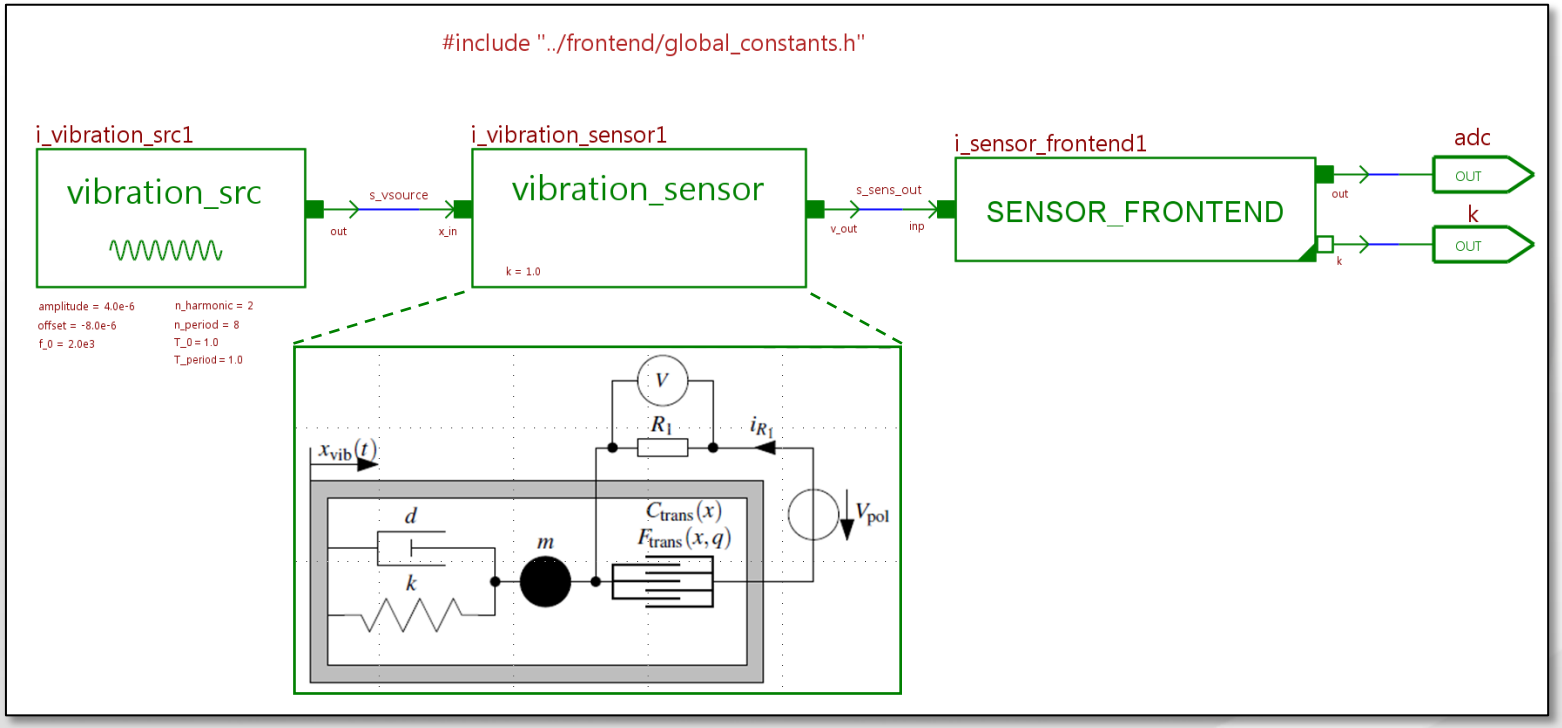


Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Lab 3: System Modeling:
Vibration Sensor with DSP Front End

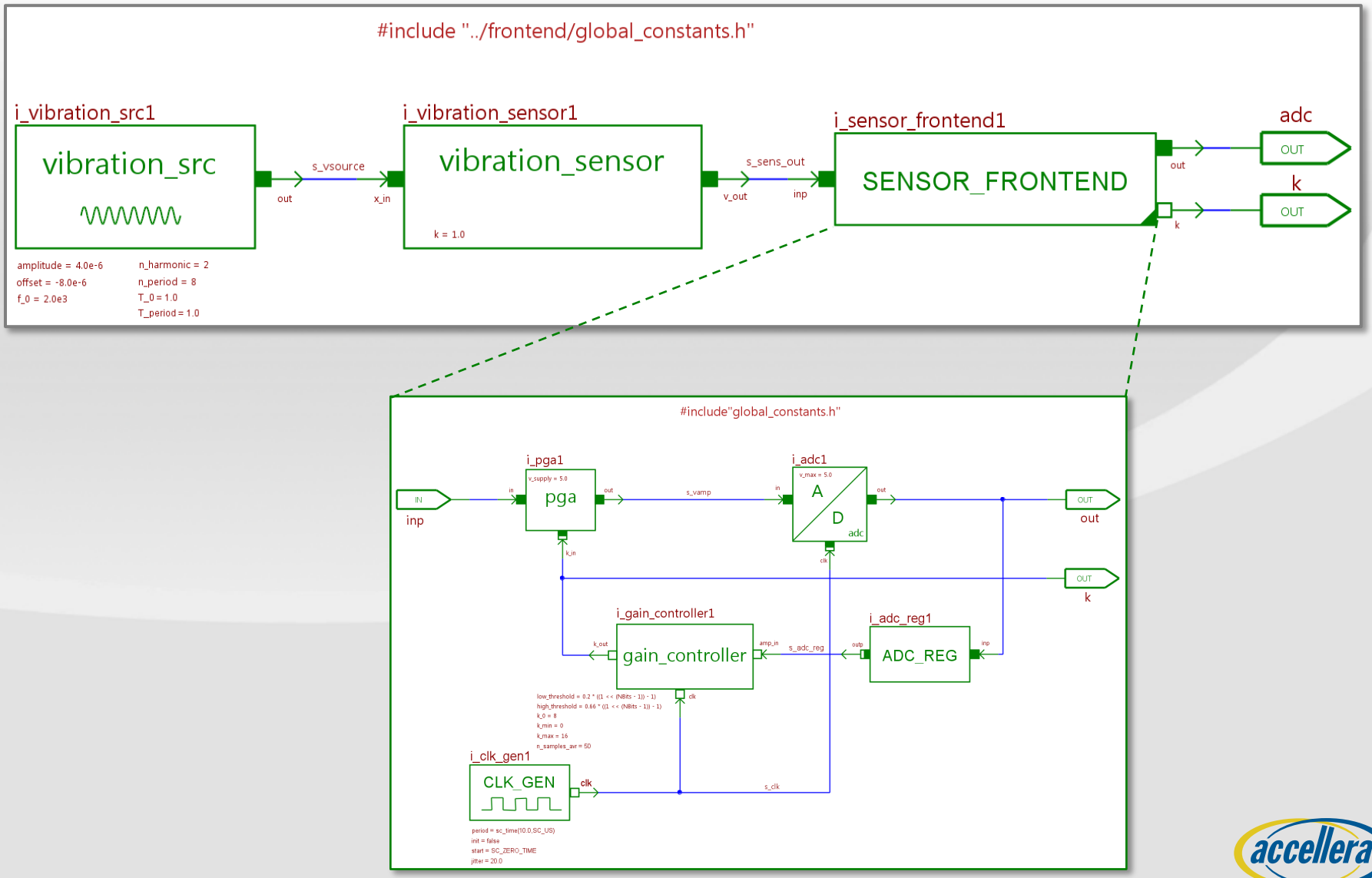
accellera
SYSTEMS INITIATIVE

Lab 3

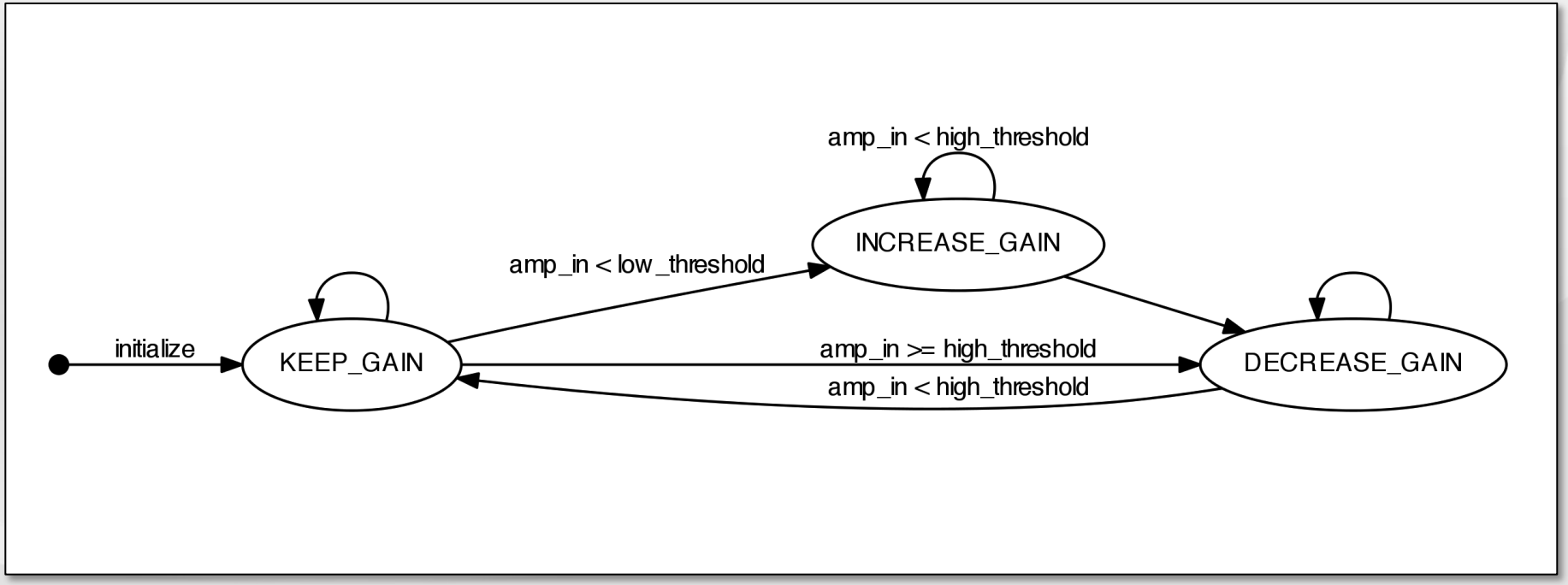


- **Goal: Model a vibration sensor with its DSP frontend:**
  - Digitizing the speed signal: Vibration Sensor → PGA → ADC
  - Feedback loop to control PGA for optimal use of ADC dynamic range: ADC → Gain Controller
- **Exercises:**
  - Evaluate the model sources
  - Carry out the simulation
  - Visualize the results in the waveform viewer

COSIDE models

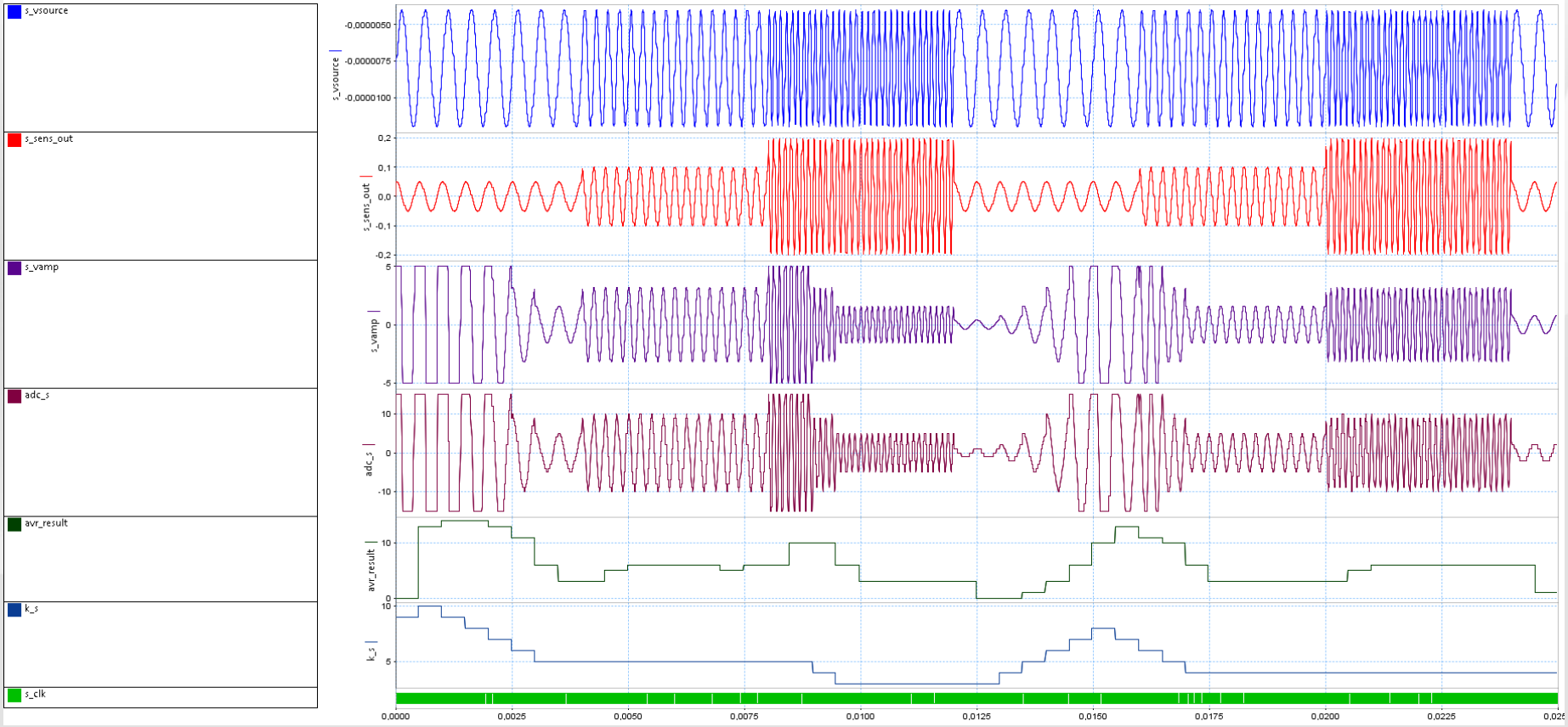


Gain controller: Finite State Machine



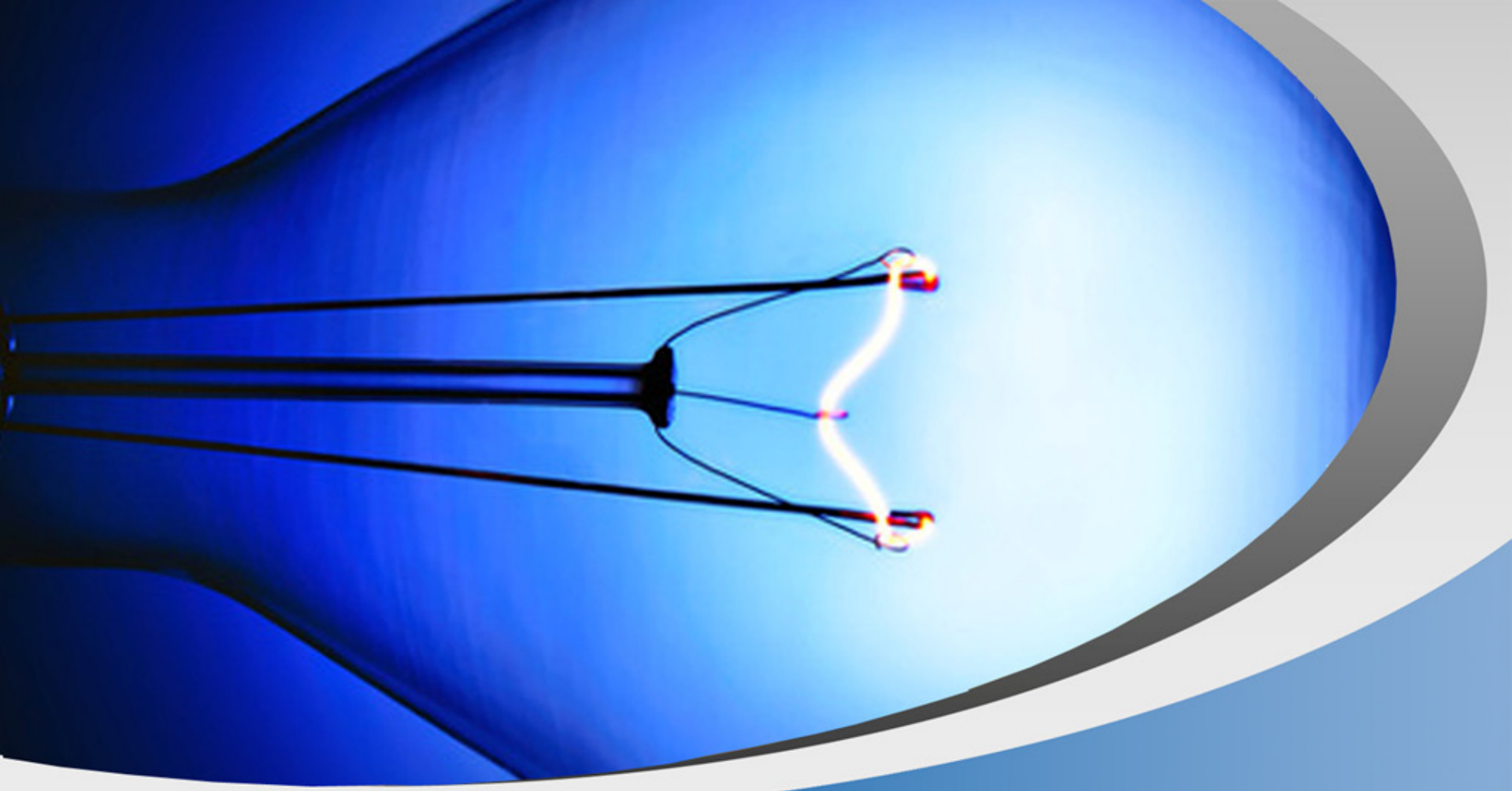
- **FSM implemented in SystemC discrete-event**

Simulation results - Waveforms



Conclusions

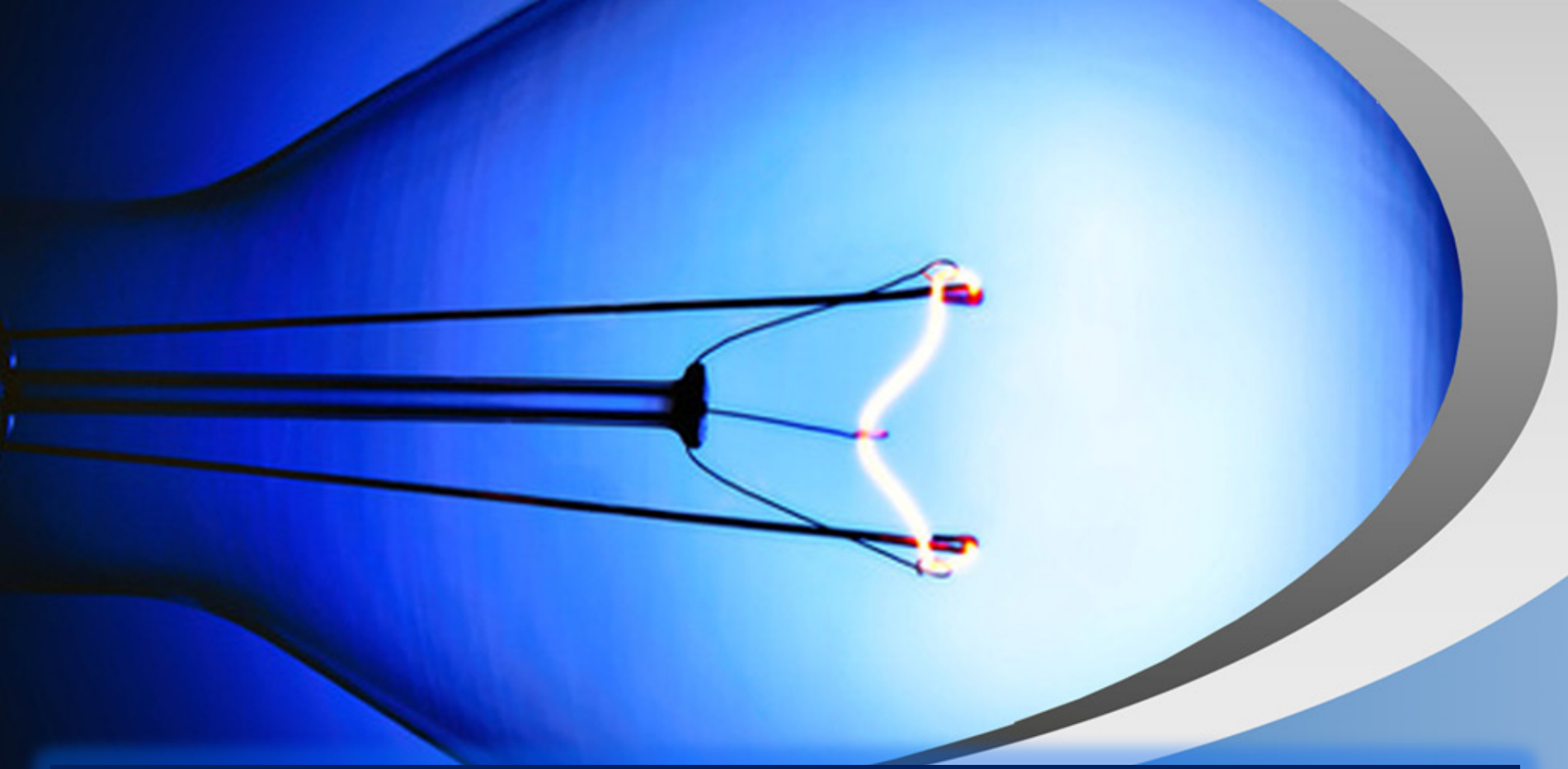
- **Modeled a vibration sensor with its DSP front end:**
  - Digitizing the speed signal: Vibration Sensor → PGA → ADC
  - Feedback loop to control PGA for optimal use of ADC dynamic range: ADC → Gain Controller
- **Synchronization of data flow computation based on clock signal**
  - Uses SystemC AMS 2.0 features (e.g., request_next_activation)



Thank you

Continue with Workshop Summary

accellera
SYSTEMS INITIATIVE



Experience the Next ~Wave~ of Analog and Digital Signal Processing using SystemC AMS 2.0

Workshop Summary

accellera
SYSTEMS INITIATIVE

Summary

- **SystemC provides a standardized modeling framework for the development of new ESL design methodologies**
- **Enabling co-design methodologies**
  - HW/SW co-design and co-verification
  - AMS + HW/SW co-design and co-verification
- **Supported design refinement methodologies**
  - TLM: loosely, approximately or cycle accurate modeling
  - AMS: mixed discrete- and continuous-time, non-conservative and conservative modeling
- **Virtual Prototyping using SystemC and SystemC AMS**
  - Supporting a wide variety of use cases, e.g., early SW development

Resources – SystemC AMS

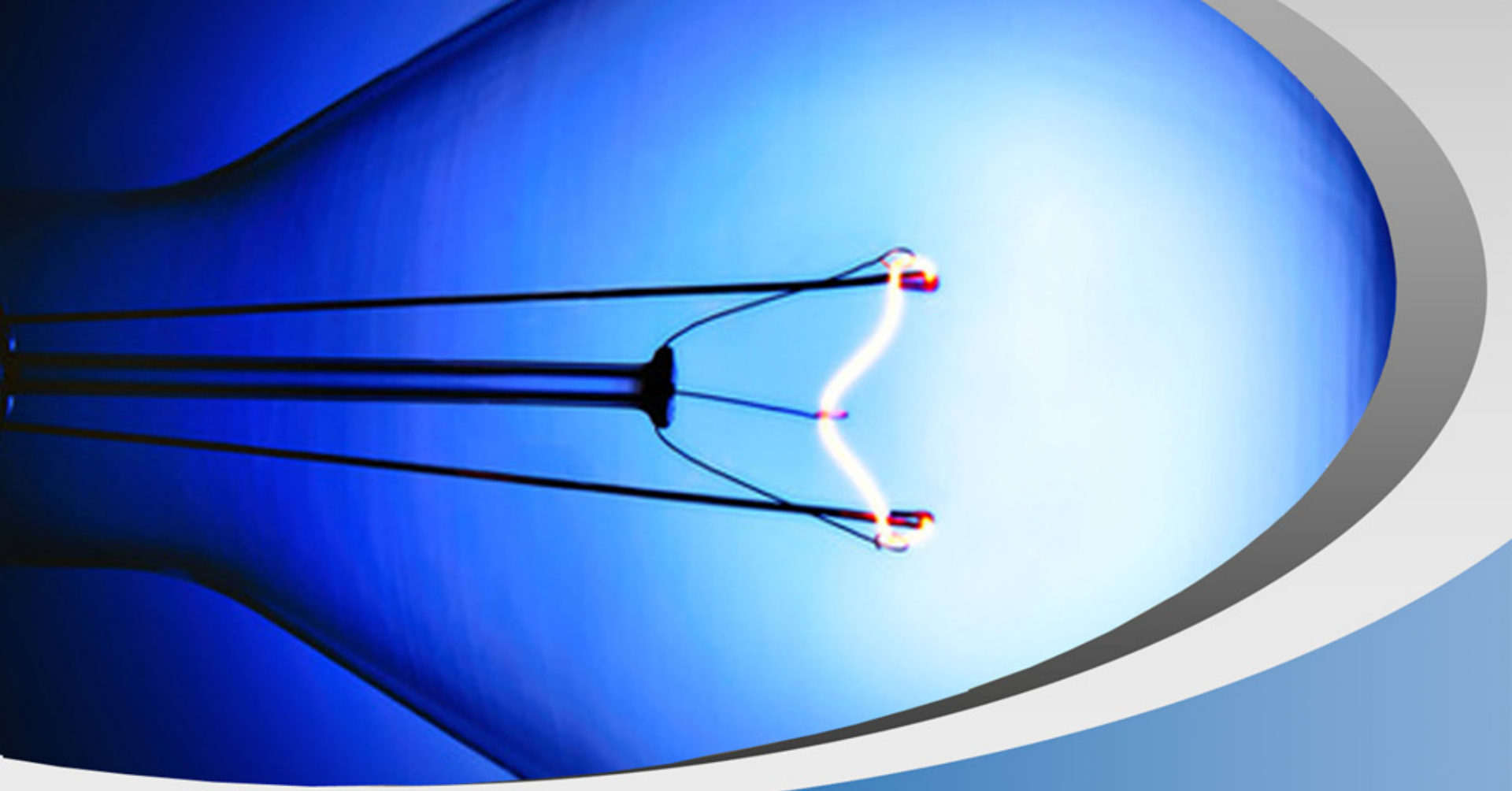
- **SystemC-AMS Homepage: http://www.systemc-ams.org/**
- **Accellera Systems Initiative SystemC Community: http://www.accellera.org/community/systemc/**
- **Accellera Systems Initiative Forums: http://forums.accellera.org/**
- **Fraunhofer SystemC AMS Open Source Offers / COSIDE: http://www.eas.iis.fraunhofer.de/en/business_areas/microelectronic_systems/system_development/open_source.html, http://www.coside.de**
- **H-INCEPTION – Ubuntu 12.04 SystemC AMS/TLM virtual machine: https://www-soc.lip6.fr/trac/hinception/**
- **Beyond DREAMS – SystemC AMS and IP-XACT Methodologies: https://wiki.eas.iis.fraunhofer.de/beyonddreams/**

Resources – post-processing tools

- **Impulse Waveform Viewer for Eclipse (tabular and VCD trace files): http://www.toem.de/index.php/projects/impulse**
- **Gaw – Gtk Analog Wave viewer: http://www.rvq.fr/linux/gaw.php**
- **GTKWave – Digital trace file viewer: http://gtkwave.sourceforge.net/**
- **gnuplot homepage: http://www.gnuplot.info/**
- **GNU Octave: http://www.gnu.org/software/octave/**

Resources – C++

- C++ Reference: http://www.cplusplus.com/reference/
- Stroustrup: A Tour of C++: http://isocpp.org/tour
- C++ video channel on the MSDN: http://channel9.msdn.com/tags/c++
- Stack Overflow C++ Forum: http://stackoverflow.com/questions/tagged/c++
- Boost C++ Libraries: http://www.boost.org/
- clang: C/C++/Objective-C compiler / static canalyzer: http://clang.llvm.org/, http://clang-analyzer.llvm.org/
- MinGW GNU toolchain for Windows: http://www.mingw.org/
- Eclipse CDT (C/C++ Development Tools): http://www.eclipse.org/cdt/
- Microsoft Visual Studio Express: http://www.visualstudio.com/



Thank you

End of Tutorial

accellera
SYSTEMS INITIATIVE