Session 1: SystemC AMS Introduction

Karsten Einwich, Fraunhofer IIS/EAS

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

- 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)

\[ 0 = i_t + \frac{v(w_{it})}{r_{it}} + c_{it} \cdot \frac{d(v(w_{it}) - v(w_p))}{dt} \]

\[ 0 = \frac{v(w_p)}{r_{prefi}} - c_{it} \cdot \frac{d(v(w_{it}) - v(w_p))}{dt} \]

\[ out = \frac{d}{dt} (k1 \cdot in + k2 \cdot out) \]
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
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

- 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.

Equation system:

\[ s_1 = \text{in} \]
\[ s_2 = f_1(s_1) \]
\[ \text{out} = f_2(s_2) \]
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 $\Delta$ of the tagged TDF time point
- TDF samples are scheduled at the first $\Delta$ of the tagged TDF time (and thus valid at least at $\Delta=1$)
**TDF elaboration and simulation**

**TDF module attribute settings:**
Execute all `set_attributes` methods

**TDF time step calculation and propagation:**
Define time step and check consistency

**TDF cluster computability check:**
Define and check cluster schedule

**TDF module initialization:**
Execute all `initialize` methods once

**TDF module activation and processing:**
Continuously execute all `processing` methods

**TDF module post processing:**
Execute all `end_of_simulation` methods once

---

© Accellera Systems Initiative. All Rights Reserved. 18 March 3, 2014
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
};
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

```c
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>`

```cpp
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 )
```
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:

```plaintext
%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:

```matlab
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
#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

```c
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