Refinement of Embedded Analog/Mixed-Signal Systems with SystemC-AMS

Christoph Grimm, Markus Damm, Jan Haase, Jiong Ou, Yaseen Zaidi

Vienna University of Technology
Warning!

This tutorial

• does not teach you syntax of SystemC AMS extensions.
• it does not even want to teach it!
• SystemC-AMS syntax will change – standardization is in progress!

But this tutorial wants to

• describe problems SystemC AMS extensions will solve
• explain underlying motivation and semantics
• teach a methodology for productive use of SystemC AMS extensions!

Not a warning, but - many thanks to Karsten Einwich, Alain Vachoux and Martin Barnasconi who contributed to AMS extensions and this tutorial!
Motivation: More than Moore!

More than Moore: Diversification

- Analog/RF
- Passives
- HV Power
- Sensors
- Actuators
- Biochips

Combining SoC and SiP: Higher Value Systems

Non-digital content System-in-package (SiP)

Information Processing

Baseline CMOS: CPU, Memory, Logic

More Moore: Miniaturization

Beyond CMOS
Refinement of Embedded Analog/Mixed-Signal Systems with SystemC-AMS

- Embedded Analog/Mixed-Signal Systems
- SystemC-AMS Overview
- Refinement methodology
- Outlook
Example 1: Cognitive Radio

Software Defined Radio, Cognitive Radio

- HW/SW System tightly interwoven with AMS
- Ideal Partitioning SW-FPGA-HW-Analog/RF?
- Required for system design: HW/SW + AMS
Example 2: Wireless Sensor Net

Wireless, energy-self sufficient sensor net

- Sensor node = AMS+RF+Processor
- Average power consumption < limit?
- Power consumption depends on
  - AMS+RF hardware,
  - protocol, routing, …
In general: E-AMS

- **Embedded Analog/Mixed-Signal** (E-AMS) system
- *Analog circuits* functionally interwoven with *HW/SW* system
- Functionality/architecture can only be understood as a whole
MoC in E-AMS

Models of Computation used:
- Discrete Event
- Data/signal flow
- Differential equations
- Electrical network

Figure: Karsten Einwich, FhG-IIS/EAS Dresden
Design of E-AMS

For real Analog/Digital Co-Design we have to break this wall!

HW/SW Refinement TLM,…

Modelling

A/D Partitioning (DSP) Algorithm design, Bit widths, …

HW/SW Subsystem

Analog Subsystem

SystemC functional + Data/Signal flow

Functional Level

Architecture Level

Subsystem Level
Refinement of Embedded Analog/Mixed-Signal Systems with SystemC-AMS

- Embedded Analog/Mixed-Signal Systems
- SystemC AMS Extensions Overview
- Refinement methodology
- Outlook
SystemC-AMS Requirements

- Very high simulation performance
  - Virtual prototyping, overall system simulation, ...
  - Tradeoff vs. reduced accuracy acceptable
- Extensible approach
  - Data flow, timed
  - Signal flow
  - Electrical network
  - Transfer functions
  - Extensions from EDA vendors, ...
    - Methodology specific support
    - Integration of simulators, e.g. SPICE!
Layered Approach for Extensibility

FSM, TLM, RTL ...

Electrical Linear Networks

Linear Signal Flow

User extension

DAE solver

Timed Data Flow

Scheduler

Synchronization

SystemC Kernel (DE)
Continuous Time Systems

- Analog behavior = Blockdiagrams with …
  - Static nonlinear functions example: \( y = a \times b \)
  - Dynamic linear functions example: Linear DAEs, Transfer functions, …

- Analog circuits
  - linear (R,L,C, …)
  - switched linear
  - linear + nonlinear: Transistors, Diodes, …
# Electrical Linear Networks (ELN)

| Electrical nodes connect network primitives | // Language primitives: 
|                                           | `sca_eln::sca_node n1;` |
| Each network needs one connection with reference node of class `sca_eln::sca_ref` | `sca_eln::sca_ref gnd;` |
| Hierarchical structure by using ports of the class `sca_eln::sca_port` | `sca_eln::sca_port p1;` |
| Network elements can be instanciated and connected like other modules ... | `sca_eln::sca_r r1("r1");`  
                                           | `r1.value = 2e3; // 2kOhm`  
                                           | `r1.p(w_it);`  
                                           | `r1.n(gnd);` |
TDF controlled sources

```cpp
sca_tdf::sca_tdf2i i1("i1", <gain>);
i1.p(w1);
i1.n(w2);
i1.ctrl(tdf_signal);
// tdf_signal is of type
// double
```

`sca_tdf::sca_tdf2v v1("v1", <gain>);
```

DE (sc_signal<double>) controlled source

```cpp
sca_tdf::sca_sc2i i1("i1", <gain>);
sca_tdf::sca_sc2v v1("v1", <gain>);
```

Conversion of physical sizes to DE and TDF

```cpp
sca_tdf::sca_i2tdf iconv1(“iconv1“);
sca_tdf::sca_v2tdf iconv1(“vconv1“);
...
```
ELN Example: RC low pass filter

SC_MODULE(lp_filter_eln) {
    sca_tdf::sca_in<double> tdf_in;
    sca_tdf::sca_out<double> tdf_out;

    sca_eln::sca_node in, out; // node
    sca_eln::sca_ref gnd;      // reference
    sca_eln::sca_r *r1;        // resistor
    sca_eln::sca_c *c1;        // capacitor
    sca_eln::sca_tdf2v *v_in;  // converter TDF -> U(t)
    sca_eln::sca_v2tdf *v_out; // converter U(t) -> TDF

    SCA_CTOR(lp_filter_eln) {
        v_in = new sca_eln::sca_tdf2v("v_in", 1.0); // scale factor 1.0
        v_in->ctrl(tdf_in);
        v_in->p(in); v_in->n(gnd); // converted to voltage

        r1 = new sca_r("r1", 10000.0); // 10kOhm resistor
        r1->p(in); r1->n(out);

        c1 = new sca_c("c1", 0.0001); // 100uF capacitor
        c1.p(out); c1.n(gnd);

        v_out = new sca_eln::sca_v2tdf("v_out", 1.0); // scale factor 1.0
        v_out1->p(out); v_out1->n(gnd); // filter output
        v_out->ctrl(tdf_out); // converted to TDF
    }
}
Linear Signal Flow Models (LSF)

Linear Signal Flow

- Continuous time
- Directed signals
- Loops allowed are equations that are solved numerically

- Predefined blocks in AMS extensions:
  - Transfer functions
  - add, sub, integ, dt
  - ...

Linear Signal Flow Models (LSF)

LSF low pass filter structure

LSF PI controller structure
Some limitations of prototype (may change!)

- No step width control
- Only linear components publicly available
- Syntax not yet in line with LRM proposal

... certainly some bugs ...

Wait and see ...
Layered Approach for Extensibility

FSM, TLM, RTL ...

Electrical Linear Networks

Linear Signal Flow

User extension

DAE solver

Timed Data Flow

Scheduler

User extension

Synchronization

SystemC Kernel (DE)
Timed Data Flow (TDF)

“cluster“ := set of connected TDF modules

order of computation

circle(period

ipl f1 f1 dec f2 del

rate=2 rate=2

delay=1

T=1
t0=0
Timed Data Flow – Code snippet

**TDF Module:**

```cpp
SCA_TDF_MODULE(par2ser)
{
    sca_tdf::sca_in<sc_bv<8>> in;
    sca_tdf::sca_out<bool> out;
}
```

**Attributes specify timed semantics**

```cpp
void set_attributes()
{
    out.set_rate(8);
    out.set_delay(1);
    out.set_timestep(1, SC_MS);
}
```

**processing() describes computation**

```cpp
void processing()
{
    for (int i=7; i >= 0; i--)
        out.write(in.get_bit(i), i);

    SCA_CTOR(par2ser)
}
```
Converter ports towards discrete event domain

sca_tdf::sc_in < <type> >
sca_tdf::sc_out < <type> >

Note: Time in TDF may run ahead DE time!

sc_time sca_get_time()
TDF example: Mixer model

SCA_TDF_MODULE(mixer) // TDF primitive module definition
{
    sca_tdf::sca_in<double> in1, in2; // TDF in ports
    sca_tdf::sca_out<double> out;       // TDF out ports
    void set_attributes()
    {
        set_module_timestep(1.0, SC_US); // time between activations
    }
    void processing()                  // executed at each activation
    {
        out.write(in1.read() * in2.read());
    }
    SCA_CTOR(mixer) {}}
};
SCA_TDF_MODULE(lp_filter_ltf)
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;
    sca_tdf::sca_ltf_nd ltf;               // computes transfer function
    sca_utio::sca_vector<double> num, den; // coefficients
    void initialize()
    {
        num(0) = 1.0;
        den(0) = 1.0; den(1) = 1.0/(2.0*M_PI*1.0e4); // M_PI=3.1415
    }
    void processing()
    {
        out.write( ltf(num, den, in.read() ) );
    }
    SCA_CTOR(lp_filter_ltf) {}
};

TDF example 2: CT Low pass filter
Current limitations of prototype (may change!)

- Syntax of prototype (0.16) does not yet match LRM proposal (as used in this tutorial)
- activation currently at constant time steps.

*Wait and see …*
Layered Approach for Extensibility

FSM, TLM, RTL ...

Electrical Linear Networks
Linear Signal Flow
User extension
DAE solver
Synchronization
Timed Data Flow
Scheduler
User extension

SystemC Kernel (DE)
Support for hierarchy

SC_MODULEs are used for structuring design hierarchically
• SCA_*** modules primitives for data flow and signal flow modeling
• Use same port classes – but no converter ports allowed
Hierarchy example: RF frontend

SC_MODULE(frontend)  // SC_MODULES used for hierarchical structure
{
    sca_tdf::sca_out<double> if_out;    // use TDF ports to connect with
    sca_tdf::sca_in<double> rf, loc_osc; // TDF ports/signals in hierarchy
    sca_tdf::sca_signal<double> if_sig; //
    mixer* mixer1;
    low_pass* lp1;
    adc* adc1;
    SCA_CTOR(frontend)
    {
        mixer1 = new mixer(“mixer1”);
        mixer1->in1(rf);
        mixer1->in2(loc_osc);
        mixer1->out(if);
        lpf1 = new lp_filter_ltf(“lpf1”);
        lpf1->in(if_sig);
        lpf1->out(if_out);
        }
    }
Refinement of Embedded Analog/Mixed-Signal Systems with SystemC-AMS

- Embedded Analog/Mixed-Signal Systems
- SystemC-AMS Overview
- Refinement methodology
- Outlook
Refinement of E-AMS

- **Computational refinement**
- **Structural refinement**
- **Interface refinement**

SystemC-AMS can break this wall. … if used in the right way!

A/D Partitioning
(DSP) Algorithm design,
Bit widths, …
Executable specification

- Check understanding of spec
- Adapt executable spec to match architecture!
- Data flow, signal flow MoC, double numbers

**Example: BASK receiver (simple …)**
Example: BASK receiver

Frontend: code hierarchy example!
Very simple demodulator could be written as follows:

```cpp
SCA_TDF_MODULE(bask_demodulator)
{
  sca_tdf::sca_in<double> in;
  sca_tdf::sca_out<bool> out;
  void attributes()
  {
    in.set_rate(20000); // port in has 20000 samples/timestep
    in.set_timestep(0.1, SC_MS);
    out.set_delay(1); // port out is delayed by one timestep
  }
  void processing() // Maps 20000 samples to 1 symbol
  {
    double val = 0.0;
    for (int i = 0; i < in.get_rate(); i++)
      val += abs(in.read(i));
    if (val > THRESHOLD) out.write(false); // THRESHOLD in header
    else out.write(true);
  }
  SCA_CTOR(bask_demodulator) {} 
}
```
1. Successively add most important non-ideal effects
   - Adapt model of computation to system partitioning (CT/DT)
   - Add noise, distortion, quantization, accurate step widths
   - Don’t detail the rest – unless you have much time

2. Evaluate overall system behavior, goto (1)

Example for refined mixer model:

```c
void processing() // Mixer refined with distortions and noise
{
    double rf = in1.read();
    double lo = in2.read();
    double rf_dist = (alpha - gamma * rf * rf) * rf; //alpha and gamma
    double mix_dist = rf_dist * lo; //user-defined
    if_out.write( mix_dist + my_noise() );
}
```
Structure refinement at architecture level:

1. Adapt architecture to intended structure
2. Add more accurate components where critical
3. Evaluate architecture behavior, goto (1)

Example BASK receiver:

- loc_osc
- if_sig
- low pass filter
- d_in
- d_in_tlm
- BASK mod.
- HW/SW
- S&H circuit
- RC ladder (Component level)
- clock recovery (e.g. TLM)
Support by converter channels, adapters

- **Converter channels** support structure refinement
  - Adaption of MoC & data types: Encapsulate MoC/Data type converter in hierarchical channels („converter channels“)!

  ![Diagram](image)

  - elec. node with voltage
  - data flow, … bit-accurate

- Also possible for TLM (not shown)
Interface refinement at architecture level

For design of subsystems & verification integration

- Adapt signal types, add clk, enable, etc.
- Blocks are executable spec for subsystem design
- Model can be used for verification integration

Example BASK receiver:

```
adc
loc Osc
low pass
filter
testbench
if_Sig
d_in
frontend

BASK
demod.
HW/SW

controller

clk, enable, ...
```
Support by converter channels, adapters

- **Converter channels** support structure refinement
  - **Adaption of MoC & data types:** Encapsulate MoC/Data type converter in hierarchical channels („converter channels“)!
  - Also possible for TLM (not shown)

- **Adapter translate** 1-port signals to multi-port signals

- **Adapter and converter channels** can be seen as „AMS Transactors“
Refinement of Embedded Analog/Mixed-Signal Systems with SystemC-AMS

- Embedded Analog/Mixed-Signal Systems
- SystemC-AMS Overview
- Refinement methodology
- Outlook
Summary

- SystemC-AMS supports seamless C-based flow for
  - executable spec
  - architecture exploration
  - integration validation
  - virtual prototyping

- Effective use requires integration of simulators for circuit design (Verilog, VHDL, SPICE,…) → EDA vendors?
Outlook

• Standardization in progress
  • You are invited to contribute!

• On agenda
  • RF/Baseband support
  • Nonlinear signal flow, nonlinear networks
  • physical domains

• At DATE:
  • TUESDAY 12:30-14:00 SystemC Interactive Forum
  • Contact
    Martin Barnasconi, NXP (AMS WG chair) or
    Christoph Grimm, TU Vienna (AMS WG vice chair)
Further information, References

- **www.systemc.org**
  (OSCI members only at the moment)

- **www.systemc–ams.org**
  (For information from former SystemC-AMS SG, provides some information for the public)

- Alain Vachoux, Christoph Grimm, Karsten Einwich
  **SystemC Extensions for Heterogeneous and Mixed Discrete/Continuous Systems.**

- Christoph Grimm: **Modeling and Refinement of Mixed Signal Systems with SystemC.**