SystemC AMS extensions: Modeling Strategies and Design Methodology

Christoph Grimm, Markus Damm
Vienna University of Technology
Chair of Embedded Systems
Modeling Strategies and Design Methodology

1. Motivation
2. Foundations
3. Modeling Strategies
4. Design Refinement
5. Outlook
Why care about modeling strategies?

1. SystemC AMS is *not* SPICE

2. SystemC AMS is *not* for circuit design – it’s for overall system modeling!

3. Used in appropriate way, SystemC AMS yields
   - high simulation performance
   - increased design productivity
Objective 1: Learn to apply SystemC AMS ...

This talk gives you ideas on...

- ...when to apply which model of computation (MoC)
- ...how to map a modeling problem to SystemC AMS extensions
Objective 2: Learn to apply it in an effective way!

- What is the best tradeoff?
- How can I keep modeling effort low?
Modeling Strategies and Design Refinement

1. Introduction
2. Foundations
3. Modeling Strategies
4. Design Refinement
5. Outlook
Timed Synchronous Data Flow (TDF) (1)

- A static schedule can be determined by the data rates set at the ports.
Timed Synchronous Data Flow (TDF) (2)

- In SystemC AMS, a *sampling period* is associated to token production/consumption of a port.

E.g. sampling period of 2 ms here...

Implies a sampling period of 20 µs here!
Timed Synchronous Data Flow (TDF) (3)

“TDF-Cluster“

Schedule: B A A C D...D

100x
Systems and Signals

Continuous Systems („Analog“)
- Signals are *piecewise continuous*

Discrete Systems („Digital“)
- Signals are sequences of *samples* (discrete-time)
- Signals are *piecewise constant* (discrete-event)

*(Embedded) (Analog) Mixed-Signal Systems*
Conversion between Signal Types

Controlled sources

Interpolation

Sampling

Sampling at 1st Delta

Event at 1st Delta

AMS extensions

ELN/LSF Converter Modules

TDF Converter Ports

Note: In ELN and LSF there are converter modules that connect with DE. However, a sampling frequency has to be defined!
## Modeling of Analog Systems

<table>
<thead>
<tr>
<th>Continuous time</th>
<th>Discrete time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bidirectional</td>
<td>(mostly) directed</td>
</tr>
<tr>
<td>circuit,</td>
<td>SC circuits</td>
</tr>
<tr>
<td>mechanical system</td>
<td>digital filters</td>
</tr>
</tbody>
</table>

### Example

#### Typical math. description

**Linear system:**
- Time domain
- Frequency domain

<table>
<thead>
<tr>
<th>Continuous time</th>
<th>Discrete time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(\dot{x}, x, u, t) = 0 )</td>
<td>( x_{k+1} = Ax_k + Bu_k )</td>
</tr>
<tr>
<td>( \dot{x} = f(x, u, t) )</td>
<td></td>
</tr>
<tr>
<td>( x = Ax + Bu )</td>
<td></td>
</tr>
<tr>
<td>( H(s) = \frac{X(s)}{U(s)} )</td>
<td>( H(z) = \frac{X(z)}{U(z)} )</td>
</tr>
</tbody>
</table>
Representation of Linear Analog Systems

- **Extraction from networks** => transfer function
  \[ H(s) = \frac{b_n \cdot s^n + b_{n-1} \cdot s^{n-1} + \ldots + b_0}{a_m \cdot s^m + a_{m-1} \cdot s^{m-1} + \ldots + a_0} \]

- **Known corner frequencies** => zero-pole representation:
  \[ H(s) = k \cdot \frac{(s-z_0) \cdot (s-z_1) \cdot \ldots \cdot (s-z_n)}{(s-p_0) \cdot (s-p_1) \cdot \ldots \cdot (s-p_n)} \]

- **From physical equations**, we get state space equations:
  \[
  \dot{x} = Ax + Bu \\
y = Cx + Du
  \]
Discrete-Time Modeling of Analog Systems

1.) Nyquist-Shannon sampling theorem

\[ F_s > 2 \times (f_{\text{max}} - f_{\text{min}}) \]  would need non-causal filter!

\[ \Rightarrow \text{For practical application: } f_s >> 2f_{\text{max}}, \text{ e.g. } f_s = 8 \times f_{\text{max}} \]

2.) Discrete-time approximation of continuous-time transfer functions

E.g. Bilinear transformation: \( s \approx 2F_s \frac{z-1}{z+1} \)

\[ H(s) = \frac{X(s)}{U(s)} \quad \text{Bilinear Transformation} \]
\[ \text{Filter identification} \quad \rightarrow \quad H(z) = \frac{X(z)}{U(z)} \]
Discrete-Time Modeling of RF Systems

- High carrier frequencies $f_c$ would slow down system simulation – other modeling approach: BB modeling

- Complex low-pass equivalent (envelope):

$$x(t) = r(t) \cos(2\pi f_c t + \varphi(t))$$
$$= \text{Re}\{r(t) e^{j(2\pi f_c t + \varphi(t))}\}$$
$$= \text{Re}\{r(t) e^{j\varphi(t)} e^{j2\pi f_c t}\}$$

$$v(t) = r(t) e^{j\varphi(t)}$$
$$= s_i(t) + j s_q(t)$$
Modeling Strategies and Design Refinement

1. Introduction
2. Foundations
3. **Modeling Strategies**
4. Design Refinement
5. Outlook
Simulation performance vs. abstraction

<table>
<thead>
<tr>
<th>MoC</th>
<th>Behavior</th>
<th>Structure</th>
<th>Communication</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDF</td>
<td>Algorithm $H(s)$</td>
<td>Functional blocks</td>
<td>Sequence of samples</td>
<td>Discrete-Time</td>
</tr>
<tr>
<td>LSF</td>
<td>$H(s)$, +, -, *const</td>
<td>Functional blocks</td>
<td>Directed signals</td>
<td>./</td>
</tr>
<tr>
<td>ELN</td>
<td>Lin. macro models</td>
<td>Network with lin. primitives</td>
<td>./</td>
<td>./</td>
</tr>
<tr>
<td>Circuit</td>
<td>DAE</td>
<td>./</td>
<td>./</td>
<td>./</td>
</tr>
</tbody>
</table>
Use of ELN for Macromodeling

Recommended usage in the following cases:

- It’s difficult to provide equations, e.g. transmission lines, power driver
- External components are relevant for the dimensioning of the system, and therefore must show up at the system -level.

For macro modeling:

1. Omit components not needed for idealized functional model
2. Replace transistors with ideal amplifiers or switches
3. If necessary, add appropriate input/output impedances
ELN: Macromodeling a PWM Driver

1. Omit protection diodes
2. Model transistors as switches
3. Don’t forget to model load $L_{coil}$ with internal resistance
SC_MODULE(pwm_driver)
{
    sc_core::sc_in<bool> in;
    sca_eln::sca-terminal out;
    sca_eln::sca_vsource vcc;
    sca_eln::sca_de::sca_rswitch highside, lowside;
    sca_eln::sca_node_ref gnd;
    sca_eln::sca_node node;

    pwm_driver(sc_core::sc_module_name nm, double _val = 5.):
        in("in"), out("out"), vcc("vcc"),
        highside("highside"), lowside("lowside"),
        gnd("gnd"), node("node")
    {
        vcc.offset = _val;
        vcc.p(node);
        vcc.n(gnd);
        highside.ctrl(in);
        highside.p(node);
        highside.n(out);
        lowside.ctrl(in);
        lowside.p(out);
        lowside.n(gnd);
        lowside.off_state=true;
    }
};
Use of LSF for Filters and Control

Recommended use in the following cases:

- Structure of computation is important (e.g. filter structure)
- Access to internal states/signals necessary
- Graphical way to describe differential equations is more convenient
- Delay in control loops is not acceptable
LSF: PID controller w/ programmable parameters

Parameters e.g. from registers in controller (SystemC)
LSF: PID controller

```c
SC_MODULE(lsf_pid_external_control) {
  sca_lsf::sca_in  in;
  sca_lsf::sca_out out;

  sc_core::sc_in<double> p, i, d; // adjustable coefficients

  sca_lsf::sca_de::sca_gain gain_p, gain_i, gain_d; // coefficients to scale gain
  sca_lsf::sca_integ integ;
  sca_lsf::sca_dot dot;
  sca_lsf::sca_add add1, add2;

  lsf_pid_external_control( sc_core::sc_module_name name )
    : in("in"), out("out"), p("p"), i("i"), d("d"),
      gain_p("gain_p"), gain_i("gain_i"), gain_d("gain_d"),
      dot("dot"), integ("integ"), add1("add1"), add2("add2"),
      sig_gain("sig_gain"), sig_integ1("sig_integ1"), sig_integ2("sig_integ2"),
      sig_dot1("sig_dot1"), sig_dot2("sig_dot2"), sig_add("sig_add")
  {
    gain_p.x(in); gain_p.y(sig_gain); gain_p.inp(p);
    gain_i.x(in); gain_i.y(sig_integ1); gain_i.inp(i);
    gain_d.x(in); gain_d.y(sig_dot1); gain_d.inp(d);
    integ.x(sig_integ1); integ.y(sig_integ2);
    dot.x(sig_dot1); dot.y(sig_dot2);
    add1.x1(sig_gain); add1.x2(sig_integ2); add1.y(sig_add);
    add2.x1(sig_add); add2.x2(sig_dot2); add2.y(out);
  }

  private:
    sca_lsf::sca_signal sig_gain, sig_integ1, sig_integ2, sig_dot1, sig_dot2, sig_add;
};
```
Use of TDF for overall data/signal flow

Recommended use cases

- Analog signal processing
- DSP functionality (even if implemented in digital!)

To consider:

1. Determine maximal frequencies in different blocks
2. Define sampling frequency $> f_{\text{max}}$ ($2 \times f_{\text{max}}$ not enough!)
3. Consider delay in cyclic dependencies
Mapping to SystemC AMS MoC

For each “functional block”:

- **Signal processing?**
  - no
  - yes

- **Control system?**
  - no
  - yes

- **Interaction with analog/physical?**
  - yes
  - no

(Over)sampling possible?

- yes
  - Delay in loops?
    - no
    - Equations (e.g., H(s)) or block diagram?
      - no
      - yes
    - Separate static non-linear & dynamic linear?
      - yes, non-linear part
      - yes, linear part
      - no
        - Need to model non-linear elements?
          - no
          - yes
        - Non-linear solver
      - yes, non-linear part
      - yes, linear part
        - TDF
      - no
      - LSF
      - ELN
      - Spice, ...

Partition system in functional blocks:
1.) Separate control- and data flow
2.) Don’t split in places where there is strong bi-directional communication
Example (Functional, Architecture level)

Signal processing function, oversampling possible, no loops

Analog part, not easy to describe in equations

Non-functional, architecture level description of digital HW/SW
Modeling Strategies and Design Refinement

1. Introduction
2. Foundations
3. Modeling Strategies
4. Design Refinement
5. Outlook
Modeling in an effective way ...

System design:
- Executable specification
- Architecture exploration
- Architecture model

Module design:
- Change module specification
- Specification of module properties
- Behavioral model
- Characterization of module
- Analog circuit

System design:
- Module design:
  - Change module specification
  - Specification of module properties
  - Behavioral model
  - Characterization of module
  - Analog circuit

System integration:
- Change design
Design Refinement

- Design Refinement interactively augments functional model (e.g. executable specification) successively with properties of implementation
  - Early, immediate analysis/verification of each step by simulation
  - Enables fast and effective exploration of architectures

- Classification
  - Refinement of Behavior
  - Refinement of Structure
  - Refinement of Interfaces/Communication
Refinement of Behavior

**Objective:** Evaluate impact of non-ideal behavior of assumed architecture/implementation using functional model

**Method:** Add non-ideal effects to executable specification by modifying `processing()` method

```c
void processing() {
    out.write(f(input) + noise());
}
```

Model based design with Matlab/Simulink®  Design refinement with SystemC AMS
Refinement of Behavior - Nonlinearities

- E.g. add nonlinearities to mixer and monitor impact on bit error rate

```c
void processing() // Mixer refined with distortions and noise
{
    double rf = rf_in.read();
    double lo = lo_in.read();
    double rf_dist = (alpha - gamma * rf * rf) * rf;
    //alpha and gamma defined by user …
    double mix_dist = rf_dist * lo;
    if_out.write( mix_dist );
}
```
Refinement of Structure

- **Objective**: Compare performance of different A/D/HW/SW partitionings more accurately

- **Method**: Map functions to assumed or existing processor (=change structure, MoC)

```
rf_sig
if_sig
mixer
frontend
if_sig
lp_filter_tdf
gain
agc_ctrl
gain
ctrl_config
d_in
BASK
demod.
symbol
loc_osc
```
Refinement of Interfaces

- **Objective:** Prepare validation of system integration - All ports accurately as in implementation (same types, same number, clocks, enable-signals, ...)

- **Methods:** Add ports, where appropriate use custom adapter classes or custom converter modules
Set up your application specific library!

- The SystemC AMS extensions as language standard only offers basic primitives— it will live from tool kits and your application specific libraries!

- The open structure of the SystemC AMS extensions allows the creation of a wide variety of application specific libraries

- Example available (free):
  TU Vienna SystemC AMS Building Block Library
  - Focuses on refinement of WSN and communication systems
  - Components for modeling communication systems (ASK, OFDM, …)
  - Functions for modeling non-ideal behavior (noise, …)
  - „Air object“ for modeling communication between sensor nodes
Modeling Strategies and Design Refinement

1. Introduction
2. Foundations
3. Modeling Strategies
4. Design Refinement
5. Summary & Outlook
Summary and Outlook

- Using SystemC AMS extensions is no black magic!

- Create *abstract* models and apply MoCs efficiently
  - Use discretization and/or linearization where possible
  - Use linear nets, transfer functions where necessary
  - Baseband modeling permits modeling of RF systems
  - Start with overall system simulation and refine towards architecture

- Use and create *libraries* with your own IP blocks and tools! Productivity increases only with appropriate libraries ...
Links and References

- [www.systemc.org](http://www.systemc.org) – SystemC Homepage
- [www.systemc-ams.org](http://www.systemc-ams.org) – SystemC AMS Homepage
- [www.systemc-ams.org/BB_library.html](http://www.systemc-ams.org/BB_library.html) - communication library