Outline

- Introduction
- Language standard overview
- AMS models of computation
- Dynamic and reactive extensions in SystemC AMS 2.0
- Example: DC motor control with Pulse Width Modulator
- Conclusions
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)
SystemC AMS applications

Communication systems

Imaging systems

Automotive systems

Image courtesy of STMicroelectronics
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

2006: SystemC AMS Draft 1 LRM

2007: SystemC 2.2 released

2008: SystemC AMS Draft 1 LRM

2009: SystemC TLM 2.0 standard

2009: SystemC Synthesizable Subset Draft 1.3

2010: SystemC AMS 1.0 LRM standard

2011: IEEE Std 1666-2011 LRM

2010: SystemC AMS 1.0 PoC released by Fraunhofer IIS/EAS

2012: SystemC AMS 2.0 draft standard

2012: SystemC AMS 2.0 PoC released by Accellera Systems Initiative

2013: SystemC AMS 2.0 LRM standard

2013: SystemC AMS 2.0 PoC test version

2014: IEEE 1666.1 (SystemC AMS) started
Positioning SystemC AMS

Functional

Architecture

Implementation

SystemC AMS

VHDL-AMS, Verilog-AMS

SystemVerilog, VHDL, Verilog
**Example: Communication System**

- **Tight interaction** between digital HW/SW and AMS sub-systems
  - Signal path: Communication protocol stack – modeling including PHY layer
  - Control path: more and more HW/SW calibration and control of analog blocks

- **Architecture modeling using SystemC, TLM and AMS**
Industry requirements and needs

- **Design of True Heterogeneous Systems-on-a-chip**
  - Analog, Mixed-signal, RF, digital HW/SW (processor) interaction
  - Multi-domain, high frequencies, high bandwidth, configurable AMS components

- **Support different levels of design abstraction**
  - Functional modeling, architecture design, (abstract) circuit representations

- **Support different use cases – also for AMS!**
  - Executable specification, architecture exploration, virtual prototyping, integration validation

Need for Virtual Prototype Environments which enable inclusion of digital HW/SW and abstract AMS/RF system-level representations
SystemC AMS advantages

- **SystemC, thus C++ based**
  - Enjoy the power of C++ (and its wealth of libraries)
  - Object oriented – modular and extendable
  - AMS class libraries available for basic building blocks (analog primitives)
  - Tool independent / EDA-vendor neutral

- **Modeling in multiple abstractions using one simulator**
  - No need for complex multi-kernel/co-simulation
  - No difficult APIs
  - Converter models and ports are part of the language
  - Allows abstraction along four axis
    - structure, behavior, communication and time/frequency

- **Transparent modeling platform**
  - Access to simulation kernel to ease debugging and introspection
Model abstraction and formalisms

**Use cases**
- Executable specification
- Virtual prototyping
- Architecture exploration
- Integration validation

**Model abstractions**
- **Discrete-time**
  - static non-linear
- **Continuous-time**
  - dynamic linear

**Modeling formalism**
- Timed Data Flow (TDF)
- Linear Signal Flow (LSF)
- Electrical Linear Networks (ELN)

**Behavior**
- Non-conservative behavior
- Conservative behavior
AMS models in Virtual Prototypes realistic?

Yes, as long as you use the right language and abstraction method

Expected simulation speed improvement [1]

SystemC AMS extensions LRM

- Language Reference Manual defines the standard of the SystemC AMS extensions

- Contents
  - Overview
  - Terminology and conventions
  - Core language definitions
  - Predefined models of computation
  - Predefined analyses
  - Utility definitions
  - Introduction to the SystemC AMS extensions (Informative)
  - Glossary (Informative)
  - Deprecated features (Informative)
  - Changes between SystemC AMS 1.0 and 2.0 standard (Informative)
SystemC AMS User’s Guide

- Comprehensive guide explaining the basics of the AMS extensions
  - TDF, LSF and ELN modeling
  - Small-signal frequency-domain modeling
  - Simulation and tracing
  - Modeling strategy and refinement methodology

- Many code examples

- Application examples
  - Binary Amplitude Shift Keying (BASK)
  - Plain-Old-Telephone-System (POTS)
  - Analog filters and networks

- Has proven it’s value: reference guide for many new users
## SystemC AMS language features

<table>
<thead>
<tr>
<th>Mixed-Signal Virtual Prototypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>written by the end user</td>
</tr>
</tbody>
</table>

### SystemC methodology-specific elements
- Transaction-level modeling (TLM), Cycle/Bit-accurate modeling, etc.

### AMS methodology-specific elements
- Elements for AMS design refinement, etc.

#### Time-domain and small-signal frequency-domain simulation infrastructure
- (synchronization layer)

### Scheduler

### Linear DAE solver

#### Timed Data Flow (TDF)
- Modules
- Ports
- Signals

#### Linear Signal Flow (LSF)
- Modules
- Ports
- Signals

#### Electrical Linear Networks (ELN)
- Modules
- Terminals
- Nodes

### SystemC Language Standard (IEEE Std. 1666-2011)
SystemC AMS methodology elements

- **Support design refinement using different models of computation**
  - Timed Data Flow (TDF) - efficient simulation of discrete-time behavior
  - Linear Signal Flow (LSF) - simulation of continuous-time behavior
  - Electrical Linear Networks (ELN) - simulation of network topology & primitives

- **Using namespaces**
  - Clearly identify the used model of computation
  - Unified and common set of predefined classes, (converter) ports and signals

- **Examples**
  - Module: `sca_tdf::sca_module`  `sca_lsf::sca_module`
  - Input port: `sca_tdf::sca_in`  `sca_lsf::sca_in`
  - Output port: `sca_tdf::sca_out`  `sca_lsf::sca_out`
  - Signals: `sca_tdf::sca_signal`  `sca_lsf::sca_signal`
  - Nodes (electrical only): `sca_eln::sca_node`
  - Terminal (in/output port, electrical only): `sca_eln::sca_terminal`
### Timed Data Flow (TDF)

- **TDF is based on synchronous dataflow**
  - A module is executed if enough samples are available at its input ports
  - The number of read/written samples are constant for each module activation
  - The scheduling order follows the signal flow direction

- **The function of a TDF module is performed by**
  - reading from the input ports (thus consuming samples)
  - processing the calculations
  - writing the results to the output ports

- **The TDF model of computation is a discrete-time modeling style**
Linear Signal Flow (LSF)

- Continuous-time behavior described in the form of block diagrams
  - LSF primitives describe relations between variables of a set of linear algebraic equations

- Only a single quantity is used to represent the signal
  - There is no dependency between flow (e.g. current) and potential (e.g. voltage) quantities
  - Uses directed real-valued signals, resulting in a non-conservative system description
Electrical Linear Networks (ELN)

- ELN modeling style allows the instantiation of electrical primitives
  - Connected ELN primitive modules will form an electrical network
- The electrical network is represented by a set of differential algebraic equations
  - following Kirchhoff’s voltage law (KVL) and Kirchhoff’s current law (KCL)
- ELN captures conservative, continuous-time behavior
Dynamic Timed Data Flow modeling

- Abstract modelling 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 behaviour where frequencies (and time steps) change dynamically
  - Often the case for clock recovery circuits or capturing jitter

- Modelling systems with varying (data) rates
  - E.g. multi-standard / software-defined radio (SDR) systems

This requires a *dynamic* and *reactive* Timed Data Flow modeling style
- Basically introduce variable time step instead of fixed/constant time step
Example: DC motor control

- Functional model in the Laplace domain modelled in SystemC AMS
- To achieve high accuracy, many module activations are necessary when using fixed time steps (AMS 1.0)
- Introducing *Dynamic TDF* to only compute when necessary, due to dynamic time step mechanism (AMS 2.0)
DC motor control loop behavior

\[ i_{\text{meas}}(t) \]
\[ i_{\text{ref}} \]
\[ v_{\text{drv}}(t) \]

- \( t_{\text{ramp}} \)
- \( t_{\text{duty}} \)
- \( t_{\text{period}} \)
Example of Pulse Width Modulator (1)

```cpp
// pwm_dynamic.h

#include <cmath>
#include <systemc-ams>

SCA_TDF_MODULE(pwm) // for dynamic TDF, we can use the same helper macro to define the module class
{
    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    pwm( sc_core::sc_module_name nm, ... )
    : in("in"), out("out") {}

    void set_attributes()
    {
        does_attribute_changes(); // module allowed to make changes to TDF attributes
        accept_attribute_changes(); // module allows attribute changes made by other modules
    }

    void change_attributes() // new callback to change attributes during simulation
    {
        double t = get_time().to_seconds(); // current time
        double t_pos = std::fmod( t, t_period); // time position inside pulse period
        ...
    }
```
Example of Pulse Width Modulator (2)

```cpp
if ( t_pos < t_ramp ) {
    // rising edge
    request_next_activation( t_ramp - t_pos, sc_core::SC_SEC );
} else if ( t_pos < t_ramp + t_duty ) {
    // plateau
    request_next_activation( ( t_ramp + t_duty ) - t_pos, sc_core::SC_SEC );
} else if ( t_pos < t_ramp + t_duty + t_ramp ) {
    // falling edge
    request_next_activation( ( t_ramp + t_duty + t_ramp ) - t_pos, sc_core::SC_SEC );
} else {
    // return to initial value
    request_next_activation( t_period - t_pos, sc_core::SC_SEC );
}
```

void processing()
{
    ... // PWM behavior
}

private:
    ... // member variables
};
### TDF vs. Dynamic TDF Comparison

<table>
<thead>
<tr>
<th>TDF model of computation variant</th>
<th>$t_{step}$ (ms)</th>
<th>$t_{ramp}$ (ms)</th>
<th>$t_{period}$ (ms)</th>
<th>Time accuracy (ms)</th>
<th>#activations per period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional TDF</td>
<td>0.01 (fixed)</td>
<td>0.05</td>
<td>5.0</td>
<td>0.01 ($= t_{step}$)</td>
<td>500</td>
</tr>
<tr>
<td>Dynamic TDF</td>
<td>variable</td>
<td>0.05</td>
<td>5.0</td>
<td>defined by sc_set_time_resolution()</td>
<td>4</td>
</tr>
</tbody>
</table>

- **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.
Summary and outlook

- SystemC AMS developments are fully driven and supported by European industry: NXP, ST, Infineon, and Continental
  - Applications: communication, automotive and imaging systems design

- SystemC AMS is a mature and proven standard
  - SystemC AMS 1.0 was released in March 2010, introducing efficient AMS modeling and system-level simulation
  - SystemC AMS 2.0 was released in March 2013, introducing reactive and dynamic behavior for AMS computations

- Third party Proof-of-Concept implementation for SystemC AMS 1.0 available under Apache 2.0 license
  - Thanks to Fraunhofer IIS/EAS Dresden

- Commercial design environments supporting SystemC AMS are available in the market
More information

- www.accellera.org
- www.accellera.org/downloads/standards/systemc/ams
- www.accellera.org/community/articles/amsspeed
- www.accellera.org/community/articles/amsdynamictdf
- www.systemc-ams.org
Thank you