What’s new in SystemC AMS 2.0

Karsten Einwich     Fraunhofer IIS/EAS
Martin Barnasconi   NXP
Accellera SystemC-AMS Working Group Member
# SystemC AMS – History

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>Open SystemC Initiative (OSCI) announced</td>
</tr>
<tr>
<td>2000</td>
<td>SystemC 1.0 released (sourceforge.net)</td>
</tr>
<tr>
<td>2002</td>
<td>OSCI SystemC 1.0.2</td>
</tr>
<tr>
<td>2005</td>
<td>IEEE Std 1666-2005 LRM</td>
</tr>
<tr>
<td>2005</td>
<td>SystemC Transaction level modeling (TLM) 1.0 released</td>
</tr>
<tr>
<td>2007</td>
<td>SystemC 2.2 released</td>
</tr>
<tr>
<td>2009</td>
<td>SystemC TLM 2.0 standard</td>
</tr>
<tr>
<td>2009</td>
<td>SystemC Synthesizable Subset Draft 1.3</td>
</tr>
<tr>
<td>2011</td>
<td>IEEE Std 1666-2011 LRM</td>
</tr>
<tr>
<td>2012</td>
<td>SystemC 2.3</td>
</tr>
<tr>
<td>1999</td>
<td>First C-based AMS initiatives (AVSL, MixSigC)</td>
</tr>
<tr>
<td>~2000</td>
<td>SystemC-AMS study group started</td>
</tr>
<tr>
<td>2005</td>
<td>First SystemC-AMS PoC released by Fraunhofer</td>
</tr>
<tr>
<td>2006</td>
<td>OSCI AMSWG installed</td>
</tr>
<tr>
<td>2008</td>
<td>SystemC AMS Draft 1 LRM</td>
</tr>
<tr>
<td>2010</td>
<td>SystemC AMS 1.0 standard</td>
</tr>
<tr>
<td>2010</td>
<td>SystemC AMS 1.0 PoC released by Fraunhofer</td>
</tr>
<tr>
<td>2012</td>
<td>SystemC AMS 2.0 draft standard</td>
</tr>
<tr>
<td>2013</td>
<td>SystemC AMS 2.0 standard</td>
</tr>
<tr>
<td>2013</td>
<td>SystemC AMS 2.0 PoC test version</td>
</tr>
</tbody>
</table>
SystemC-AMS 2.0 – News Summary

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

- **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
Application for Dynamic TDF

- Automotive domain
- PWM stages
- Reaction to threshold crossings (e.g. for modeling PLL’s)
- Dataflow (dsp) algorithm with clock recovery
- Adaptive systems – systems supporting adjustable data rates or different/configurable algorithms
- Nonlinear dynamic systems
Timed Dataflow

- Dataflow is an untimed MoC

- Timed dataflow tags each sample and each module execution with an absolute time point

- Therefore the time distance (timestep) between two sample/two executions is assumed as constant

- This time distance has to be specified

- Enables synchronization with time driven MoC like SystemC discrete event and embedding of time dependent functions like a continuous time transfer function
If more than one timestep assigned consistency will be checked
Problems defining dynamic timesteps

- Each change requested by one module of the cluster influences the timestep of all modules inside the cluster.

- Changes can only become valid after the whole cluster has been calculated.

- Runtime inconsistencies may occur.
Concept Dynamic Timed Dataflow (DTDF)

- DTDF is an extension of the current TDF MoC
- All modules in a cluster must support DTDF if one module requests DTDF feature
- Attributes (e.g. the timestep) can only be changed before the cluster execution starts

- An additional optional callback of sca_tdf::sca_module – change_attributes is executed before the cluster start – only in the context of this callback attributes can be changed
- TDF Modules can be signed to support DTDF be calling the method accept_attribute_changes() / does_attribute_changes() in the context of the constructor or set_attributes() and may be dynamically
New AMS 2.0 Dynamic TDF features

New callback and member functions to support Dynamic TDF:

- **change_attributes() / reinitialize()**
  - callback provides a context, in which the time step, rate, or delay attributes of a TDF cluster may be changed

- **request_next_activation(…)**
  - member function to request a next cluster activation at 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
DC motor control loop behavior

$i_{meas}(t)$

$v_{drv}(t)$

$i_{ref}$

$t_ramp$

$t_duty$

$t_period$

$t/sec$
Example for TDF Module using dynamic Feature

```cpp
#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);
    }

    sca_tdf::sca_in<double> in;
    sca_tdf::sca_out<double> out;

    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

<table>
<thead>
<tr>
<th>TDF model of computation variant</th>
<th>$t_{\text{step}}$ (ms)</th>
<th>$t_{\text{ramp}}$ (ms)</th>
<th>$t_{\text{period}}$ (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 ($= t_{\text{step}}$)</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.
Methods for reactivity and dynamic timestep

**set_timestep, set_rate, set_delay**

Same methods like for set_attributes – consistency will be checked

```cpp
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& );
```

Requests an additional activation, same argument combinations like next_trigger

Limits the next timestep

Zero timesteps allowed – analog solver will invalid the last timestep and re-calculate
Decoupling Clusters

Timing Domain 1
e.g. DTDF

Timing Domain 2
e.g. TDF
Decoupling clusters

- Connecting (D)TDF modules with **non-multiple timesteps**
- Connecting **TDF and DTDF** clusters

- Problem: analogue continuous semantic requires **interpolation**
  - Interpolation requires the next resolution point

- We will use special **out port** for decoupling
  - A outport, which is able to interpolate must have at least one sample timestep delay
  - A outport which holds the value will have the evaluate-update semantic (if the read and write time is equal, the old value will be read)
  - We will have the possibility to apply a user-defined interpolation method
Example Decoupling Module

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 max.
        //possible timestep
        out.set_ct_delay(0.05, sc_core::SC_MS);
    }

};
Repetition of Timesteps

- LSF and ELN networks deriving the timestep from the connected TDF cluster

- Using the dynamic SystemC-AMS 2.0 feature (e.g. request_next_activation) zero timesteps are allowed

- A zero timestep 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 timestep)

- This enables solver iterations and modelling of nonlinear dynamic behaviour
Dynamic nonlinear Modeling – Simple PWL Diode

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

  void set_attributes()
  {
    vdiode.set_delay(1);
    does_attribute_changes();
  }

  void initialize()
  {
    vdiode.initialize(0.0);
    recalculate = ron_state = false;
  }

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

  void processing();

  double ron, roff, vthres;

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

  private:
    bool ron_state, recalculate;
};

void characteristic::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);
  }
}
Applications for SystemC AMS

- Invehicle networks
- xDSL systems
- WLAN chipsets
- Imaging sensors
- Breaking systems
- Airbag systems
- Powertrain
- Sensor circuits (magnetic, pressure, gyro, …)
- Smart Cards
- Driving systems
- Power supply
- …
Tools for SystemC AMS
More information

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


www.systemc-ams.org

SystemC-AMS 1.0 and 2.0 “proof-of-concept” library
http://www.eas.iis.fraunhofer.de/systemcamsdownloads