SystemC AMS Update

Karsten Einwich

COSEDA technologies

SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY
OCT 23, 2018 | MUNICH | GERMANY

Presentation Copyright Permission

SystemC AMS update

Agenda

- SystemC AMS Update
  - Generalized AC
  - Piece wise linear
  - Interactive tracing
  - Physical domains
- Future thoughts

SystemC AMS update

Generalized AC - Motivation

- Provide AC-Implementation for SC-modules (e.g. implemented digital filter)
- Provide AC-implementation for hierarchical models, which can not be modelled as composed submodules (e.g. SD-converter)
- Cross boundaries – e.g. bussystems for AC-simulation

SystemC AMS AC Modeling

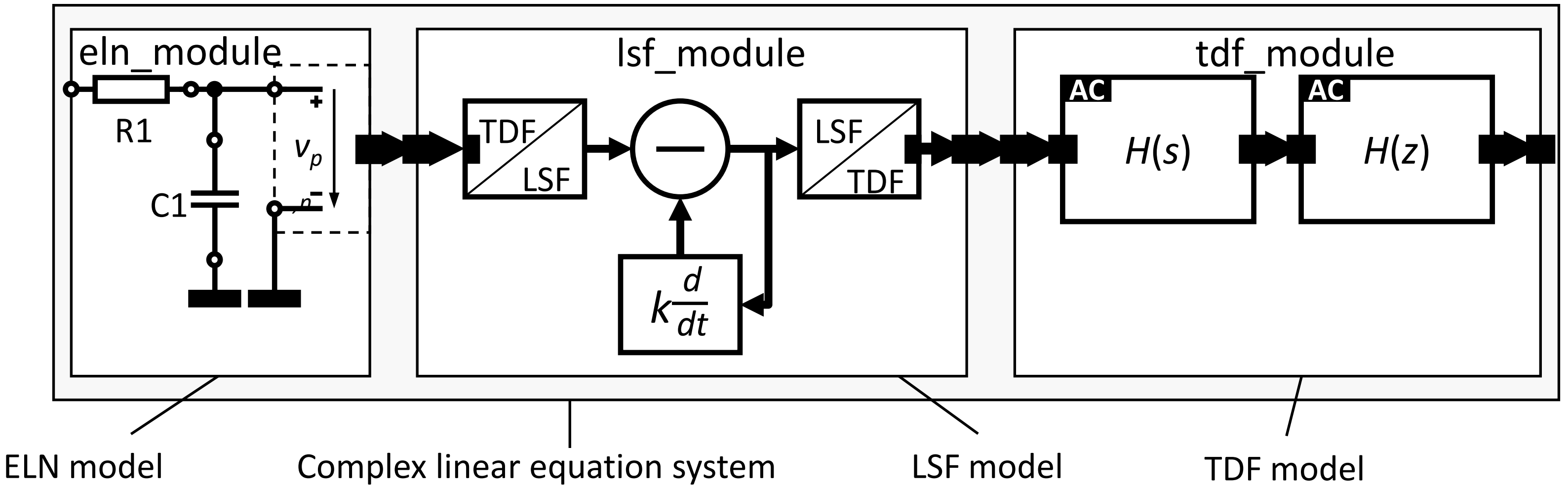
Small Signal Frequency Domain Analysis (AC-Analysis)

- AC-analysis:
  - Calculates linear complex equation system stimulated by AC-sources
  - -> Linear frequency dependent transfer function (bode diagram)
- AC noise domain
  - solves the linear complex equation system for each noise source contribution (other source contributions will be neglected)
  - adds the results arithmetically

- ELN and LSF description are specified in the frequency domain
- TDF description must specify the linear complex transfer function of the module inside the callback method ***ac_processing*** (otherwise the out values are assumed zero)
  - This transfer function can depend on the current time domain state (e.g. the setting of a control signal)

SystemC AMS AC Modeling

Small-Signal Frequency-Domain Analysis for ELN, LSF and TDF



- Linear equation system contribution for LSF/ELN:

$$q(t) = A\dot{x} + Bx \;\rightarrow\; \underset{\substack{\uparrow\\ \text{Sources}}}{q(f)} = Aj\omega x(f) + Bx(f) \rightarrow 0 = Cx(f) - q(f) \quad (C = Aj\omega + B)$$

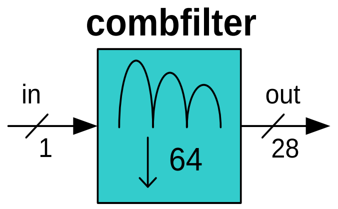
SystemC AMS AC Modeling

Frequency Domain Description for TDF Models

```
SCA_TDF_MODULE(combfilter)
{
  sca_tdf::sca_in<bool>          in;
  sca_tdf::sca_out<sc_int<28> > out;

  void set_attributes()
  {
    in.set_rate(64);  // 16 MHz
    out.set_rate(1);  // 256 kHz
  }

  void ac_processing()
  {
    double      k  = 64.0;
    double      n  =   3.0;

    // complex transfer function:
    sca_complex h;
    h = pow( (1.0 - sca_ac_z(-k)) /
             (1.0 - sca_ac_z(-1)),n);

    sca_ac(out) = h * sca_ac(in) ;
  }
```

```
void processing()
{
      int x, y, i;
      for (i=0; i<64; ++i) {
          x = in.read(i);
      …
      out.write(y);
}
  SCA_CTOR(combfilter)
  {
      …
  }
};
```

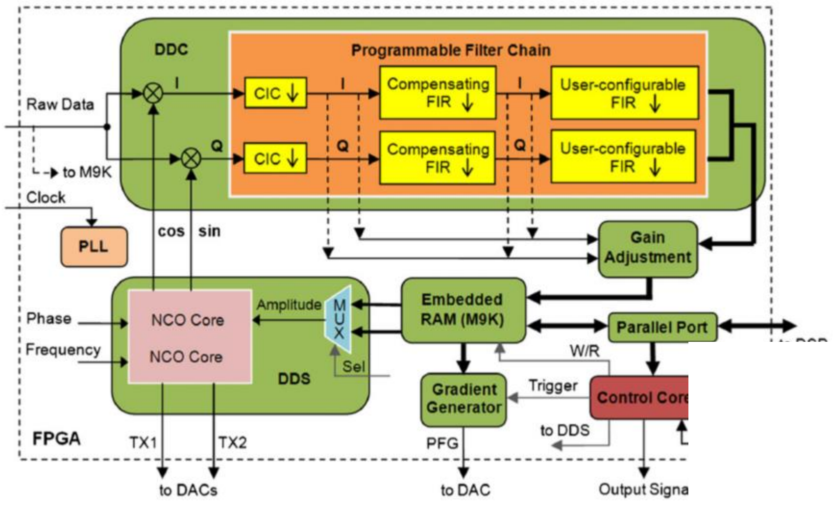
$$H(z) = \left(\frac{1 - z^{-k}}{1 - z^{-1}} \right)^n$$



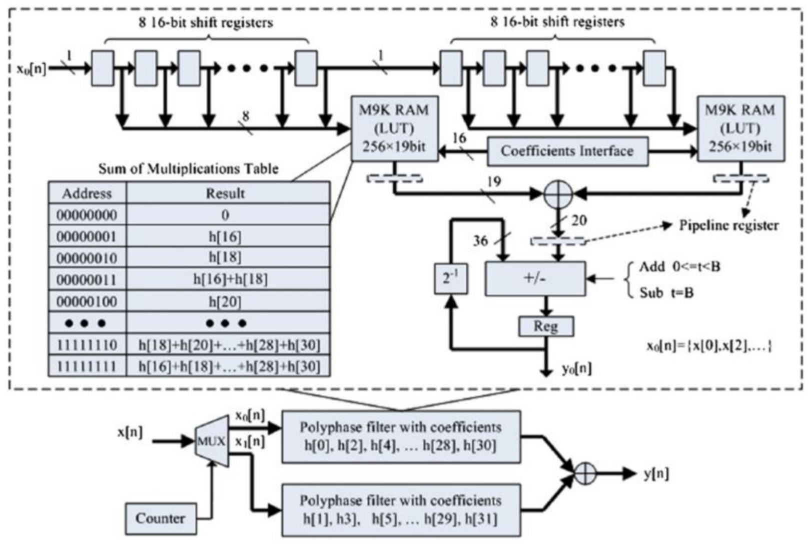
$$z = e^{j2\pi f/f_s}$$

Generalized AC

Motivation



Weinan Tang and Weimin Wang
“A single-board NMR spectrometer based on a software defined radio architecture”



Sum of Multiplications Table

Address	Result
00000000	0
00000001	h[16]
00000010	h[18]
00000011	h[16]+h[18]
00000100	h[20]
• • •	• • •
11111110	h[18]+h[20]+…+h[28]+h[30]
11111111	h[16]+h[18]+…+h[28]+h[30]

Generalized AC

Concept

- Each SC-module can be derived from a sca_ac_module base class
- This base class has a pure virtual method for providing the ac transfer function
- Each sc_core::sc_port_base connected to a sc_core::sc_interface can be accessed via sca_ac_analysis::sca_ac and thus used as an AC-in or outport
- All AC-modules instantiated as childs (lower in hierarchy) of an sca_ac_module are ignored

Generalized AC

Current Implementation

```
class sca_ac_object
{
public:

void ac_enable();
void ac_disable();
bool is_ac_enabled();

sca_ac_object();

virtual ~sca_ac_object();
};

class sca_ac_module : public sca_ac_object
{
  public:

      virtual void ac_processing()=0;
  protected:

     virtual ~sca_ac_module(){}
};

template<class T>
sca_complex& sca_ac(const sc_core::sc_out<T>&);

template<class T>
const sca_complex& sca_ac(const sc_core::sc_in<T>&);
```

- Base class provides some “gimmick” methods – to activate deactivate the ac module during elaboration
- Base class used to provide the ac-implementation
- Port access functions

Generalized AC

Example

```
SC_MODULE( mod_lp ), sca_ac_analysis::sca_ac_module
{
  sc_core::sc_in<int>  inp; // input port
  sc_core::sc_out<int> outp; // output port


  void ac_processing()
  {
    sca_util::sca_complex in = sca_ac_analysis::sca_ac(inp);

    sca_ac_analysis::sca_ac(outp) = sca_ac_analysis::sca_ac_ltf_nd(num,den,in);
  }

  SC_CTOR(mod_lp)
  {
    num(0)=1.0;

    den(0)=1.0;
    den(1)=1.0/(2.0*M_PI*1e3);
  }

private:

  sca_util::sca_vector<double> num, den;

};
```

Piece Wise Linear

Motivation

- Extension to SystemC AMS ELN and LSF
- Permits modelling of non-linear behavior by a piece wise linear approximation
- Especially efficient for switched circuits
  - PWM, DCDC-converter, …

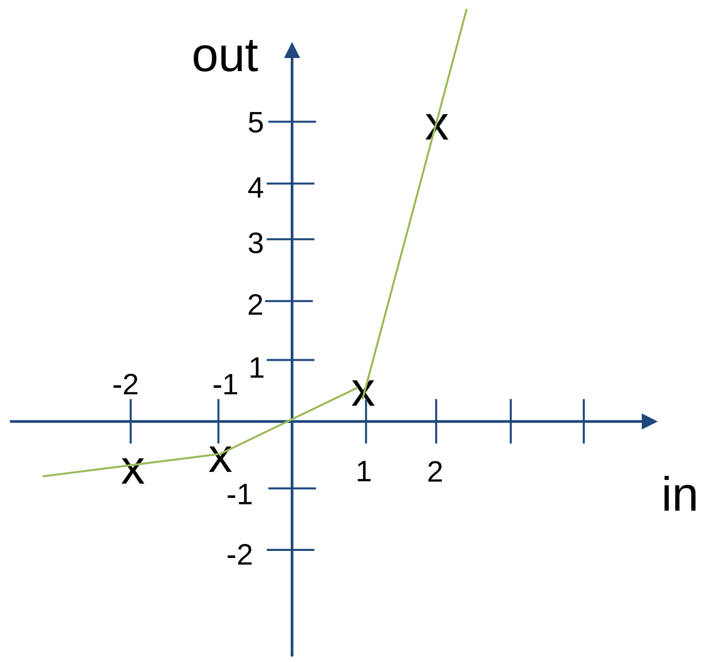
Piece Wise Linear

SCA PWL Libraries – Principle

- The SystemC AMS standard defines linear elements for electrical networks and signal flow only
- The COSIDE piece wise linear models permit the use of approximated non-linear elements without a significant influence on the solver performance and robustness in case of abstract models
- Well suited for non-linear circuits, which have different linear working points (e.g. switched networks with diodes)
- For ELN we provide an additional double vector parameter pwl_value for all controlled sources (sca_cccs, sca_ccvs, sca_vccs, sca_vcvs) and for LSF we provide this additional parameter for sca_lsf::sca_gain
  - If the vector is not empty (otherwise the original scalar parameter value is used) it will be interpreted as piece wise linear characteristic – input-output value pairs
- For ELN we provide piece wise constant voltage controlled resistor and capacitor

Piece Wise Linear

PWL Vector Definition

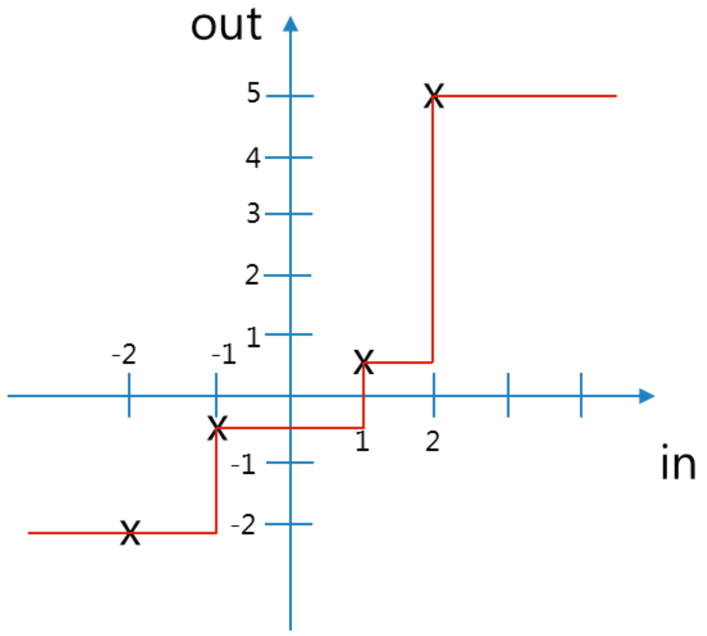


- The vector represents value pairs – vector size (number of pairs) must be at least 2
- Before the first and after the last point the segment will be continued
- The solver will not skip segments – the solver reduces internally the step width
- sca_eln::sca_cccs, sca_eln::sca_ccvs, sca_eln::sca_vccs, sca_eln::sca_vcvs, sca_lsf::sca_gain

```
pwl_value=sca_create_pair_vector(-2.0,-0.6, -1.0,-0.4, 1.0,0.4, 2.0,5.0)
pwl_value={ {-2.0,-0.6} , {-1.0,-0.4}, {1.0,0.4}, {2.0,5.0} }  //C++11
```

Piece Wise Linear

PWC (piece wise constant) Vector Definition

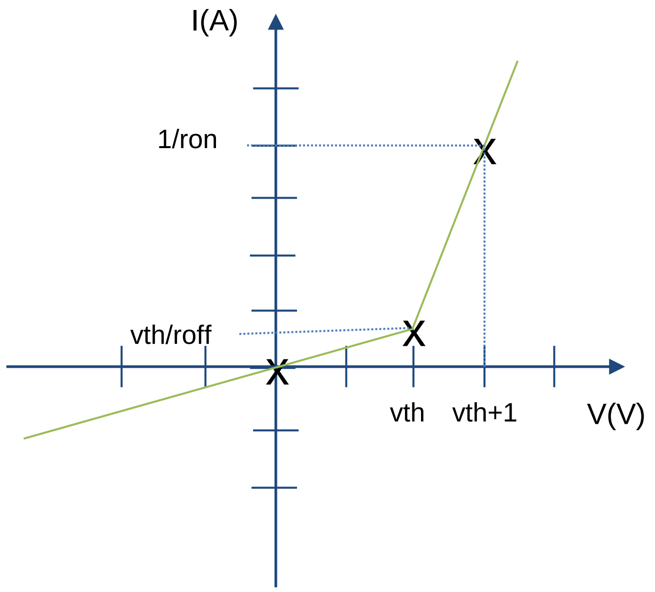


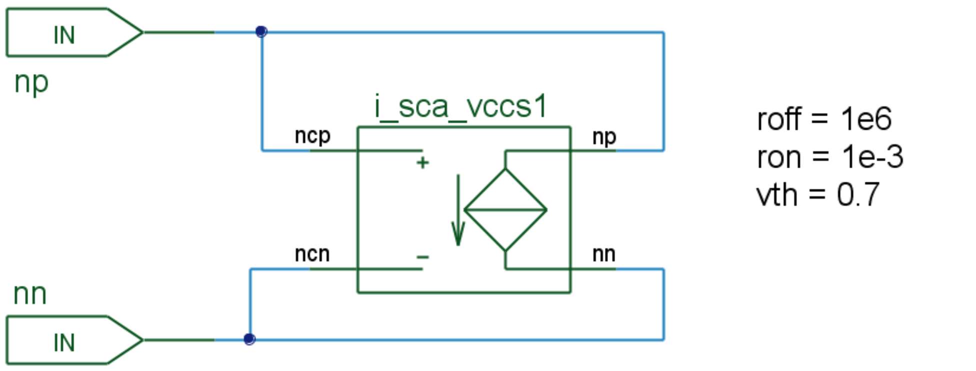
- The vector represents value pairs – the vector must contain at least one pair
- Before the first and after the last point the segment will be continued
- The solver will not skip segments – the solver reduces internally the step width
- sca_eln::sca_vcc (voltage controlled capacitor), sca_eln::sca_vcr (voltage controlled resistor)

```
pwc_value = sca_create_pair_vector(-2.0,-2.0, -1.0,-0.4, 1.0,0.4, 2.0,5.0)
pwc_value = { {-2.0,-2.0}, {-1.0,-0.4}, {1.0,0.4}, {2.0,5.0} } //C++11
```

Piece Wise Linear

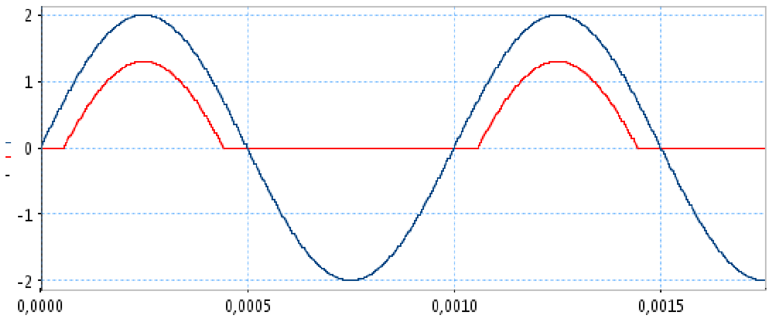
PWL modelling example – simple diode





roff = 1e6
ron = 1e-3
vth = 0.7

pwl_value = sca_util::sca_create_vector(0.0, 0.0,   p.vth, p.vth/p.roff,   p.vth+1.0, 1.0/p.ron)



Interactive Debug Feature / Extendable and unified Tracing

Motivation / History

- Enabling Interactive feature of Cadence Incisive for SystemC AMS
  - Cooperation: NXP-Cadence-COSEDA
  - -> vendor independent interface for SystemC AMS defined
  - **Can/Should be generalized for SystemC ?**

Interactive Debug Feature / Extendable and unifed Tracing

Feature

- Getting current (at SystemC time) value of ELN nodes, TDF and LSF signals, current traceable ELN modules and TDF trace variable
  - As string (using >> operator)
  - As object of the corresponding type
- Callback if a new value is available
- Force / release values for TDF signals (ELN/LSF would structural change the equation system)
- (Deposit value)

Interactive Debug Feature

Interface in sca_traceable_object 1/2

- virtual bool register_trace_callback(sca_trace_callback, void*);
  - typedef void (*sca_trace_callback)(void*);
  - Registers callback, which is executed if a new value has been produced, returns false if the corresponding object does not support tracing – the callback is called with the given void pointer as argument
- virtual const std::string& get_trace_value() const;
  - Returns the current value (as string) of the object at the current SystemC time (sc_core::sc_time_stamp())
  - Returns an empty string, if the object does not support tracing e.g. the signal is non-causal

Interactive Debug Feature

Interface in sca_traceable_object 2/2

May also useful for SystemC signals

- virtual bool force_value(const std::string&);
  - Forces the signal with the value given by the string, the value becomes valid with the start of the next cluster period, the force only influences the read's form the signal – the written and thus traced value remains
  - Returns false, if the value cannot be forced (the string cannot be converted or the object does not support value forcing like electrical elements)
- virtual void release_value();
  - Releases a forced value, the value is released with the start of the next cluster period

Interactive Debug Feature

Additional methods for sca_tdf::sca_signal<T> + ports

- const T& get_typed_trace_value() const;
  - Returns the current value of the object at the current SystemC time (sc_core::sc_time_stamp())
  - Will be used by const std::string get_trace_value()
- void force_typed_value(const T&);
  - Forces the signal with the value, the value becomes valid with the start of the next cluster period, the force only influences the read's form the signal – the written and thus traced value remains
  - The method is used by void force_value(const std::string&)

SystemC AMS Physical domain Modelling

Motivation

- Modelling the environment of integrated systems
- Running application scenarios
- Executable specification

SystemC AMS Physical domain Modelling

Requirements

- Physical domains can be modelled using the available SystemC / SystemC AMS classes
- Physical domain libraries can be generated by derivation using analogy relations
- If required you can build on top of SystemC/SystemC AMS unit systems (e.g. using the newest C++ feature)
- -> Missing: debug/tracing cannot recognize the units

SystemC AMS Physical domain Modelling

Discussion

- Discussion whether property annotation is sufficient
  - String properties for:
    - domain
    - Unit
- Virtual sc(a)_interface function (default implementation – empty string

Future thought

- Can we do more generalization?
  - E.g. enable customized static analyzes (budget (level, noise, nonlinearity, …) analysis, power, …)
    - Missing exploration capabilities – e.g. which port is connected to a port, which ports are connected to an interface, …
  - Combine TDF / SystemC on primitive level -> easier TLM->TDF interaction
  - Allow direct connection ELN<->LSF