SystemC-AMS for the design of complex Analog Mixed-Signal SoC's

Karsten Einwich
Fraunhofer IIS / EAS Dresden
Content

Introduction / Motivation SystemC-AMS

Some Code Snippets

Application Examples

- Wireline Communication Application
- Mixed Signal Embedded Core Application
- Automotive Application

Conclusion
SystemC-AMS is …

- Extension Library for SystemC, permitting modeling of Analog Mixed Signal behavior
- Prototype versions publicly available based on a Fraunhofer implementation
- Public version supports modeling of:
  - Non-conservative systems
  - Multi rate synchronous dataflow (SDF)
  - Linear electrical networks
  - Linear behavioral functions (linear transfer function numerator/denumerator and pole zero, state space),
  - Frequency domain simulation
  - Powerful trace functionality
- Experimental extensions available at Fraunhofer: Switched Capacitor solver, Nonlinear DAE solver with de-synchronization
SystemC-AMS is an extension of SystemC

- no changes to the SystemC implementation

- use of the same SystemC implementation

- no restrictions for the use of the SystemC language

- as far as possible, only LRM documented features used for the implementation of the library
Application Areas of SystemC-AMS

Modeling, Simulation and Verification for:

- Functional **complex** integrated systems (EAMS – Embedded Analogue Mixed Signal)
- Analogue **Mixed-Signal** systems / Heterogeneous systems
- **Specification** / Concept and System Engineering
- **System design**, development of a (“golden”) reference model
- Embedded **Software** development
- Next Layer (Driver) Software development
- **Customer model**, IP protection

- -> it is **not a replacement** of Verilog/VHDL-AMS or Spice
- -> compared to Matlab, Ptolemy, … SystemC-AMS supports architectural exploration/refinement and software integration
SystemC AMS History

- 2000 Fraunhofer and Infineon developed SystemC extension library mixsigc
- 2000 University Frankfurt and Continental Teves developed AVSL
- 2001 SystemC-AMS study group founded
- Analog extensions from different universities (South Hampton, University Ancona), Dataflow implementation from Shukla
- Ca. 2002 reimplementation of mixsigc to systemc-ams -> prototype of the study group
- 2006 official approval as OSCI AMSWG with Martin Barnasconi from NXP as chair
- 2008 White paper publication at DAC
- December 2008 DRAFT1 Language Reference Manual publicized
SystemC/SystemC-AMS specific Advantages

- Can be tailored and optimized for specific applications
- Support of customized methodologies and their combination
- The tradeoff between accuracy, simulation performance and modeling effort can be optimized for each system part by using the interoperability of an arbitrary number of Models of Computations (MoC)
- Encapsulation of subsystems which leads to scalability and modularity
- Easy software integration and powerful debug possibilities
- Full power of C++ available (e.g. language, libraries, encapsulation concepts)
- Easy IP protection by pre-compilation and integration into other tools and design flows via C interfaces
Modeling with multiple MoC

Linear electrical networks (conservative)

Discrete event (SystemC modules)

C – Code for target processor

Signaflow (non-conservative), frequency domain

Embedded linear analogue equations

Multi-rate static dataflow, frequency domain

**red:** Linear electrical networks
**magenta:** Linear DAE’s
**blue:** Static data flow
**green:** Pure SystemC

NASCUG 27-07-09 © Fraunhofer IIS/EAS, 2009
void sig_proc() {
    double tmp;  // high or low cut-off freq.
    if(fc_high.read())    tmp = ltf_1(b, a1, s, in.read());
    else                        tmp = ltf_0(b, a0, s, in.read());

    if (tmp >  v_max)    tmp =   v_max; //output voltage
    else if (tmp < –v_max)    tmp = –v_max; // limitation

    out.write(tmp); // assign output voltage to port
}

SCA_CTOR(prefi_ac)
{
    // default parameter values
    fc0 = 1.0e3;  fc1=1.0e5;  v_max = 1.0;
}

SCA_SDF_MODULE(prefi_ac)
{
    // signal import
    sca_sdf_in<double>  in;
    sca_sdf_out<double> out;  // signal output

    // control / DE signal from SystemC
    // (connected to sc_signal<bool>)
    sca_scsdf_in<bool> fc_high;

    double fc0, fc1;  // cut-off frequency
    double v_max;    // max. out value

    sca_ltf_nd ltf_0, ltf_1;  // filter equation obj.
    sca_vector<double> a0, a1, b;
    sca_vector<double> s;      // state vector

    void init() // filter coeffs for transfer function
    {
        const double r2pi = M_1_PI * 0.5;
        b(0) = 1.0;     a1(0) = a0(0) = 1.0;
        a0(1) = r2pi/fc0;  a1(1) = r2pi/fc1;
    }

    H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}

    prefi_ac
}

H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}
Frequency Domain Specification

void sig_proc()
{
    int x, y, i;
    for (i=0; i<64; ++i) {
        x = in.read(i);
        ... out.write(y);
    }
}

SCA_CTOR(ac_tx_comb)
{
    ...
}

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

SCA_SDF_MODULE(ac_tx_comb)
{
    sca_sdf_in<bool>         in;
    sca_sdf_out<sc_int<28> > out;

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

    void ac_sig_proc()
    {
        double k = 64.0; // decimation factor
        double n = 3.0; // order of comb filter

        sca_complex z1 = sca_ac_z(in.get_T().to_seconds(), -1);

        // complex transfer function:
        sca_complex h = pow((1.0 – pow(z1,k)) / (1.0 – z1), n);

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

    void sig_proc()
    {
        int x, y, i;
        for (i=0; i<64; ++i) {
            x = in.read(i);
            ... out.write(y);
        }
    }

    SCA_CTOR(ac_tx_comb)
    {
        ...
    }
}
SC_MODULE(prefiExternals) {
    // synchronous dataflow inport
    sca_sdf_in<double> kit;

    // connect with sc_signal<bool>
    sc_in<bool> fch;

    // electrical port
    sca_elec_port pout;

    // internal nodes declaration
    sca_elec_node w_it, w_sw;
    sca_elec_ref gnd;

    // component declarations
    sca_r *r_it, *r_prefi, *r_prefi2;
    sca_c *c_it;
    sca_sdf2i *i_t;
    sca_sc_rswitch *sw_prefi;
}

SC CTOR(prefiExternals) {
    i_t = new sca_sdf2i("i_t");
    i_t->p(gnd);
    i_t->n(w_it);
    i_t->ctrl(kit);

    r_it = new sca_r("r_it");
    r_it->p(gnd);
    r_it->n(w_it);
    r_it->value = 2.0e3;

    ...}
POTS - System

- Complete System functionality modeled
- All relevant analogue effects
- Digital parts “bittrue”, original code of embedded software
- Hundreds of simulation scenarios as regression tests available
- Simulation scenarios partially reused for silicon verification
- Embedded software debugged before silicon
Simulation Time for Vinetic 2CPE System

SystemC-AMS Simulation
- 2 channel including: SLIC, externals, AFE, DFE, ASDSP and part of Carmel FW
- 1 sec realtime → 1,5h simulation time

VHDL RTL
- 2 channel including: AFE, DFE, ASDSP, Carmel and Interfaces
- 1 sec realtime → 300h simulation time

Nano Sim (Fast CMOS simulator)
- 2 channel including: AFE top level
- 1 ms realtime → 15h simulation time

Titan Simulation
- 2 channel including: AFE top level
- 1 ms realtime → 500h simulation time

SystemC-AMS Simulation
- only one channel
- reduce sampling rate for analog blocks (used for FW simulation only)
- 1 sec realtime → 90 sec simulation time

Source: Gerhard Nössing Infineon COM
- Transient settling behavior
- Interaction Voice / Data transmission
- Training algorithm
- BER estimations
- Numerous of use scenarios
- Interaction of different lines
- Multi level simulation environment essential
Backlash Voice to data path
Mixed-Signal Embedded Core - SATA

- Serial ATA physical layer chip set for 3 / 6 / 8 GBit serial data transmission e.g. to/from hard discs
- Concept engineering model of transceiver /receiver including the pll’s
- Goal estimation of bit error rates, simulation of pll locking behavior
- Integration into as VHDL Model into Modelsim as reference model and stimuli generator for the digital components
SATA Project Results

♦ Simulation performance ~ 2h/ms = 6e6 clock cycle
♦ PLL settling / locking
♦ Equalizer coefficient adaption
♦ Estimation of BER
Simulation Performance Challenge for Automotive Applications

Temperature compensated TPMS sensor (MEMS)
- application (driving) 2 hrs. (1/t ~ 250µHz)
- battery voltage update rate ~ 20mHz
- temperature update rate ~ 200mHz
- pressure update rate (wakeup) ~ 2Hz
- analog processing rate ~ 1MHz
- digital processing rate ~ 4Mhz

Source: Wolfgang Scherr Infineon AIM
Automotive Sensor Projects

- System level model including the embedded processor on a cycle accurate level
- Switched capacitor converter
- Diagnose modes, offset calibration, temperature dependencies, noise, manchester interface, synchronization via supply voltage, ...
- Original code of embedded software
- TLM based modeling for processor communication
- IP protected customer model as Matlab/Simulink Module (mex – dll)
- Simulation performance ~ 10min /sec
Window lifter system

+V_s

LIN-Bus

Wake-Up Inputs

Relay Driver with Embedded MCU & LIN

Temp. / +V_s Supervision

Supply

Dual-Hall Sensor

Relay

Relay Contact

GND

NASCUG 27-07-09
© Fraunhofer IIS/EAS, 2009
Window lifter simulation results

A huge amount of digital HW/SW ...

Magnetic flux → digital sensor out

Electronics: voltages/currents → torque

Mechanics: position, torque, forces, ...
Conclusion

- SystemC together with the extension SystemC AMS is suitable for creating executable specification, virtual prototypes and architectural level models for EAMS systems

- An experimental prototype can be downloaded under: www.systemc-ams.org (not compatible with the DRAFT 1 standard)

- SystemC AMS DRAFT 1 standard is public available: www.systemc.org

- OSCI SystemC AMS 1.0 standard is expected in December 2009

- Information of the Fraunhofer SystemC AMS activities and documentation: www.systemc-ams.eas.iis.fraunhofer.de