High-Performance Mixed-Signal ESL Design of a Magneto Resistive Sensor Application

Martin Barnasconi, NXP Semiconductors
Sumit Adhikari, NXP Semiconductors

accellera SYSTEMS INITIATIVE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

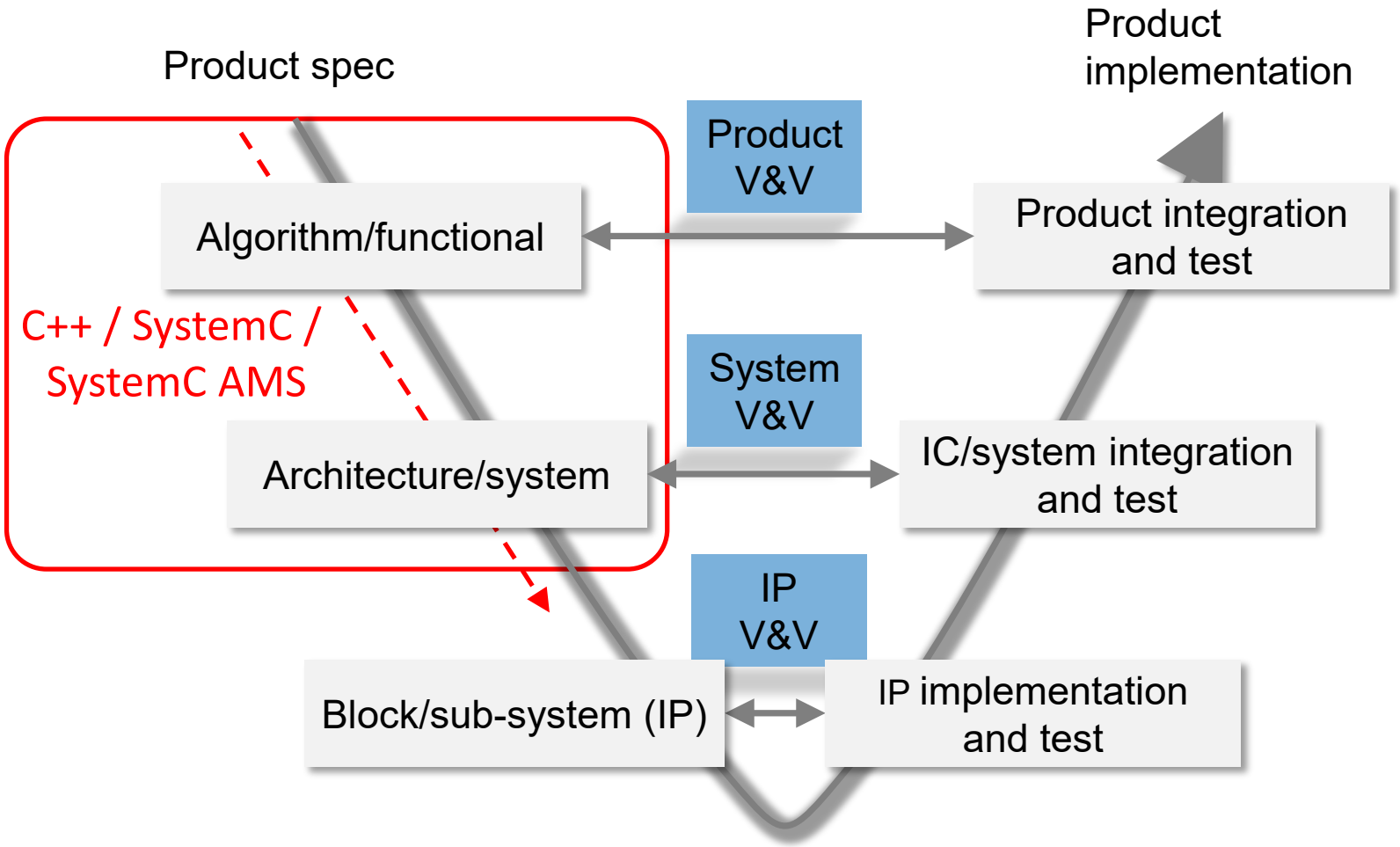
Outline

- Introduction
- SystemC AMS extensions
- ESL design refinement methodology for mixed-signal systems
- Application example: Magneto Resistive (MR) Sensor
- SystemC AMS model of a programmable gain amplifier
- Simulation results
- Conclusions

Introduction

- Most virtual prototypes only focus on digital HW/SW system components – AMS functionality is neglected
- However, today's embedded systems contain AMS and RF components which tightly interact with the HW/SW system
- Advanced modeling approaches needed to include AMS behavior in the architecture design phase
- Application of a model-based ESL design refinement flow based on SystemC and SystemC AMS

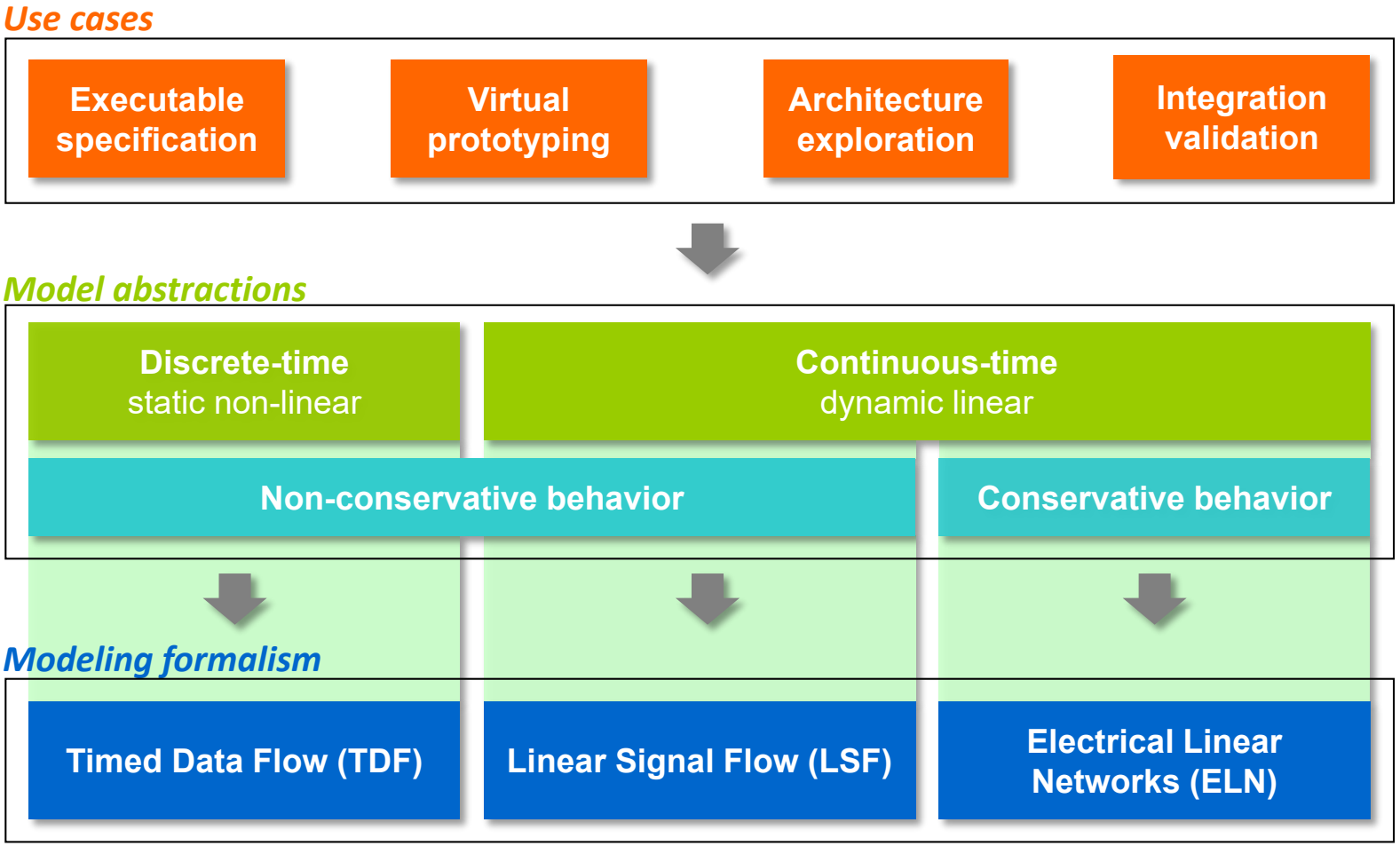
V-model and ESL refinement flow



SystemC AMS extensions

- SystemC AMS extensions defined in IEEE Std 1666.1
- SystemC AMS defines new models of computation for efficient modeling of analog and RF functionality
  - Timed Data Flow to describe non-conservative signal processing functionality, including multi-rate systems
  - Linear Signal Flow and Electrical Linear Networks to describe conservative continuous-time descriptions
- Proof-of-concept implementation of SystemC AMS is available under Apache 2.0 license
- SystemC AMS models and simulator can be integrated in commercial EDA tools and flows

SystemC AMS model abstractions and modeling formalisms

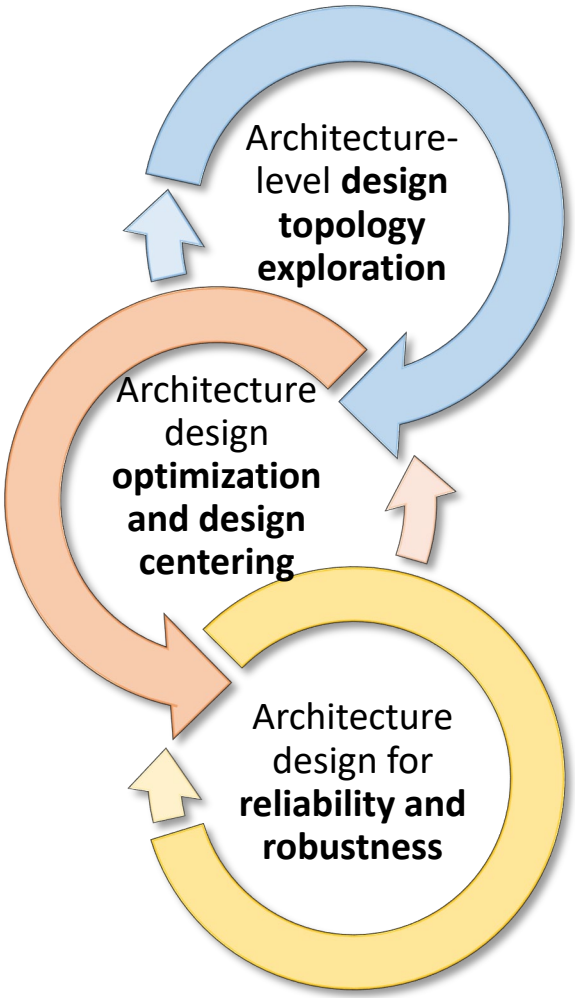


Source: Accellera Systems Initiative

Modeling in SystemC/-AMS

SystemC	*SystemC AMS*
State machines	Signal sensing, amplification and mixing
Digital protocols (e.g. I2C, SPI, …)	A/D and D/A conversion
Digital filters	Digital filters
DSP algorithms/functions	DSP algorithms/functions
Processor instruction set model	Power supply ripple
Register interfaces	Continuous-time filtering
Calibration and control algorithms	Noise (white noise, 1/f noise, …)
Transaction-level (TLM) communication	Non-linearities
FIFO's	DC offsets
…	Saturation effects
	Impedance mismatches
	Small-signal (AC) characteristics
	Charge/discharge effects
	…

ESL design refinement methodology

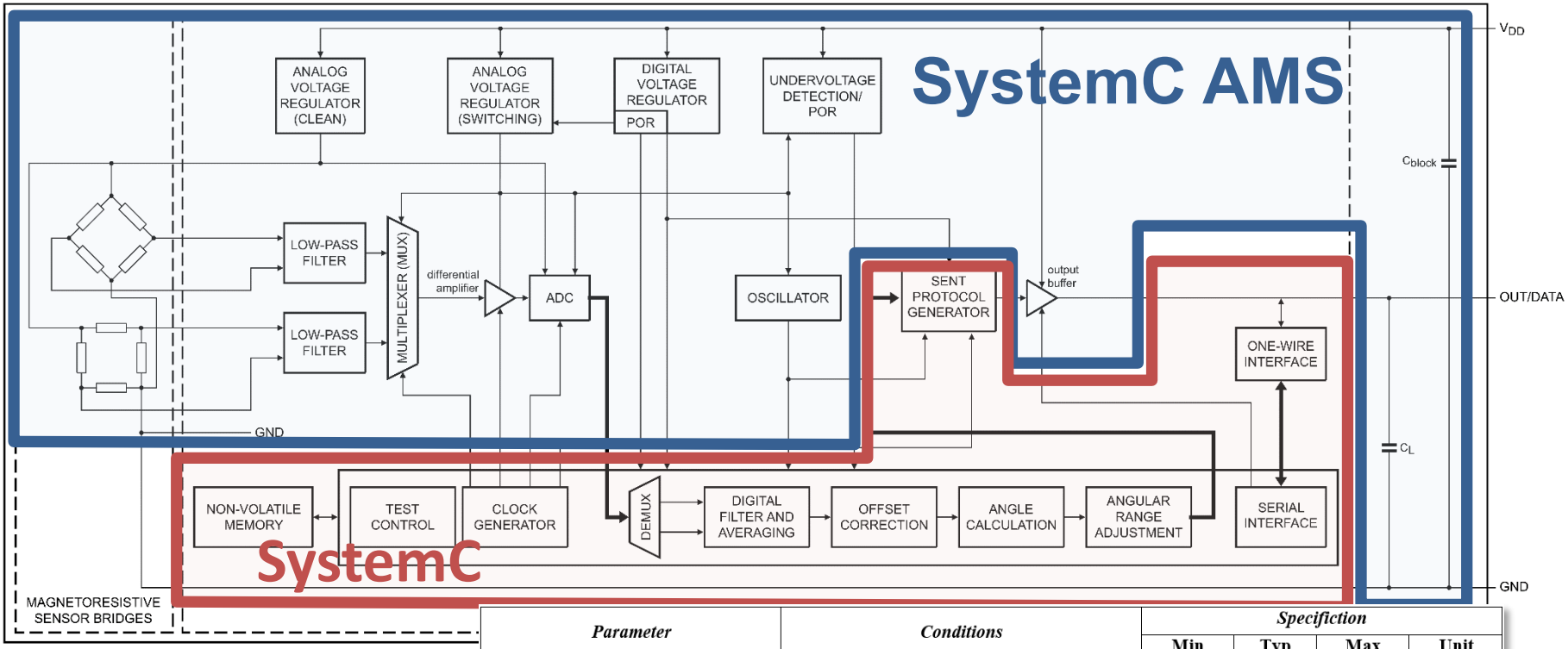


- Implement the architecture in SystemC AMS and SystemC
- Design refinement using transfer functions, switches, accurate regulation and control loops

- Modeling of passive components, thermal noise, 1/f noise and non-idealities
- Include characterized behavior in the system-level model, incl. PVT variations and dependencies

- Monte Carlo (MC) simulations using statistical library
- Apply extensive failure injection and analysis
- If MC or failure analysis fails, re-optimize system architecture

Application example: Magneto Resistive (MR) Sensor



Parameter	*Conditions*	*Specifiction*			
		Min	**Typ**	**Max**	**Unit**
Angle resolution				0.05	deg
Maximum angle	programmable angular range	6		180	deg
Operating temparature		-40		+160	°C
Linerarity error	temperature range -40 °C to +160 °C	-1		1	deg
Angular error	temperature range -40 °C to +160 °C	-1.2		1.2	deg

Programmable Gain Amplifier (PGA) in SystemC AMS

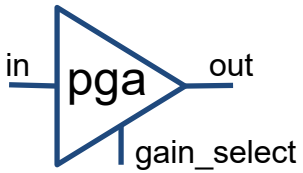
```
class pga: public sca_tdf::sca_module
{
 public:
  sca_tdf::sca_in<double>       in;
  sca_tdf::sca_out<double>      out;
  sca_tdf::sca_de::sca_in<bool> gain_select;

  pga( sc_core::sc_module_name nm, double f3dB1_,
       double dcgain1_, double f3dB2_, double dcgain2_ )
  : in("in"), out("out"), gain_select("gain_select"),
    f3dB1(f3dB1_), dcgain1(dcgain1_), f3dB2(f3dB2_),
    dcgain2(dcgain2_) {}

  void initialize() // initialize numerator and denominator
  {
    num1(0) = dcgain1;  den1(0) = 1.0;
    den1(1) = 1.0 / ( 2.0 * M_PI * f3dB1 );
    num2(0) = dcgain2;  den2(0) = 1.0;
    den2(1) = 1.0 / ( 2.0 * M_PI * f3dB2 );
  }

  void set_attributes()
  {
    request_next_activation( gain_select.default_event() );
    does_attribute_changes();
    accept_attribute_changes();
  }
```

Enable dynamic time steps

```
  void processing() // time-domain implementation
  {
    double pga_out;
    if( gain_select.read() )
      pga_out = ltf_nd1( num1, den1, state, in.read(), 1.0 );
    else
      pga_out = ltf_nd2( num2, den2, state, in.read(), 1.0 );
    out.write(pga_out);
  }

  void change_attributes()
  {
    request_next_activation( gain_select.default_event() );
  }

 private:
  sca_tdf::sca_ltf_nd ltf_nd1, ltf_nd2;
  sca_util::sca_vector<double> num1, num2, den1, den2;
  sca_util::sca_vector<double> state;
  double f3dB1, f3dB2;
  double dcgain1, dcgain2;
};
```

Continuous-time filter

Dynamic time step when gain input changes



Programmable Gain Amplifier (PGA) in SystemC AMS – incl noise

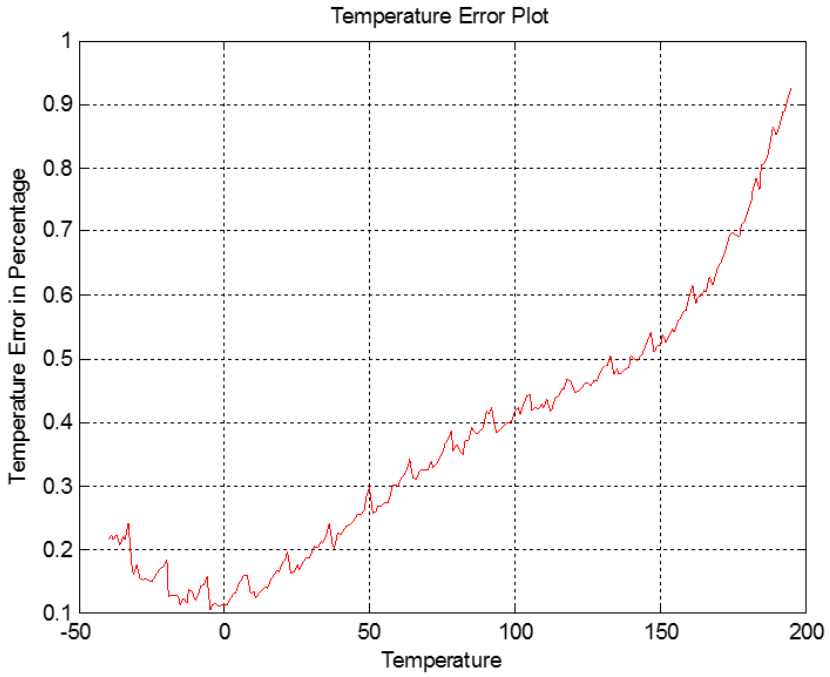
```
class pga_noise: public sca_tdf::sca_module
{
 public:
  sca_tdf::sca_in<double>       in;
  sca_tdf::sca_out<double>      out;
  sca_tdf::sca_de::sca_in<bool> gain_select;

  pga_noise( sc_core::sc_module_name nm, double f3dB1_,
       double dcgain1_, double f3dB2_, double dcgain2_ )
  : in("in"), out("out"), gain_select("gain_select"),
    f3dB1(f3dB1_), dcgain1(dcgain1_), f3dB2(f3dB2_),
    dcgain2(dcgain2_) {}

  void initialize() // initialize numerator and denominator
  {
    num1(0) = dcgain1;  den1(0) = 1.0;
    den1(1) = 1.0 / ( 2.0 * M_PI * f3dB1 );
    num2(0) = dcgain2;  den2(0) = 1.0;
    den2(1) = 1.0 / ( 2.0 * M_PI * f3dB2 );
    Rth = 1e-6; Temp = 290.0; k = 1.38064852e-23;
    fs = 8e6; mu = std::sqrt(4*k*Temp*Rth*fs/2.0);
  }

  void set_attributes()
  {
    request_next_activation( gain_select.default_event() );
    does_attribute_changes();
    accept_attribute_changes();
  }
```

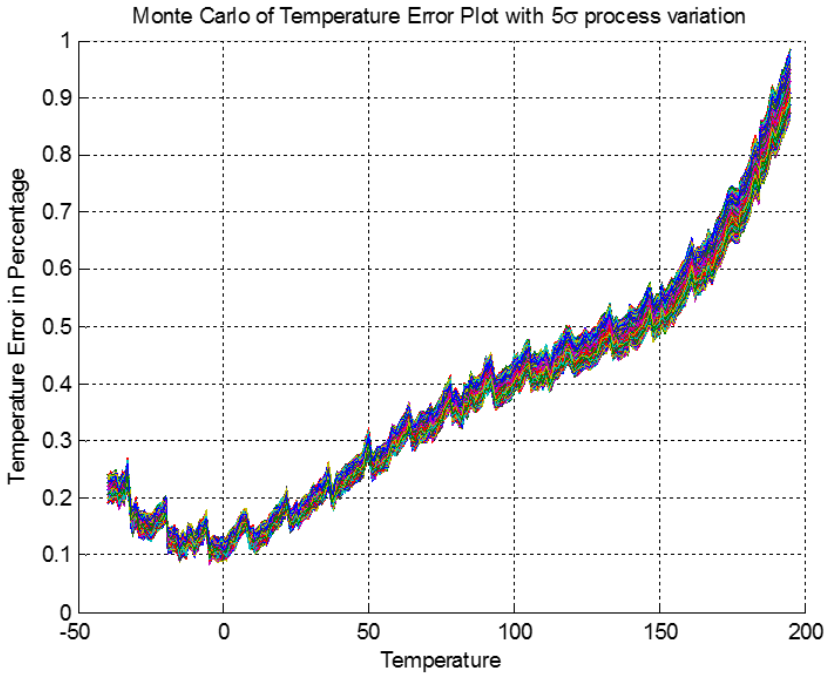
Noise (4kTBR)

```
void processing() // time-domain implementation
  {
    double pga_out;
    if( gain_select.read() )
      pga_out = ltf_nd1( num1, den1, state, in.read() +
                mu*r.get_value(), , 1.0 );
    else
      pga_out = ltf_nd2( num2, den2, state, in.read() +
                mu*r.get_value(), , 1.0 );
    out.write(pga_out);
  }

  void ch
  {
    reque                                    lt_event() );
  }

 private:
  sca_tdf
  sca_uti                                    l, den2;
  sca_uti
  double
  double
};
```

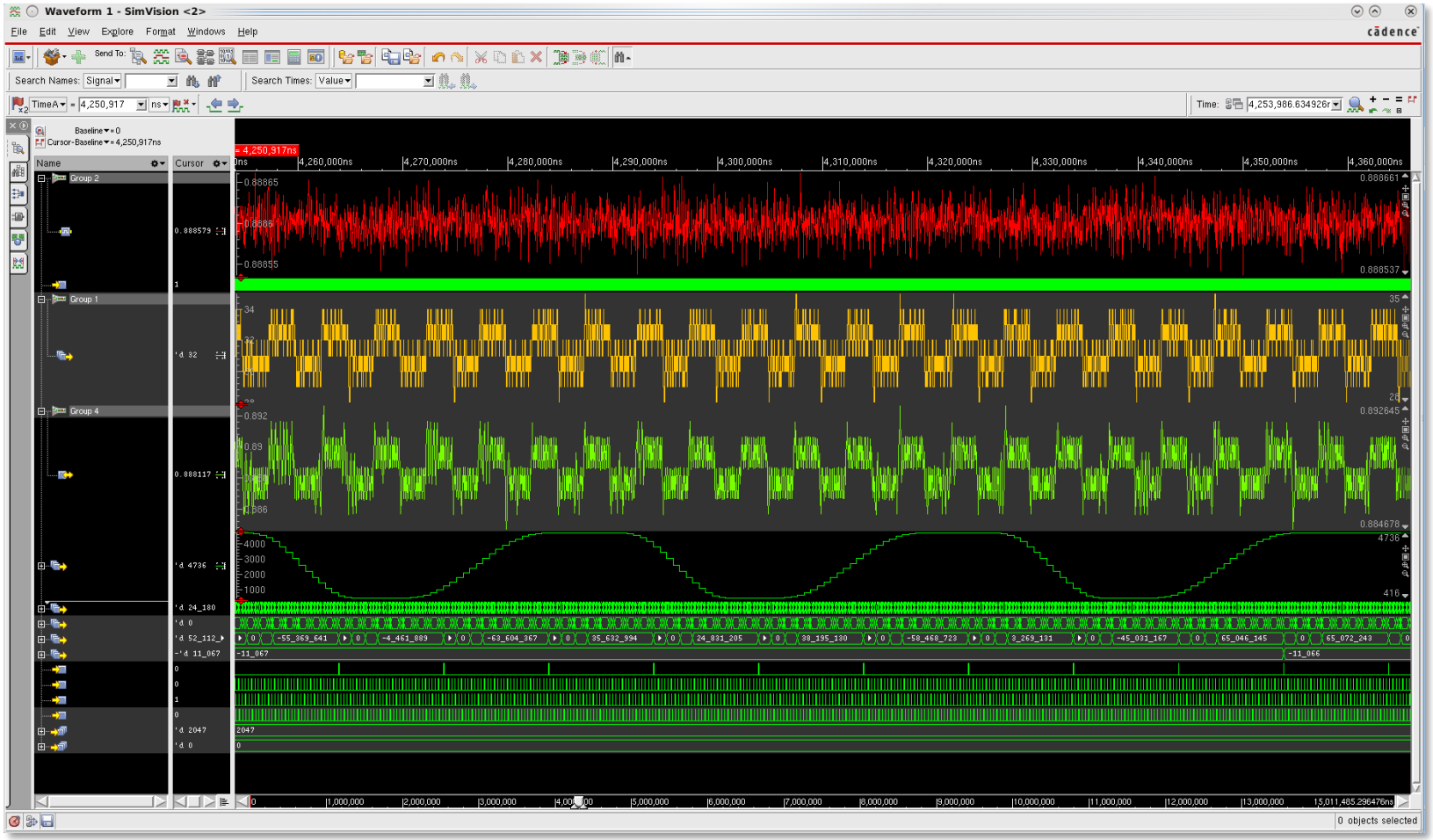
```
// file: gaussian_rnd.h
// Helper class to calculate a
// Gaussian random number using
// the GNU Scientific Library

#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

template < typename P >
class gaussian_rnd
{
 public:
  gaussian_rnd(...);
  P get_value();

 private:
  const gsl_rng_type* T;
  gsl_rng*            r;
};
```

Typical temperature error of the MR Sensor





Mixed-signal transient simulation



Typical simulation speed for each design refinement step

Design refinement step	Simulation speed	
	Simulation time (sec)	Wall clock time (sec)
A. Architecture-level design topology exploration	1	~60
B. Architecture design optimization and design centering	1	~500
C. Architecture design for reliability and robustness (using Monte Carlo)	1ms	~5 (per MC run)

Conclusions

- An mixed-signal ESL design refinement methodology has been developed using SystemC and SystemC AMS
- SystemC AMS offers unique capabilities to accurately describe analog/mixed-signal behavior, while keeping a high simulation speed
- The ESL design refinement flow has been successfully applied in the concept design phase of a Magneto Resistive (MR) Sensor
- The use of libraries such as Boost and the GNU Scientific Library (GSL) facilitated analog performance and Monte-Carlo analysis of the mixed-signal sensor application

Acknowledgment

- Karsten Einwich – COSEDA Technologies GmbH
- Vincent Motel – Cadence Design Systems

Questions