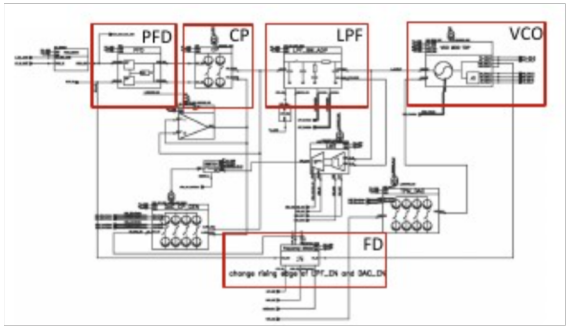
SystemC AMS and upcoming frameworks for the free design of Analog/Mixed-Signal Systems

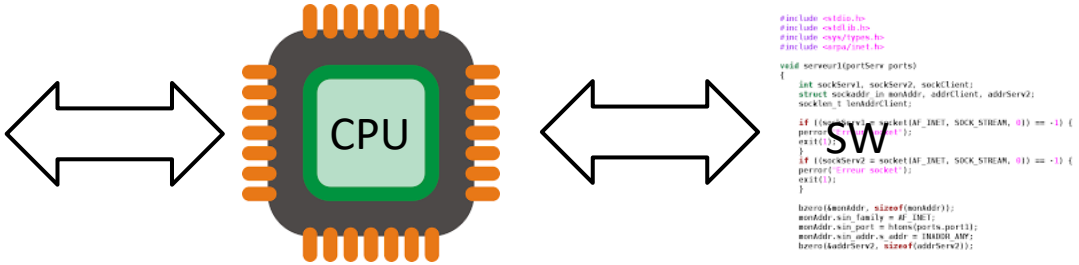
Christoph Grimm, Carna Zivkovic

TU Kaiserslautern
Chair of Cyber-Physical Systems

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

SYSTEMC™: free eco-system for HW/SW/AMS systems





SystemC ; IEEE Std. 1666

- Free simulator for HW/SW Systems; C++ based; C++ based framework
- https://accellera.org/downloads/standards/systemc

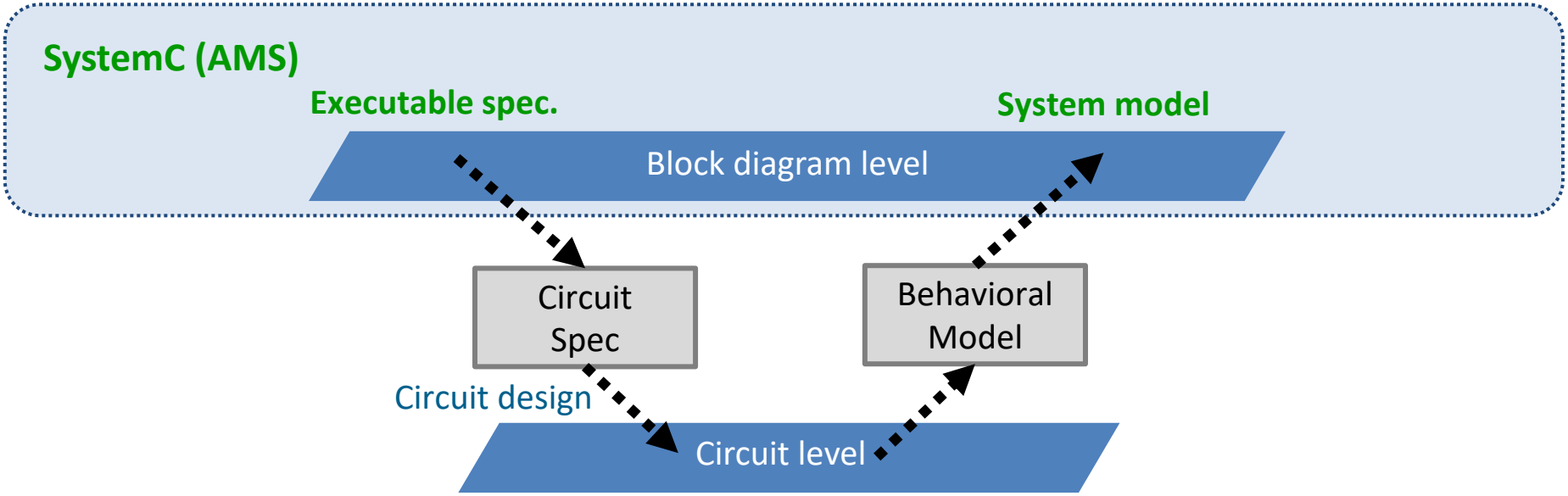
SystemC AMS; IEEE Std. 1666.1-2016

- Support DSP methods, analog
- https://www.coseda-tech.com/systemc-ams-proof-of-concept

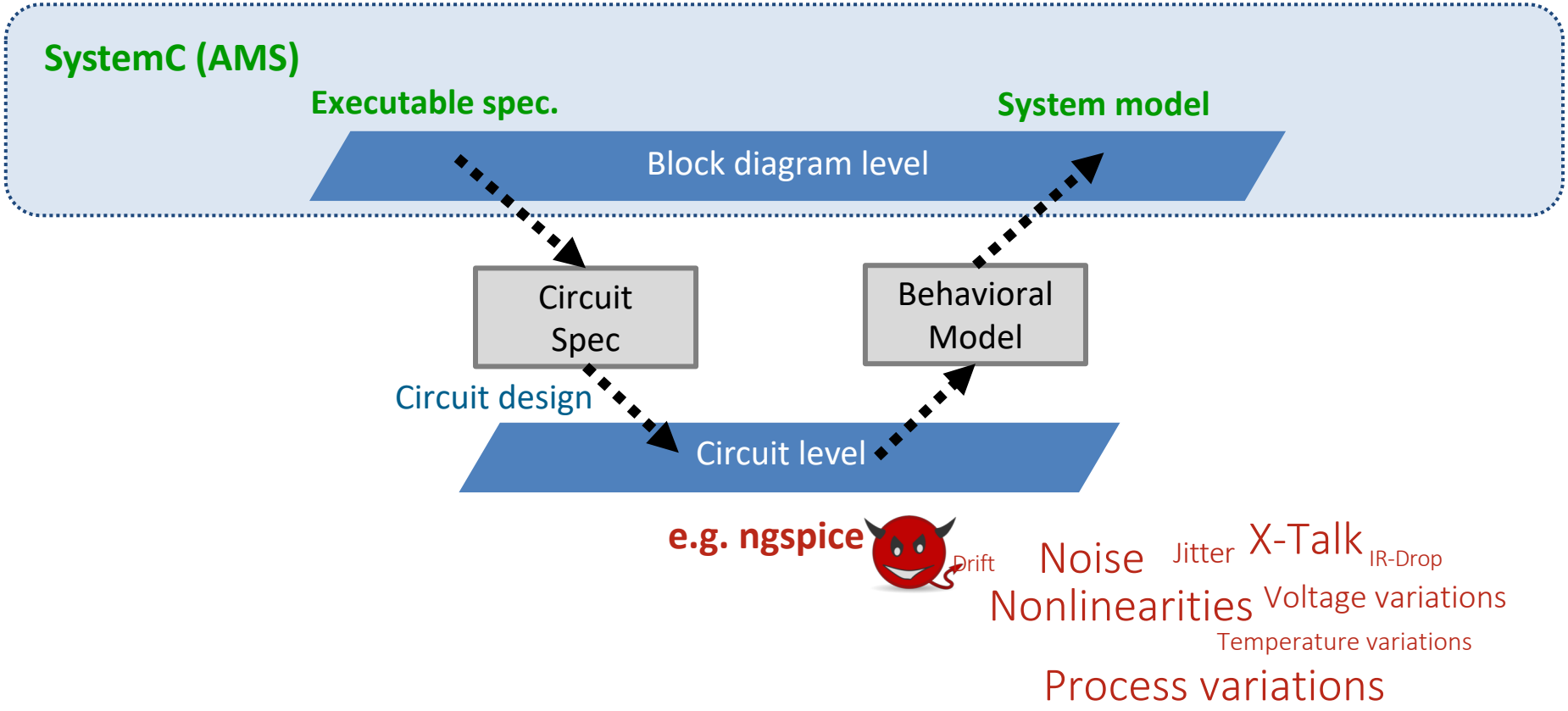
AADDlib

- Symbolic execution/simulation of e.g. C++ 11, SystemC, SystemC AMS
- https://github.com/TUK-CPS/AADD

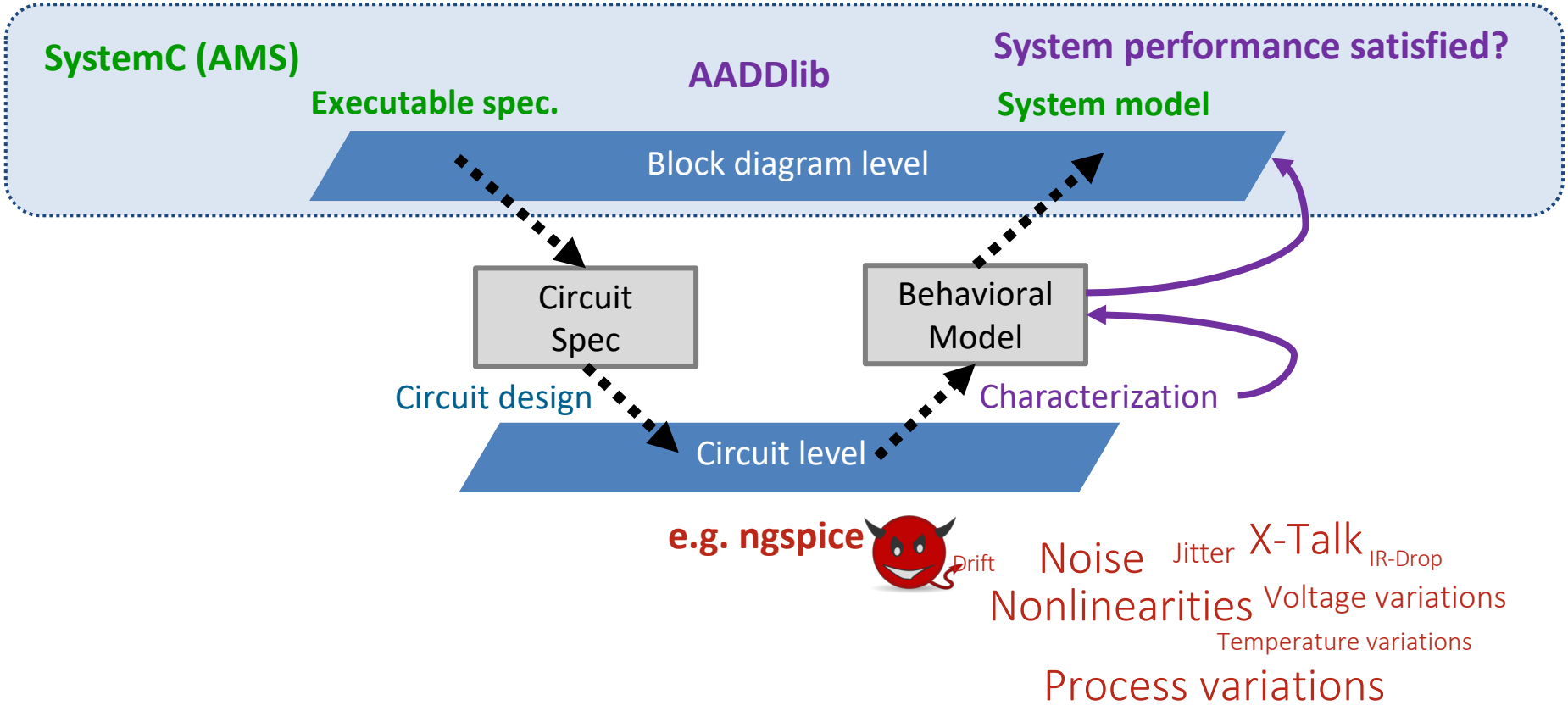
SystemC (AMS), SPICE and AADDlib in Development Process



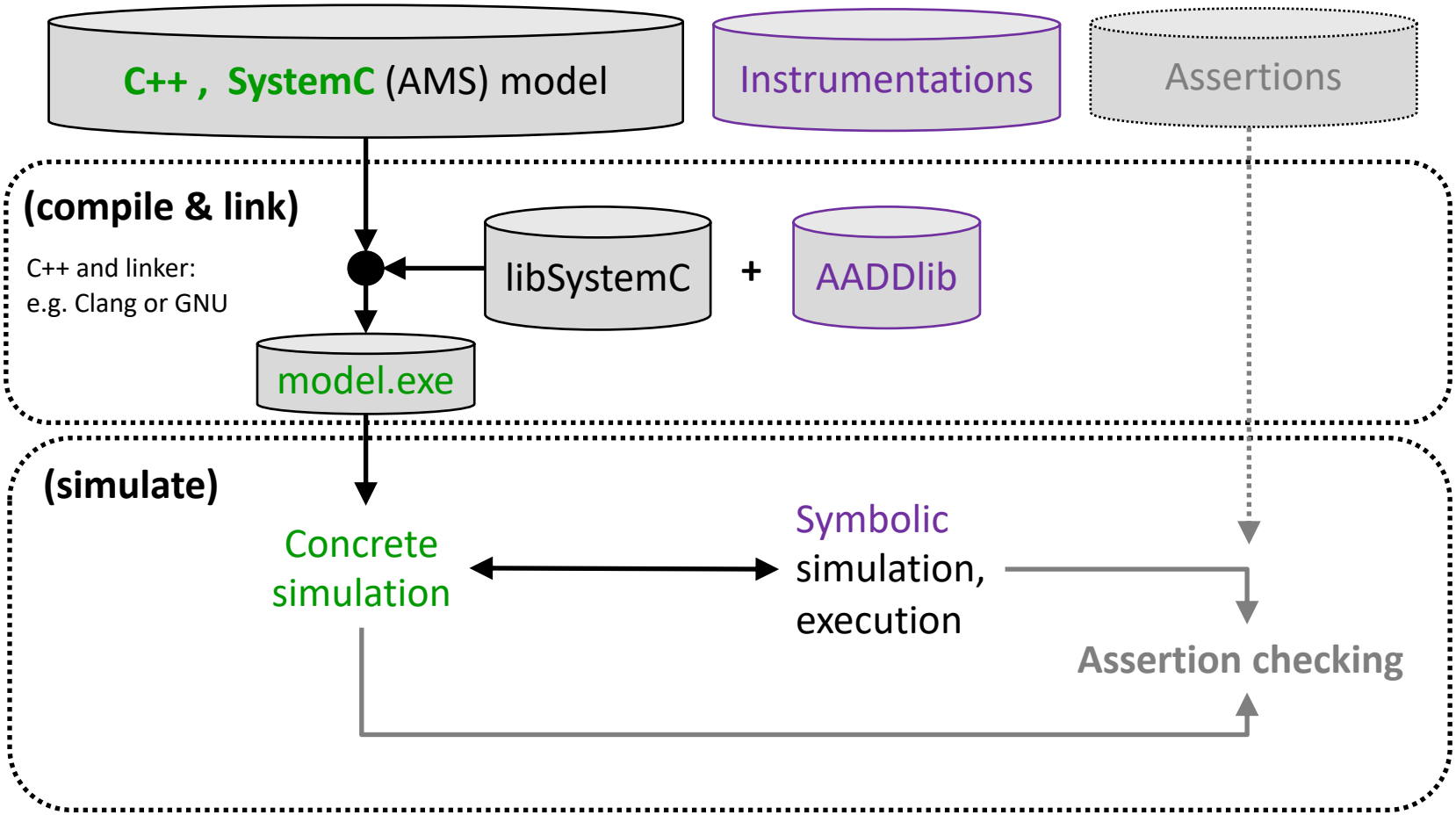
SystemC (AMS), SPICE and AADDlib in Development Process



SystemC (AMS), SPICE and AADDlib in Development Process



Overview of Tool Integration



SystemC AMS and upcoming frameworks for the free design of Analog/Mixed-Signal Systems

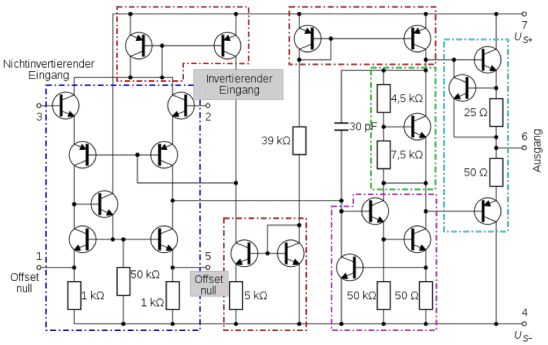
1) SystemC AMS

2) Performance verification with AADDlib

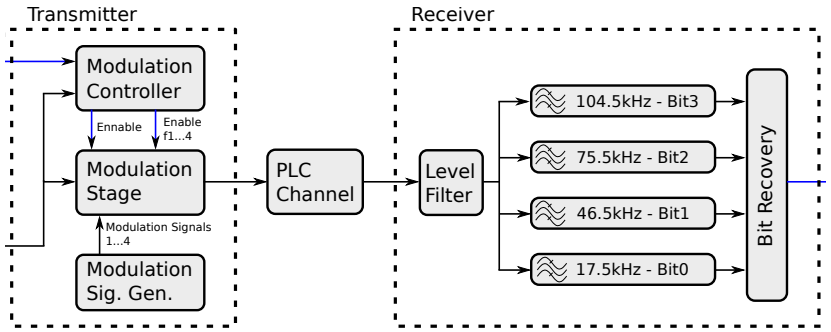
3) Case study

4) Outlook

SystemC AMS: Block Diagram Level, but not Circuit Level



Circuit schematic, e.g. ***SPICE***



Block diagram, e.g. ***Matlab/Simulink, SystemC AMS***

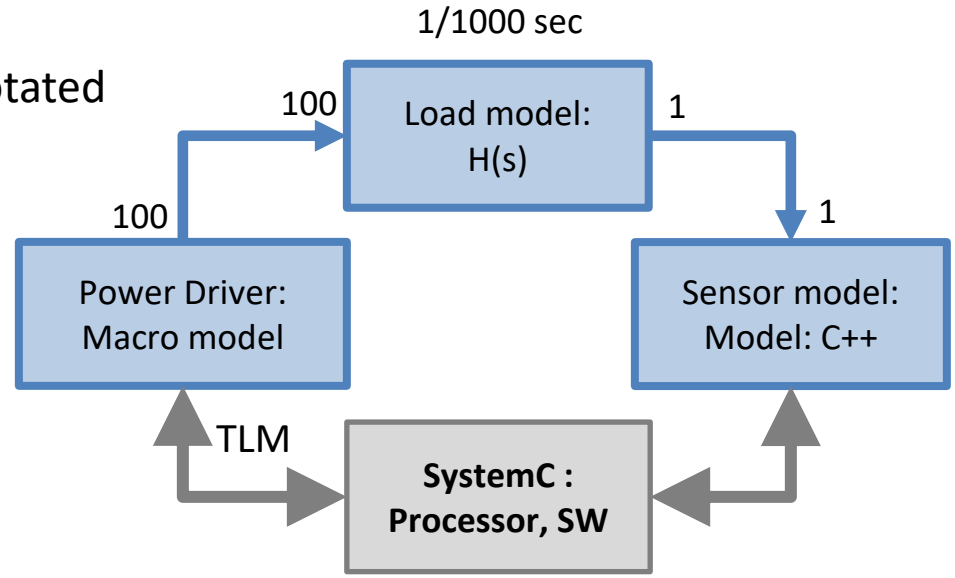
SystemC AMS: simulation of AMS systems at **functional** and **block diagram level**

- **Directed interaction** of functional blocks
- Blocks are either **ideal functions** or **behavioral models**, no circuit-level models

Block Diagrams in SystemC AMS

TIMED DATA FLOW (TDF):

- Functions of blocks in block diagrams are processed **in data flow's direction**.
- Ports may have different rates
  - Static data-flow + timing annotated
  - Scheduling before simulation



LINEAR SIGNAL FLOW (LSF)

- Structure sets up system of linear equations

Specification of a Block's Function

By a (static) function (TDF only)

- E.g. mixer:

```
y = rf_in*carrier;
```

By a transfer function

- E.g. filter:

```
y = ltf_1(nom, denom, x);
```

By a macro model (ELN)

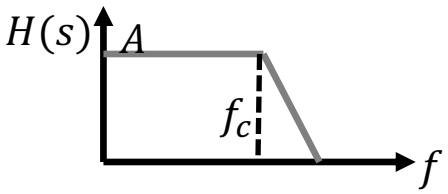
- E.g. power driver

```
(SPICE-like Structure of switches,
controlled sources, R L C)
```

… or by arbitrary C++ code, maybe mixing all the above options.

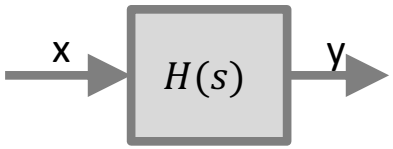
Simple Example: Executable Specification of a Filter

Function of a "filter" is described by a **transfer function, e.g.**



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

SystemC AMS allows us to specify the behavior of a block by C++ -Code




```
sca_module("LP") {
  sca_in<double> x;
  sca_out<double> y;
  (…)
  void sca_processing() {
    A(0) = 1.0;
    denom(0)=1.0; denom(1)=r2pi/fc;
    y = ltf_block(A, denom, x);
  }
}
```

SystemC AMS and upcoming frameworks for the free design of Analog/Mixed-Signal Systems

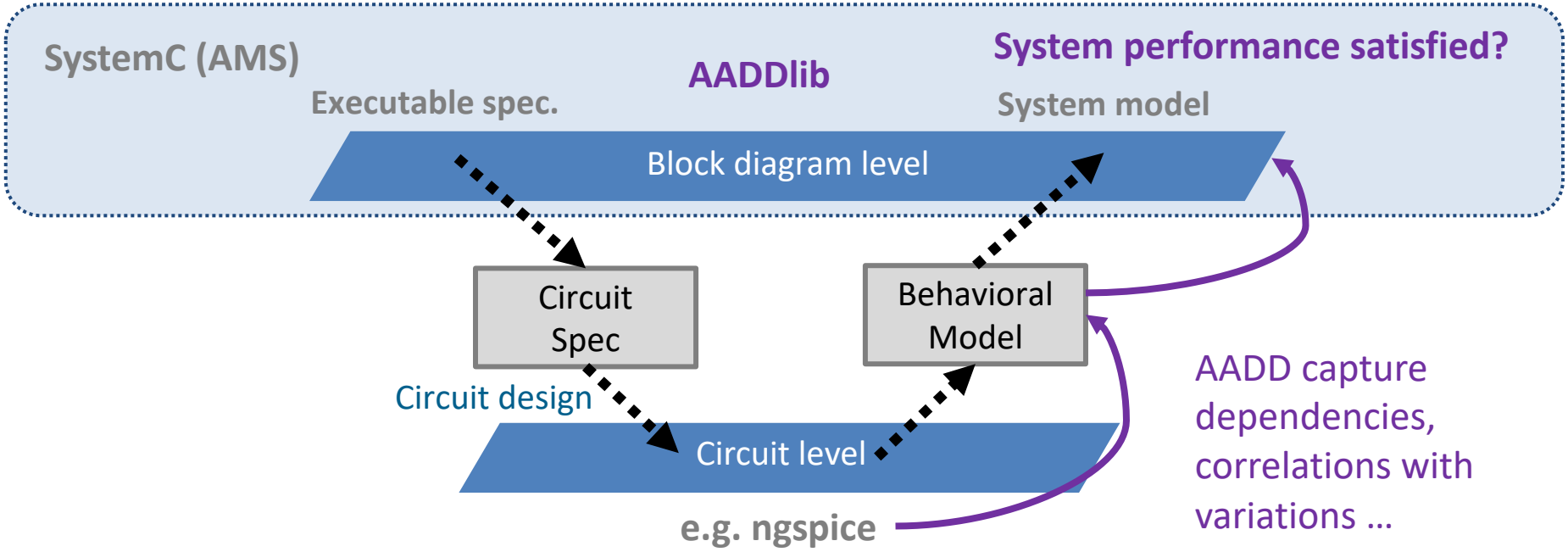
1) SystemC AMS

2) Performance verification with AADDlib

3) Case study

4) Outlook

SystemC (AMS), SPICE and AADDlib in Development Process



Drift Noise Jitter X-Talk IR-Drop
Nonlinearities Voltage variations
Temperature variations
Process variations

Refined model of a filter … more realistic!

Relevant properties of filter for the system are amplification, corner frequency, noise, limitation, …

- E.g. corner frequency f_c, amplification A can be modeled by a range
- In specification: means allowed range
- In behavioral model: models uncertainties, impact of PVT variations

```
void processing()
  (...)
  A(0) = 1.0 + dA;            // Range of possible A
  denom(0)=1.0;
  denom(1)=r2pi/(fc + df);    // Range of possible fc

  if (x > 5.0) then x = 5.0; // limitation at input
  x += noise(3);              // assume some noise

  y = ltf_block(A, denom, x);

  if y > 5.0 then y = 5.0;    // limitation at output
```

Affine forms, affine arithmetic [Andrade, Stolfi, 1997]

```
A(0) = 1.0 +/- dA;        // Range of possible A
     = 1.0 +   dA * ε_i   // By affine form
```

Affine forms represent linear dependencies in symbolic way:

$$\tilde{x} := x_0 + \sum_{i=1}^{n} x_i \varepsilon_i \quad \text{with noise symbols } \varepsilon_i \in [-1,1]$$

- x_0 is *center value*,
- x_i are *partial deviations*, models sensitivity to variations e.g. P, V or T.
- Nonlinear dependencies handled by safe inclusion with x_i not used otherwise. *e.g. Chebychev approximation*

Limitations:

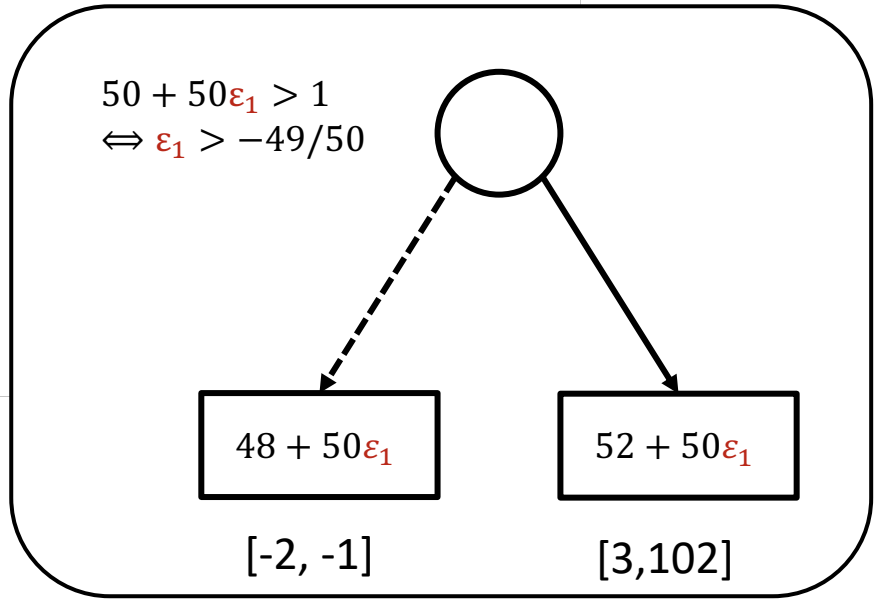
- *Functions with local extrema, discontinuous functions, e.g.*

```
if y > 5.0 then y = 5.0;   // limitation at output
```

Basic Example: Use of libAADD in C++ 11

```
if y > 5.0 then y = 5.0;   // limitation at output
```

```
#include "aadd.h"

doubleS a;  // Data types with S are computed symbolically.

int main()
{
    a = doubleS(0,100); // a takes double value from range [0,100]

    ifS(a > 1)          // symbolic cond. and iteration statements
       a = a + 2;
    elseS
       a = a - 2;
    endS;

    cout << "a is: " << endl;
    cout << a;

}
```



SystemC AMS and upcoming frameworks for the free design of Analog/Mixed-Signal Systems

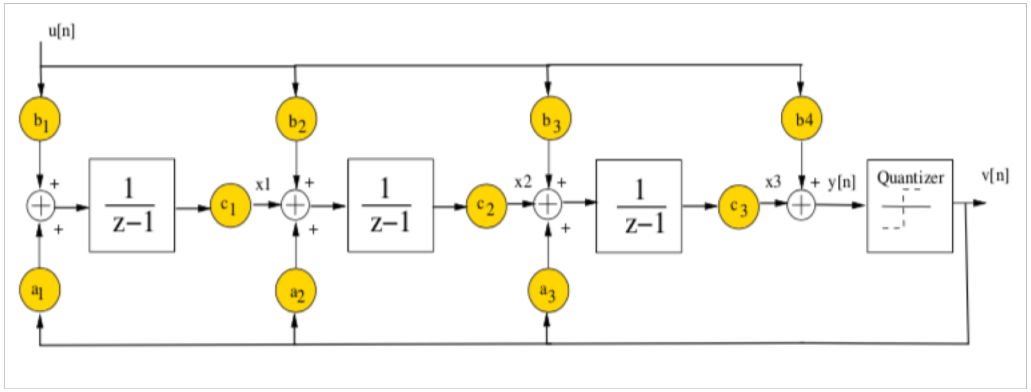
1) SystemC AMS

2) Performance verification with AADDlib

3) Case study

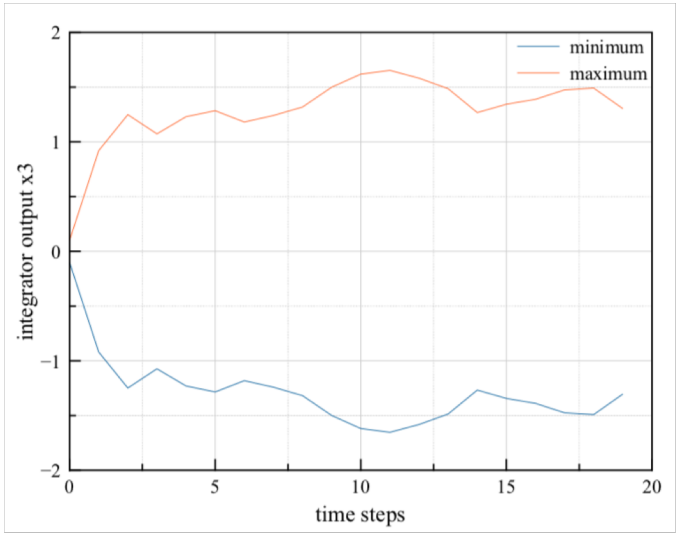
4) Outlook

Benchmark: 3rd order sigma-delta modulator



Initial input and integrator values
$x_1 \in [-0.1, 0.1]$
$x_2 \in [-0.1, 0.1]$
$x_3 \in [-0.1, 0.1]$
$u \in [-0.5, 0.5]$

```
SCA_TDF_MODULE(quantizer) {
  sca_tdf::sca_in<doubleS> y;
  sca_tdf::sca_out<doubleS> v;
  void processing() {
    ifS(y>0) v=1; elseS v=-1; endS
  }
  quantizer(sc_module_name nm){}
}
```



Comparison with State of the Art

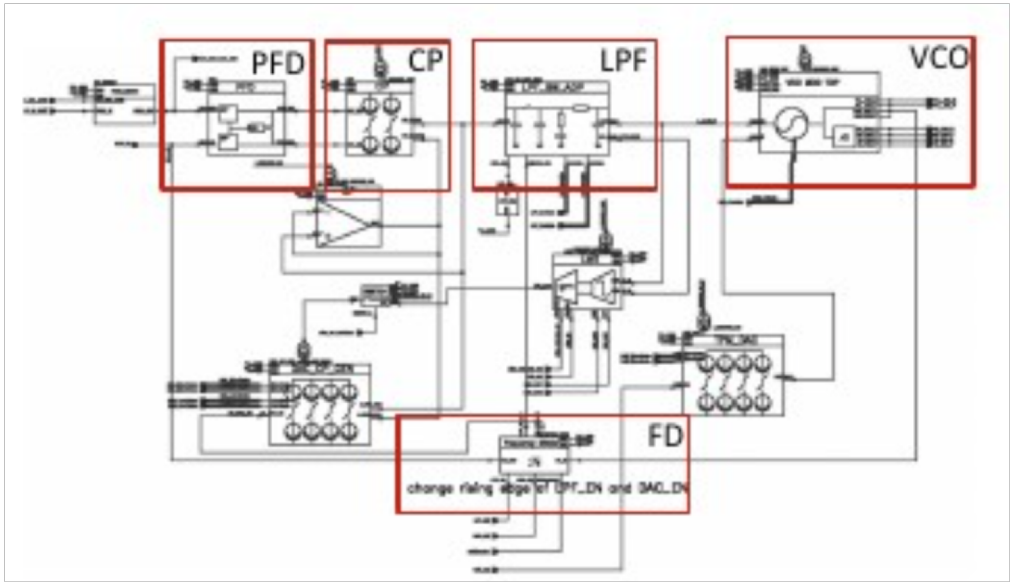
Initial input and integrator values	[Sammane 2007] d/dt [Dang 2004]	AADDlib incl. Property 1	Speedup
$x_1 \in [0.012, 0.013]$ $x_2 \in [0.01, 0.02]$ $x_3 \in [0.8, 0.82]$ u=0.54=const	**31 s** for N=38 time steps [Sammane 2007]	**1.93 s** for 100 time steps	> 16
$x_1 \in [-0.1\ 0.1]$ $x_2 \in [-0.1, 0.1]$ $x_3 \in [-0.1, 0.1]$ u=[-0.5, 0.5]	**2 h** for N=30 time steps (d/dt tool) [Dang 2004]	**525 s** for 30 time steps	> 13

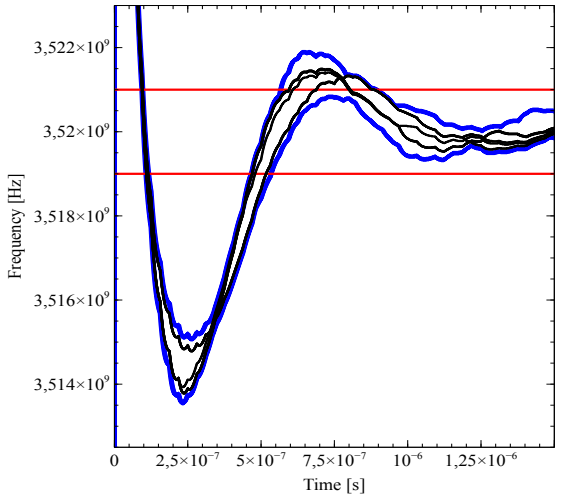
Property 1: $\mathrm{G}(x_i, -2, 2)$ with $i \in \{1, 2, 3\}$ for x_3 was satisfied in both cases.
Property 2: $\mathrm{G}(y, -2, 2)$ failed for second set of initial conditions, simulation was stopped.

Does it scale for “real” designs?
Larger case study …

Cadence-design of dual-charge-pump PLL of ZigBee transceiver

- 8434 lines of C++ Code in SystemC by netlister
- 7-12 parameters with symbolic variations
- 40.000-120.000 time steps
- 4-11 min. for symbolic simulation, assertion checking





Blue: bounds of AADD stream

Black: corner case simulations

SystemC AMS and upcoming frameworks for the free design of Analog/Mixed-Signal Systems

1) SystemC AMS

2) Performance verification with AADDlib

3) Case study

4) Outlook

Conclusion

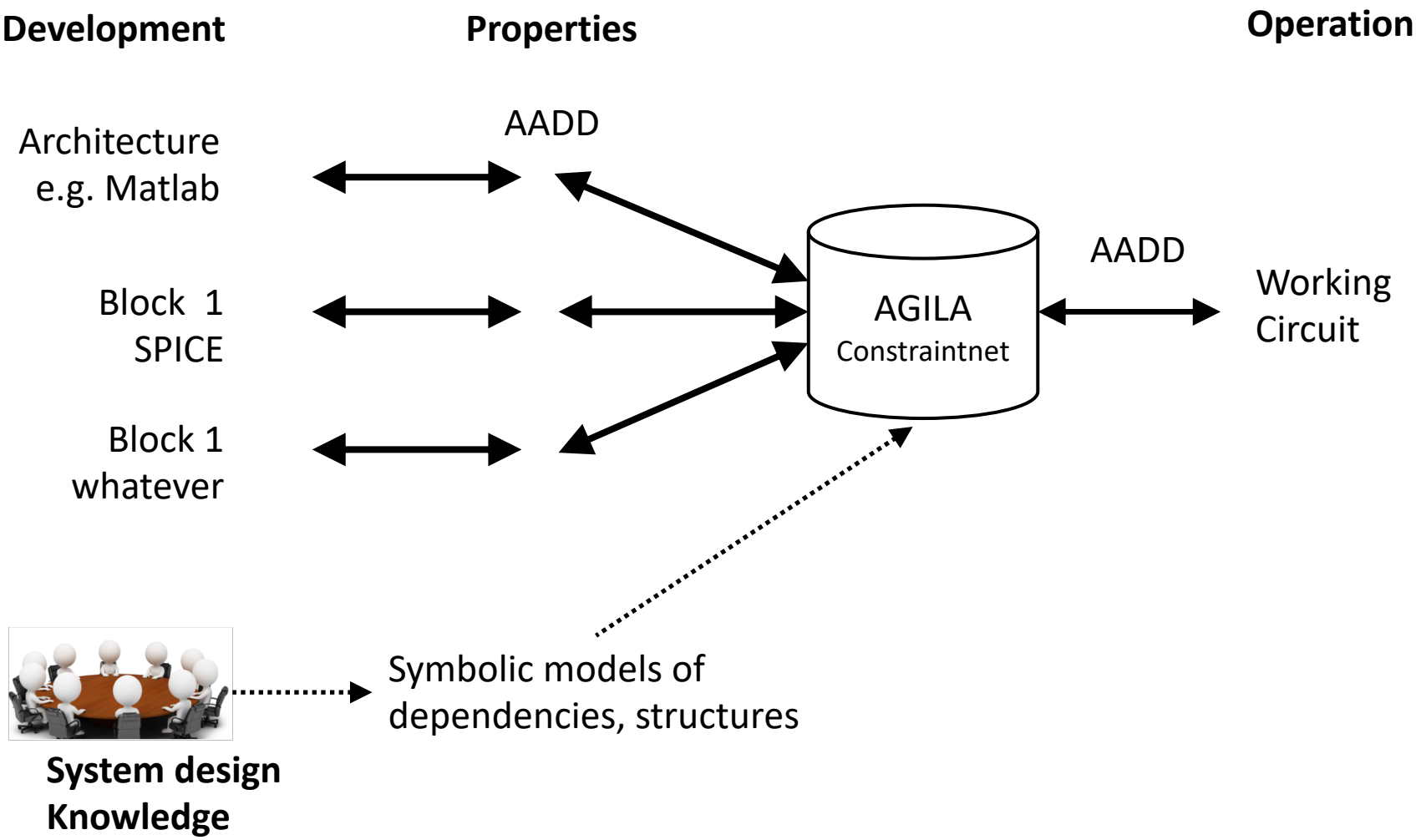
- Combination of SystemC (AMS), AADDLib, SPICE provides a seamless flow, linking circuit-level design and system-level design

- **Interesting observation**: Tool flow used in Automotive / Microelectronics companies for analysis of power train, ECU software, impact on emissions, …

  - Collaboration: Funding for 3-4 PhD or Postdoc from industry
  - Collaboration: AADDlib + VHDL?

- Next major version of AADDlib in preparation
  - Based on Java (jAADD) and Z3

Outlook: Agile development of AMS systems with AGILA



References

- F. Pecheux, C. Grimm, T. Maehne, M. Barnasconi, K. Einwich: *"SystemC AMS based frameworks for virtual prototyping of heterogeneous systems."* 2018 IEEE International Symposium on Circuits and Systems (ISCAS 2018), May 2018. doi: 10.1109/ISCAS.2018.8351864. (online: https://ieeexplore.ieee.org/document/8351864/)
- C. Zivkovic, C. Grimm, M. Olbrich, O. Scharf, and E. Barke, *"Hierarchical verification of AMS systems with affine arithmetic decision diagrams."* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2018, issn: 0278-0070. doi: 10.1109/TCAD.2018.2864238. (online: https://ieeexplore.ieee.org/document/8428606)
- C. Radojicic, .C Grimm, A. Jantsch, M. Rathmair, "*Towards Verification of Uncertain Cyber-Physical Systems."* Proceedings 3rd International Workshop on Symbolic and Numerical Methods for Reachability Analysis, SNR@ETAPS 2017, Uppsala, Sweden, 22nd April. doi: 10.4204/EPTCS.247.1 (online: https://arxiv.org/abs/1705.00519v1)
- C. Grimm, M. Rathmair: Invited: *Dealing with Uncertainties in Analog/Mixed-Signal Systems*. Proceedings of the 54th Annual Design Automation Conference 2017 (DAC 2017); Article no. 35; DOI 10.1145/3061639.3072949 (online: https://dl.acm.org/citation.cfm?doid=3061639.3072949)

Many Thanks!

SAVE THE DATE // FAC 2017
Frontiers of Analog CAD
2017, June, 21st and 22nd // Frankfurt University

Dr. **Christoph Grimm,**
Professor

TU Kaiserslautern
Chair of Design of Cyber-Physical Systems (CPS)

Gottlieb-Daimler Straße 49-418
67663 Kaiserslautern

Tel. +49 (0) 631 - 205 3283
Fax: + 49 (0) 631 - 205 3299

grimm@cs.uni-kl.de
http://cps.cs.uni-kl.de

Complementary Slides

Instrumentation of Selection and Iteration Statements

In EBNF from C++14 grammar:

```
1:  if (condition)
2:    statement
3:  [ else
4:    statement ]
```

instrumented →

```
1:  ifS (condition)
2:    statement
3:  [ elseS
4:    statement ] endS
```

```
1:  while (condition)
2:    statement
```

instrumented →

```
1:  whileS (condition)
2:    statement endS
```

- **ifS, elseS, whileS, endS** are macros that introduce more complex C++ code.
- Instrumentations can be done manually or by simple preprocessor.
- *statement* can be any C++ statement including *selection* and *iteration statements*.

Definition: Path condition & block condition

Path condition

- Conjunction of all conditions from start of program to current state
- Represented in BDD-like part of symbolic variables of type BDD/AADD

Block condition

- Conjunction of elementary predicates whose control flow is not yet joined

  (*"nested blocks"* of selection and iteration statements in C++)

```
1:  a = 0;
2:  ifS (c2)
3:    a = 1;
4:  elseS
5:    a = 2; endS;
6:
```

Block condition is c2.

Block condition is $\overline{c2}$

$a = 1|_{c2}$ *or* $2|_{\overline{c2}}$.
Block condition is true.

Algorithm: Block condition tracking

Global stack of conditions stores all conditions (maybe negated) of the block condition as a BDD (boolean part) w/ symbolic predicates on ε_i (comparisons):

- For all **conditional branches** (no matter by which statement: ifS, whileS, ...)
  - Push condition on stack of block conditions
- For all **merges in control flow**
  - Pop condition from stack of block conditions

Overloaded assignment operators (=, -=, +=, ... on AADD or BDD):

```
1:  METHOD assign(rval, lval: AADD or BDD)
2:    bc := AND(all conditions on stack);
3:    lval := ITE(bc, lval, rval);
4:    return lval;
```

Symbolic Simulation of SystemC (AMS, TDF)

- Concrete and (value-)symbolic signals are sequences of samples $s_i = (s, t)$
  Concrete signal: $s_T := \langle s_0, s_1, s_2, \dots \rangle$
  Symbolic signal: $\hat{s}_T := \langle \hat{s}_0, \hat{s}_1, \hat{s}_2, \dots \rangle$; values can be AADD but not time-tags.

- Concrete and symbolic process activation

```
sc_in<bool>    clk; // concrete signal
sc_in<doubleS> th;  // symbolic signals
sc_out<intS>   cnt;

void count() {
  if (th > 2) cnt += 1;
}
SC_METHOD(count) sensitive << clk;
```

- *Concrete process activation if clk is concrete.*
- *Symbolic activation if clk is symbolic.*

- **TDF & DE MoC** with concrete clk require no further precautions;
  otherwise: overloaded update() of sc_prim_channel needed