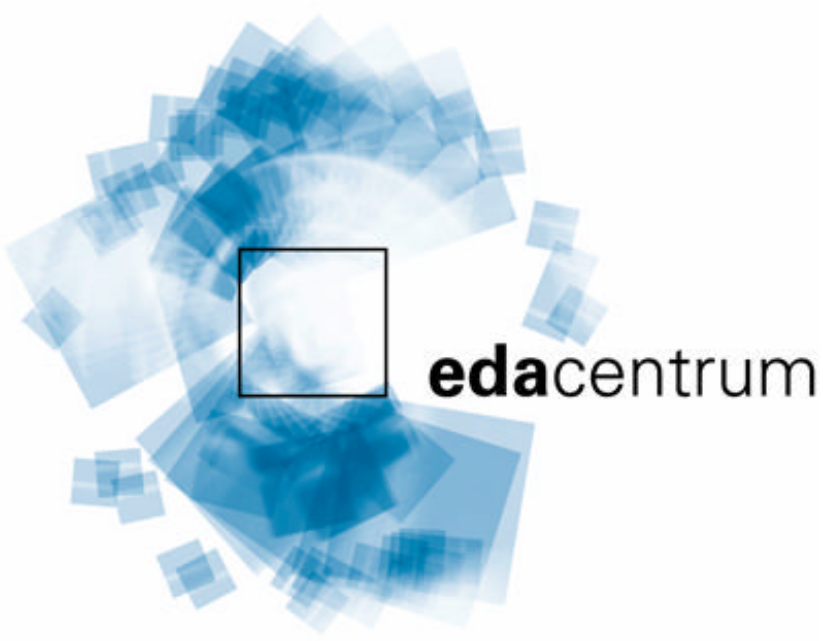


– SystemC and VHDL –

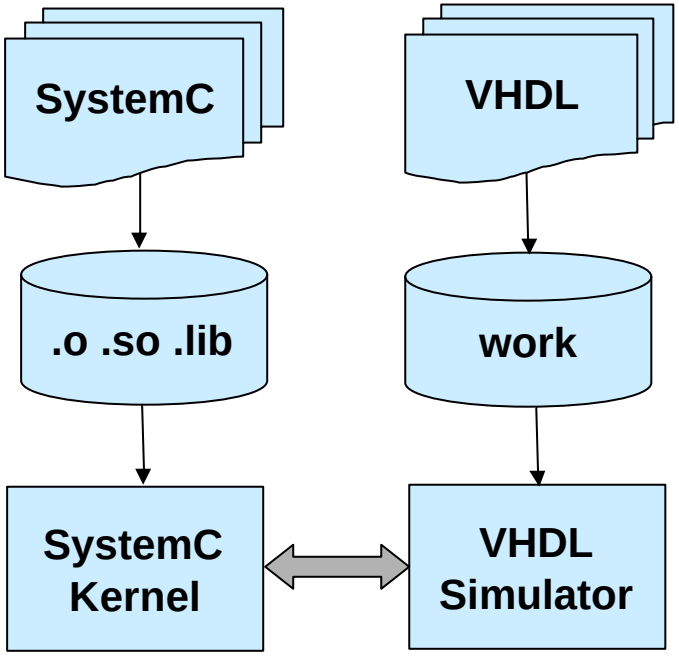
Martin Radetzki
consulting@edacentrum.de
www.edacentrum.de

Mechanisms for Mixing Languages

- **Foreign Language Interface (FLI)**
  - defined in VHDL, limited use, allows calling C functions
  - not a mechanism for coupling VHDL & SystemC simulation
- **Open Model Interface (OMI) – IEEE Std. 1499**
  - hardly available, and if available, doesn‘t work
- **Programming Language Interface (PLI)**
  - gives access to simulator internals; difficult to use
- **Proprietary mechanisms, e.g. from verification add-ons**
  - usually works fine, but ties you to that vendor
- **Mechanism to connect modules from different languages**
  - as easy (?) as simulating VHDL and Verilog together
  - known as language co-simulation or mixed language simulation

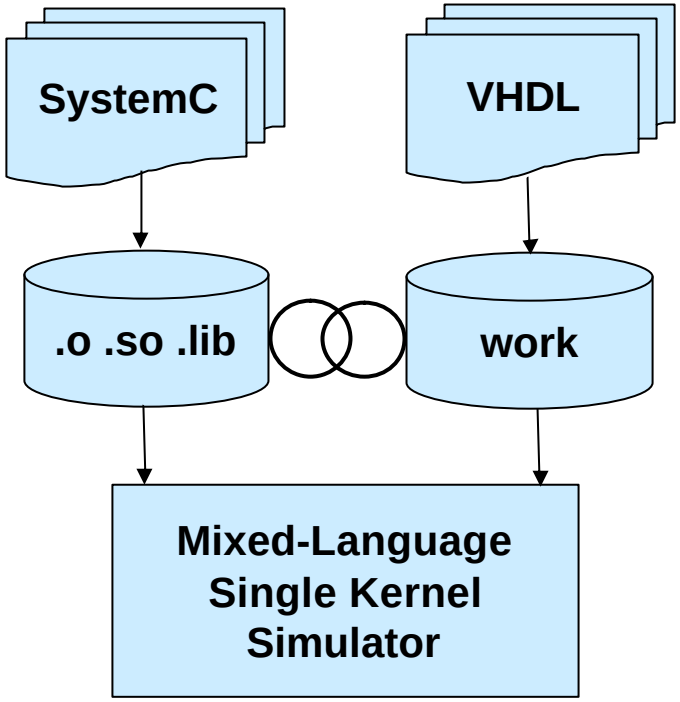
Definition of Terms

Co-Simulation



- Multiple simulator kernels
- Communication overhead

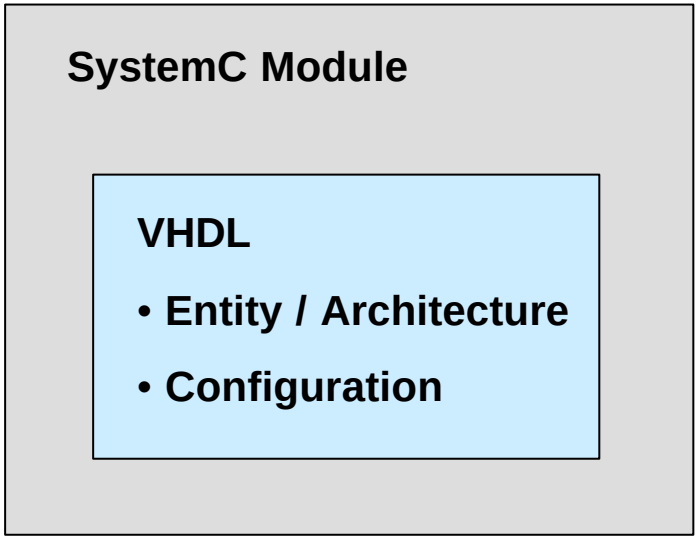
Mixed Language Kernel



- Single simulator kernel
- Significantly smaller overhead

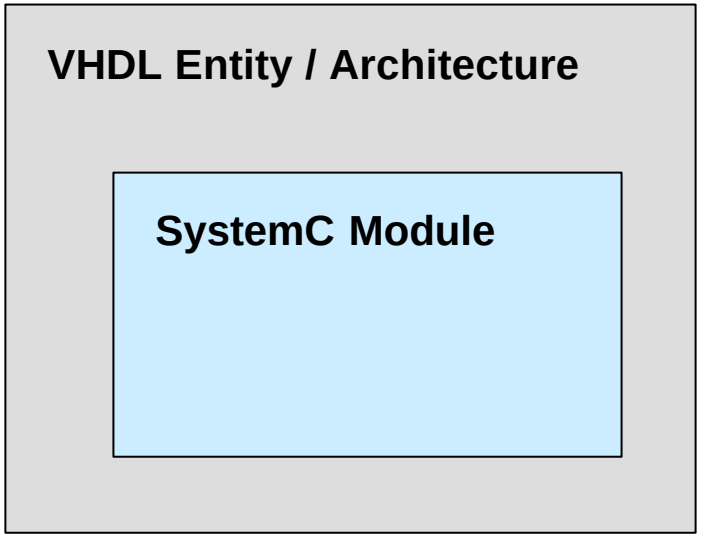
Mixing SystemC and VHDL

VHDL in SystemC



- Typical case
- SystemC testbench
- VHDL design under test

SystemC in VHDL



- SystemC model
- VHDL RTL not yet avail.
- Higher level in VHDL

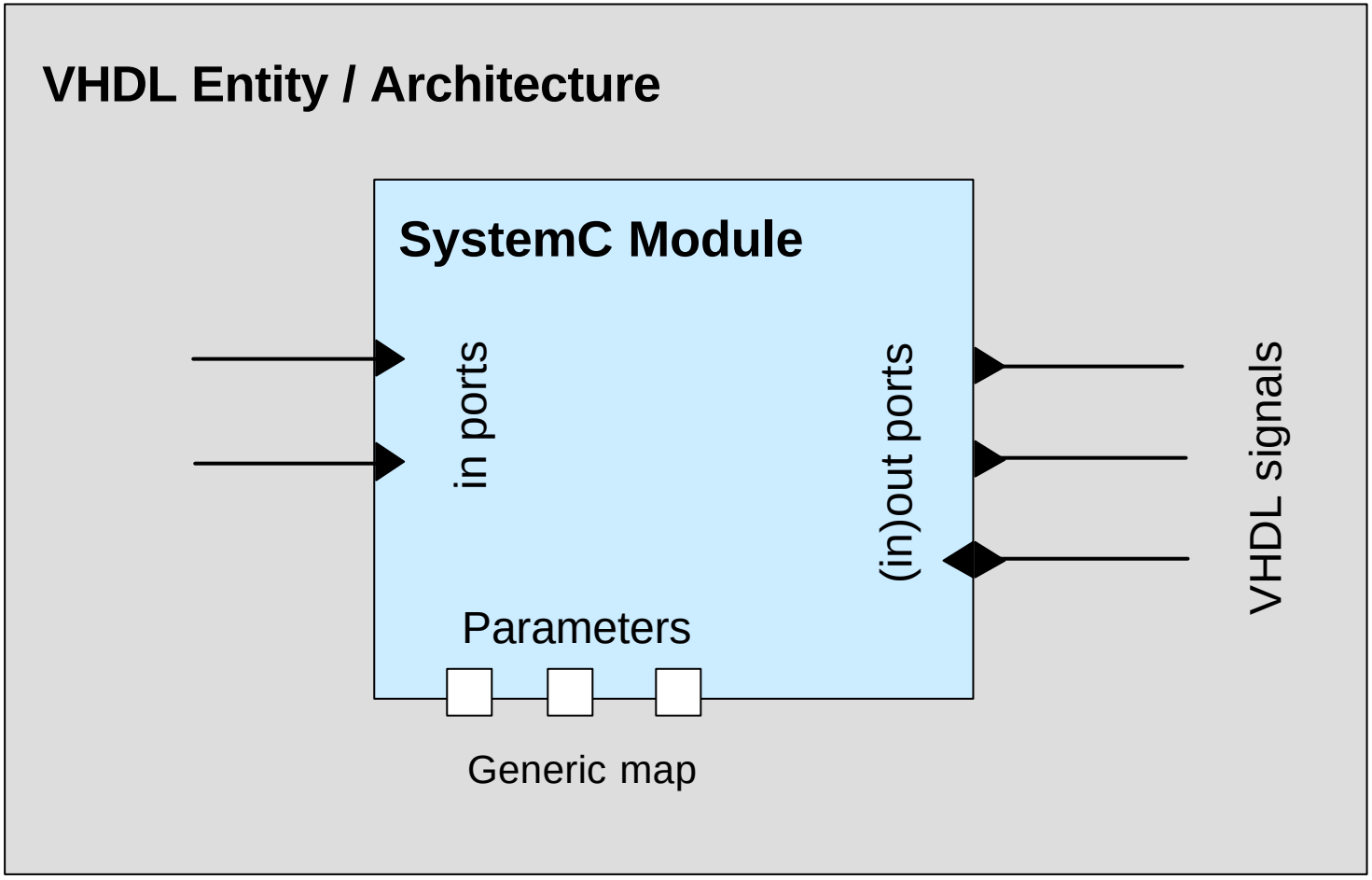
Mixed-Language Fundamentals

› **In principle, mixing VHDL and SystemC is easy**
  – Both languages have modules with ports
  – Simply instantiate a module from one language using the other language's instantiation syntax

› **The devil is in the details**
  – The instantiated module is missing when compiling higher level
  – Languages provide different ports and data types
  – Parameter (generics) mechanisms are quite different
  – SystemC modules may use higher-level communication

› **This presentation shows how it works**
  – Sample simulator: Modelsim 6.0

SystemC in VHDL



SystemC Module To Be Instantiated

comparator.h

```
SC_MODULE(Comparator)
{
  // ports
  sc_in<int>   in1, in2;
  sc_out<bool> out1;

  // process declaration
  void action();

  SC_CTOR(Comparator) {
    SC_METHOD(action);
    sensitive << in1
              << in2;
  }
};
```

comparator.cpp

```
#include "comparator.h"

void Comparator::action()
{
  out1 = false;
  if (in1 > in2) {
    out1 = true;
  }
}

SC_MODULE_EXPORT(Comparator);
```

need to „export“ every top module of a SystemC module hierarchy

Instantiation in VHDL

vgencomp Comparator generates a VHDL component

```
component Comparator
   port(
     in1  : in  std_logic_vector(31 downto 0);
     in2  : in  std_logic_vector(31 downto 0);
     out1 : out std_logic
   );
end component;
... -- instantiation to be written by user
c1 : comparator port map(a, b, a_gt_b);
c2 : comparator port map(
                   in1  => b,
                   in2  => c,
                   out1 => b_gt_c
                 );
c3 : comparator port map(c, a, out1 => c_gt_a);
```

VHDL “stub“ / socket

component instantiation

Binding by Name

- **No configuration for instances of component Comparator**
- **Instances are connected to a module**
  - that has the same name as the configuration
  - that has port names and types matching the component‘s ports

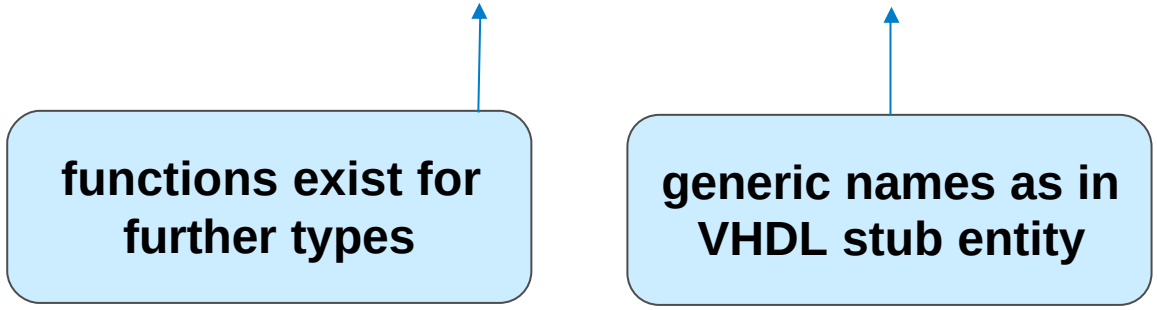
- **SystemC is case-sensitive**
  - “Comparator“ != “comparator“
- **VHDL is case-insensitive**
  - “Comparator“ = “comparator“

- **User manual: all names must match exactly (that‘s safe)**
- **Practice: names match regardless of case**
  (but maybe not when using other tools)

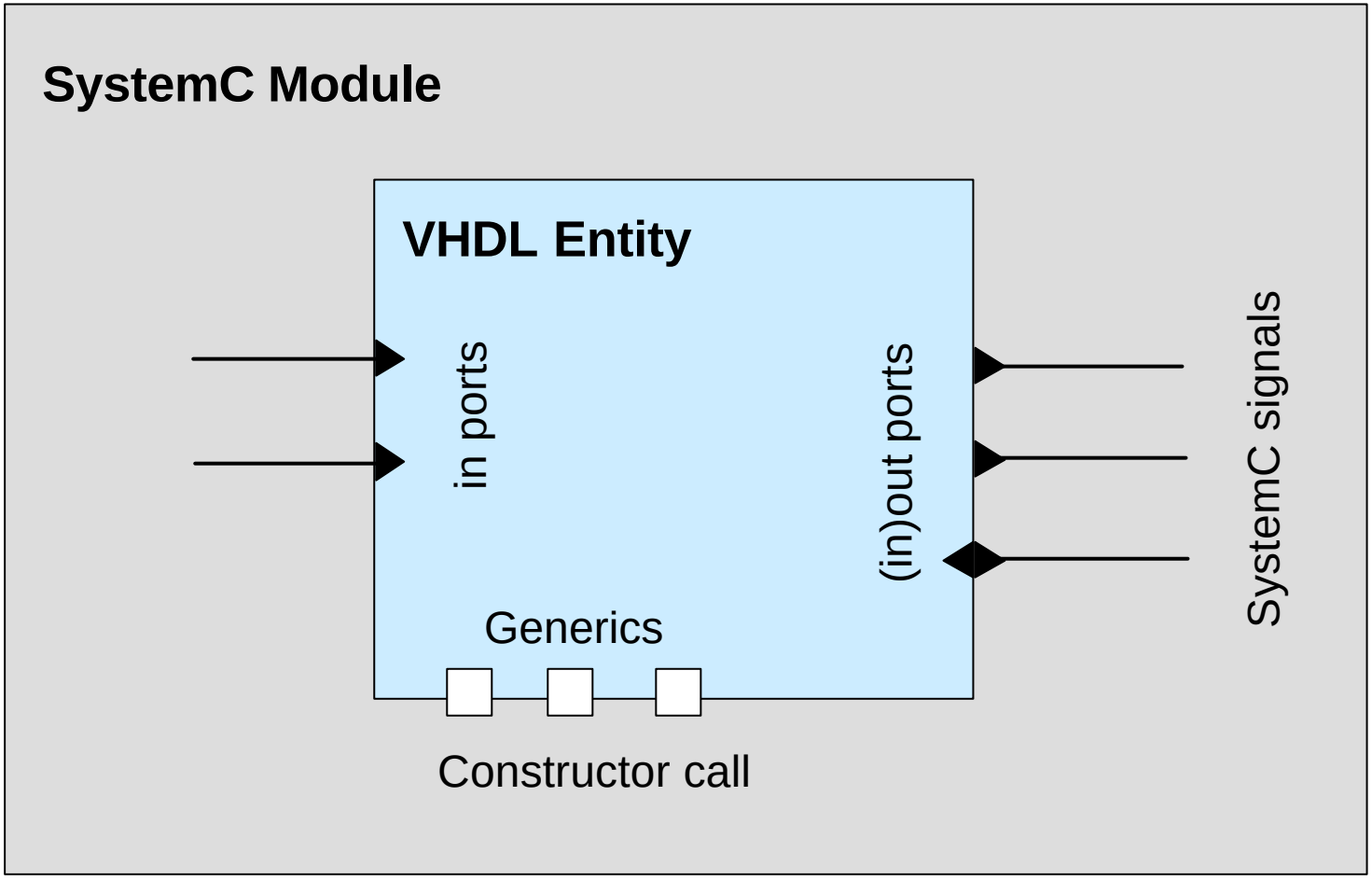
Passing Parameters from VHDL to SystemC

- **Currently requires use of a workaround flow**
- **Generation of stub entity for the SystemC module**
  - Option 1: Automatic generation
  - Option 2: Manual generation
- **SystemC module constructor must obtain generic values through specific function calls, to be applied in the same order as the generics appear in VHDL:**

```
int   generic_int  = sc_get_int_param("int_generic");
float generic_real = sc_get_real_param("real_generic");
bool  generic_bool = sc_get_bool_param("bool_generic");
```



VHDL in SystemC



Foreign Module Declaration

```
entity sorter is
   port(
     a : in std_logic_vector(31 downto 0);
     b : in std_logic_vector(31 downto 0);
     c : in std_logic_vector(31 downto 0);
     biggest : out std_logic_vector(31 downto 0) );
end;
```

VHDL entity to instantiate

```
scgenmod sorter
```

simulator command

```
class Sorter : public sc_foreign_module
{
public:
   sc_in<sc_lv<32> > a;
   sc_in<sc_lv<32> > b;
   sc_in<sc_lv<32> > c;
   sc_out<sc_lv<32> > biggest;

  Sorter(sc_module_name nm, const char* hdl_name)
    : sc_foreign_module(nm, hdl_name), a("a"), b("b"), c("c"), ...
  {}
};
```

SystemC stub

VHDL in SystemC – Instantiation

```
#include "sorter.h"
...
SC_MODULE( testbench )
{
  // signals to wire up the device under test
  sc_signal<sc_lv<32> > asig, bsig, csig, fsig;
...
  Sorter dut;
...
  SC_CTOR( testbench ) : ...,
    dut( "dut", "sorter" )
  {
  ...
    dut.a(asig);
    dut.b(bsig);
    dut.c(csig);
    dut.biggest(fsig);
  ...
```

SystemC stub

instantiation

initialization with additional parameter

port / signal connection

Passing Parameters from SystemC to VHDL

```
class Sorter : public sc_foreign_module
{
public:
   ...
  Sorter(sc_module_name nm,
         const char* hdl_name,
         int num_generics,
         const char* generic_list)
    : sc_foreign_module(nm, hdl_name,
          num_generics, generic_list),
            a("a"), b("b"), c("c"), ...
            {}
```

Additional stub constructor parameters to pass generic values

Pass values on to SystemC foreign module class

```
Sorter dut; // class member
...
// in class constructor
const char *generic_list[3];
generic_list[0] = strdup("int_generic=7");
generic_list[1] = strdup("real_generic=1.5");
generic_list[2] = strdup("bool_generic=false");
dut = new Sorter("dut", "sorter", 3, generic_list);
```

VHDL generic names

Call constructor with generic_list; requires dynamic instantiation

Compatible SystemC and VHDL Ports

SystemC port	VHDL port

`sc_in<T>`
`sc_out<T>`
`sc_inout<T>`
	`in`
	`out / buffer`
	`inout`
	} VHDL type depends on T - see next slide

`sc_in_resolved`
`sc_out_resolved`
`sc_inout_resolved`
	`in std_logic`
	`out std_logic`
	`inout std_logic`

`sc_in_rv<N>`
`sc_out_rv<N>`
`sc_inout_rv<N>`
	`in std_logic_vector(N-1 downto 0)`
	`out std_logic_vector(N-1 downto 0)`
	`inout std_logic_vector( ... )`

`sc_in_clk`
`sc_out_clk`
`sc_inout_clk`
	`in boolean / bit / std_[u]logic`
	`out boolean / bit / std_[u]logic`
	`inout boolean / bit / std_[u]logic`

other port types are not supported on the language boundary	

Compatible SystemC and VHDL Data Types

SystemC type T	VHDL type
`bool / sc_bit` `sc_logic`	`boolean / bit / std_[u]logic` `std_[u]logic`
`sc_bv<N>` `sc_lv<N>`	`bit_vector(N-1 downto 0)` `std_[u]logic_vector(N-1 downto 0)`

`sc_[u]int<N>`
`[unsigned] char`
`[unsigned] int`
`[unsigned] long`
	`bit_vector(N-1 downto 0)`
	`std_[u]logic_vector(N-1 downto 0)`

other data types are not supported on the language boundary	

VHDL to SystemC Logic Value Mapping

std_logic	sc_logic	sc_bit	bool
0	0	0	false
L	0	0	false
1	1	1	true
H	1	1	true
Z	Z	0	false
U	X	0	false
W	X	0	false
X	X	0	false
-	X	0	false

no logic values are lost when mapping from SystemC to VHDL

electronic design automation centrum

Conclusion

Mixed-language simulation for SystemC and VHDL ...

- **works**
- **uses mechanisms similar to mixed VHDL / Verilog simulation**
- **is fairly simple as long as no generics / parameters involved**
- **employs some mechanisms that are not standardized**

If you need consulting on whatever aspects of SystemC ...

- **talk to us today**
- **contact consulting@edacentrum.de later**