Supporting Dimensional Analysis in SystemC-AMS

Torsten Mähne Alain Vachoux

Laboratoire de Systèmes Microélectroniques (LSM)
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Suisse

Behavioral Modeling and Simulation Conference (BMAS) 2009
17 to 18 September 2009, Doubletree Hotel, San Jose, California, USA

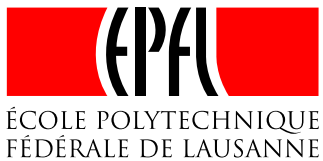
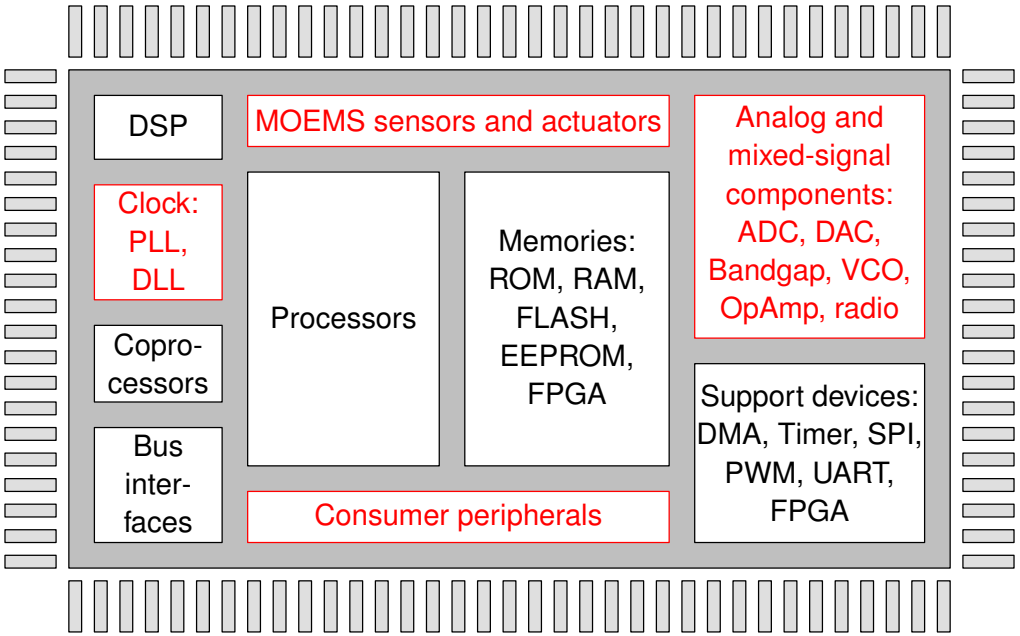


Table of Contents

Heterogeneous SoCs Design Process Challenges



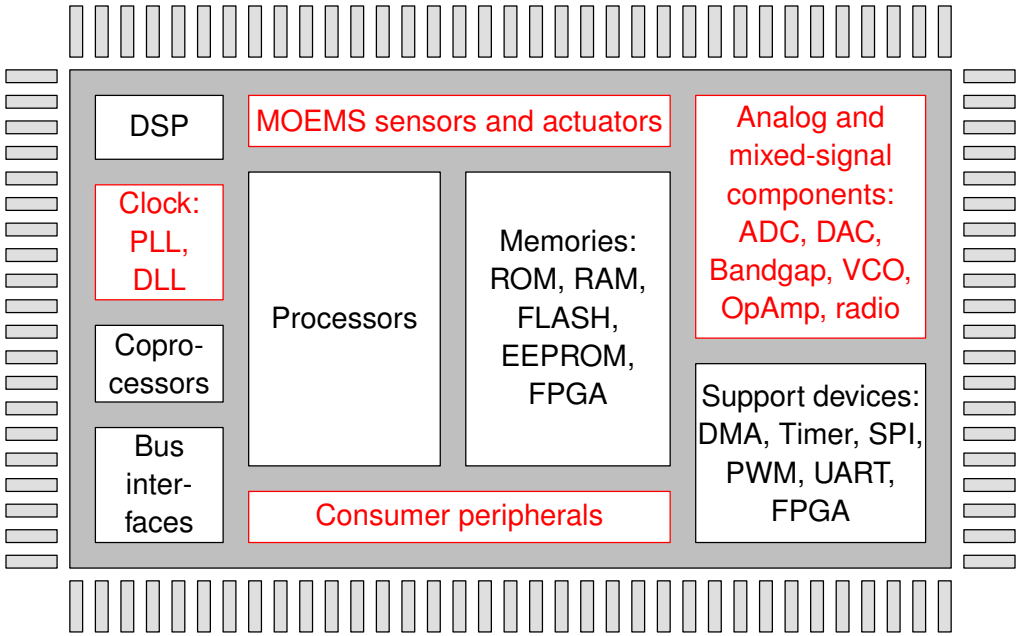
- Component reuse and retargeting
- Multiple engineering/physical domains
- Multiple Models of Computations (MoCs)
- Conflicting naming conventions for (elec., mech., . . .) quantities

↝ Strict interface specifications—not only value type

↝ Dimensional analysis: *quantity<unit<dimension, system>, value_type>*

↝ Check model assembly and calculation consistency

Heterogeneous SoCs Design Process Challenges



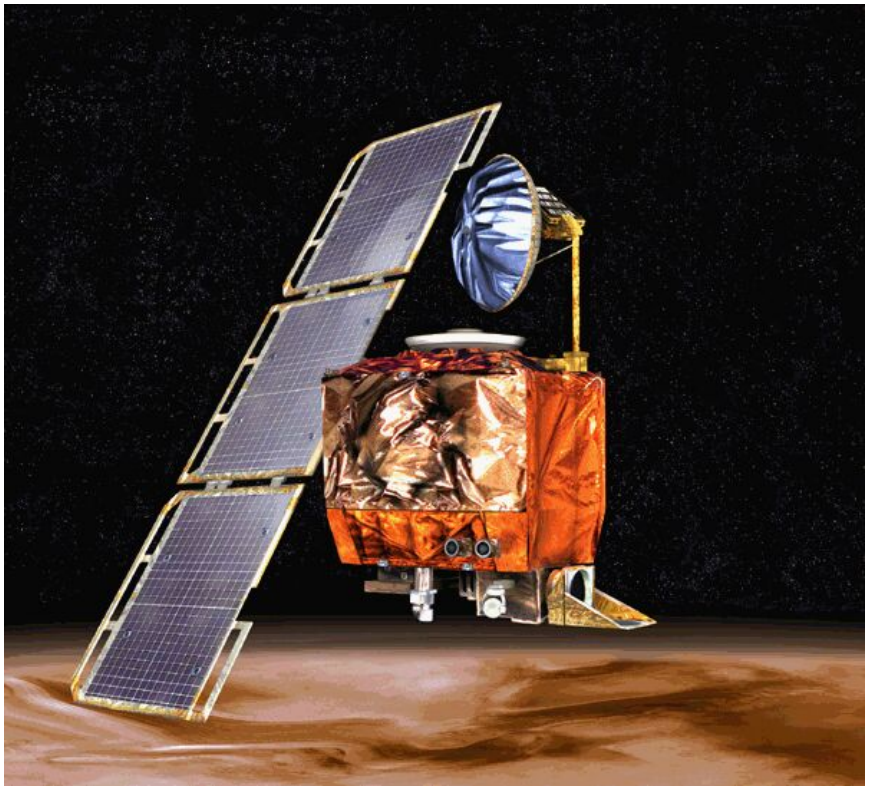
- Component reuse and retargeting
- Multiple engineering/physical domains
- Multiple Models of Computations (MoCs)
- Conflicting naming conventions for (elec., mech., . . .) quantities

⇝ Strict interface specifications—not only value type

⇝ Dimensional analysis: *quantity<unit<dimension, system>, value_type>*

⇝ Check model assembly and calculation consistency

Motivation (1/2): Loss of NASA's Mars Climate Orbiter

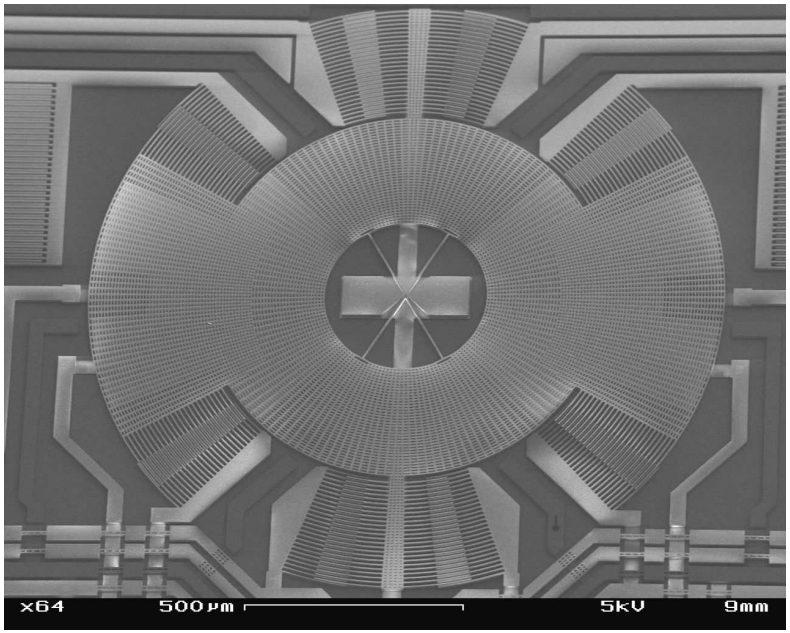


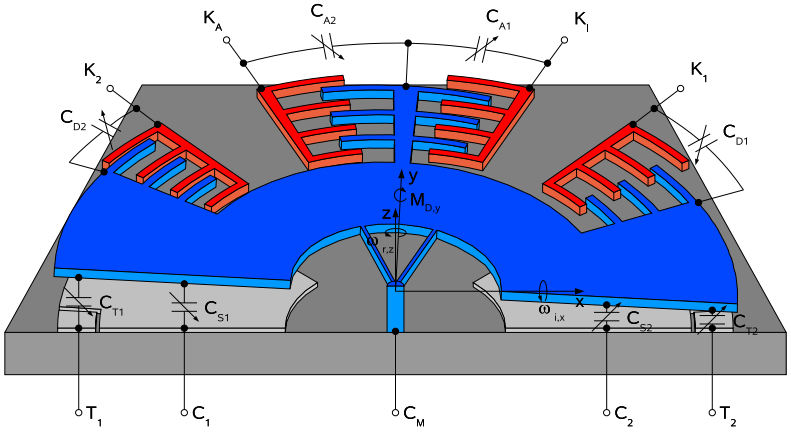
Mars Climate Orbiter (MCO) was lost on 23 September 1999 during its orbit entering maneuver.
Details: http://marsprogram.jpl.nasa.gov/msp98/orbiter/

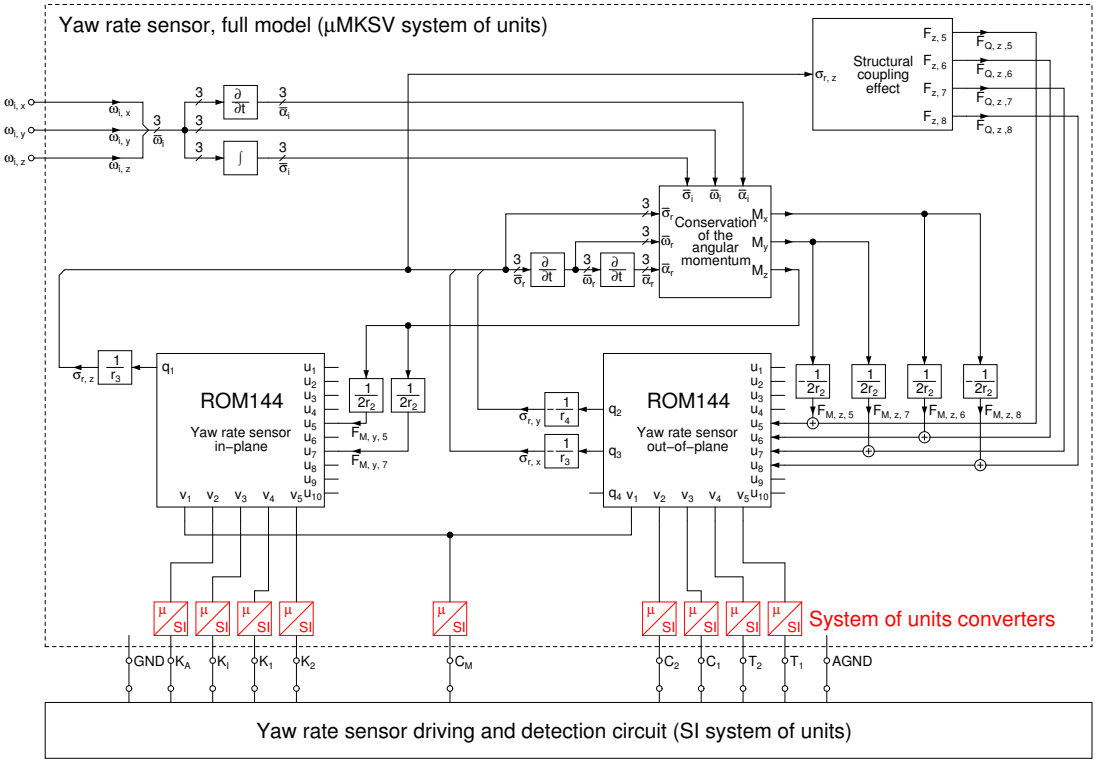
Statements by Arthur Stephenson (Chairman of MCO Mission Failure Investigation Board, 10 November 1999):

- "The 'root cause' of the loss of the spacecraft was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software, as NASA has previously announced."
- "The failure review board has identified other significant factors that allowed this error to be born, and then let it linger and propagate to the point where it resulted in a major error in our understanding of the spacecraft's path as it approached Mars."

Motivation (2/2): Usage of Multiple Systems of Units







- MEMS yaw rate sensor behavioral model
- Sensor model using µMKSV system of units
- Control circuit model using SI system of units

⇝ Insertion of system of units converters

State of the Art: Dimensional Analysis in HDLs

- VHDL-AMS: Unit specification with attributes purely for presentation

```
subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
attribute UNIT OF VELOCITY : subtype is "meter/second";
attribute SYMBOL OF VELOCITY : subtype is "m/s";
```

- Modelica: SI unit specification (+ scale factors) in attributes of `Real`

```
Real(unit="m.s-1") v = 2.0;
```

  - Tool-specific dimensional analysis solutions, e.g., Dymola, Simulator X
  - Efforts to formalize and generalize physical unit checking

- F#: Units part of language and dimensional analysis built in compiler

```
[<Measure>] type kg
[<Measure>] type N = kg m/s^2
let gravity = 9.808<m/s^2>
let metresToFeet (l:float<m>) = l * 3.28084<ft/m>
```

  - Extensible by declaring new units and system of units
  - No notion of dimension (classes of measurement units)
  - Programing language without dedicated supported for system modeling

State of the Art: Dimensional Analysis in HDLs

- VHDL-AMS: Unit specification with attributes purely for presentation

```
subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
attribute UNIT OF VELOCITY : subtype is "meter/second";
attribute SYMBOL OF VELOCITY : subtype is "m/s";
```

- Modelica: SI unit specification (+ scale factors) in attributes of `Real`

```
Real(unit="m.s-1") v = 2.0;
```

  - Tool-specific dimensional analysis solutions, e.g., Dymola, Simulator X
  - Efforts to formalize and generalize physical unit checking

- F#: Units part of language and dimensional analysis built in compiler

```
[<Measure>] type kg
[<Measure>] type N = kg m/s^2
let gravity = 9.808<m/s^2>
let metresToFeet (l:float<m>) = l * 3.28084<ft/m>
```

  - Extensible by declaring new units and system of units
  - No notion of dimension (classes of measurement units)
  - Programing language without dedicated supported for system modeling

State of the Art: Dimensional Analysis in HDLs

- VHDL-AMS: Unit specification with attributes purely for presentation

```
subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
attribute UNIT OF VELOCITY : subtype is "meter/second";
attribute SYMBOL OF VELOCITY : subtype is "m/s";
```

- Modelica: SI unit specification (+ scale factors) in attributes of `Real`

```
Real(unit="m.s-1") v = 2.0;
```

  - Tool-specific dimensional analysis solutions, e.g., Dymola, Simulator X
  - Efforts to formalize and generalize physical unit checking

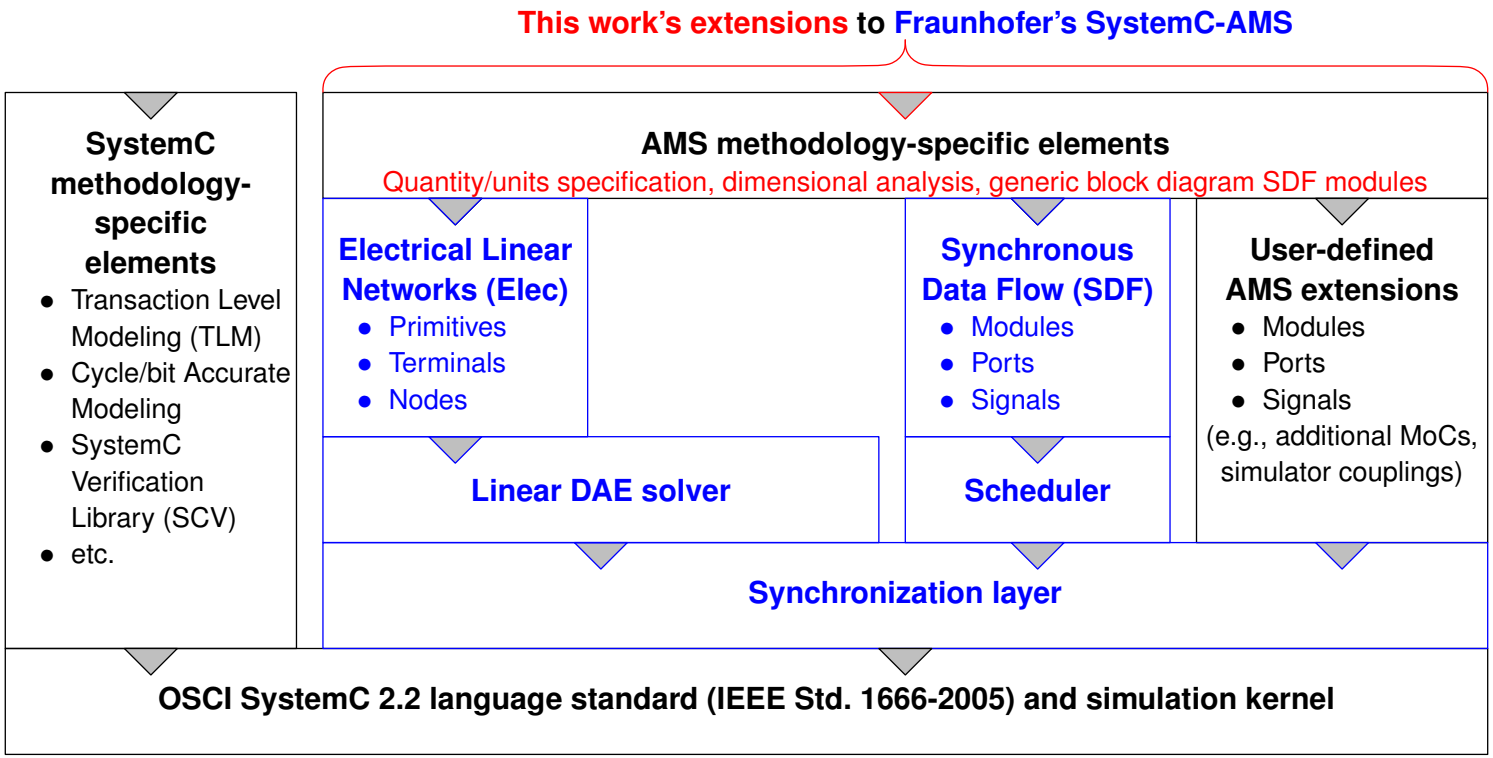
- F#: Units part of language and dimensional analysis built in compiler

```
[<Measure>] type kg
[<Measure>] type N = kg m/s^2
let gravity = 9.808<m/s^2>
let metresToFeet (l:float<m>) = l * 3.28084<ft/m>
```

  - Extensible by declaring new units and system of units
  - No notion of dimension (classes of measurement units)
  - Programing language without dedicated supported for system modeling

AMS Extensions to SystemC

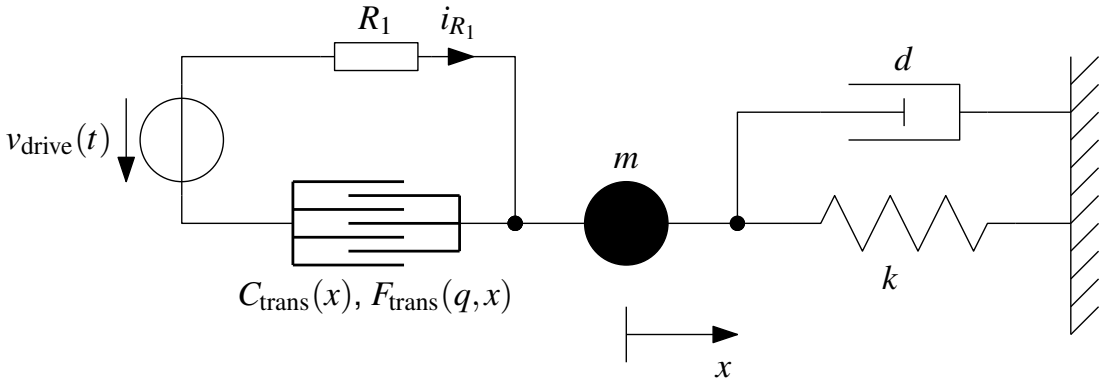
Goal: Simulation of heterogeneous SoCs using several synchronised MoCs
OSCI standardisation effort: based on SystemC-AMS (Fraunhofer IIS/EAS)



Our contribution: New modeling capabilities for formal/consistent description of energy conserving multi-domain systems at a high level of abstraction.

Modeling of Multi-Domain Systems (1/2)

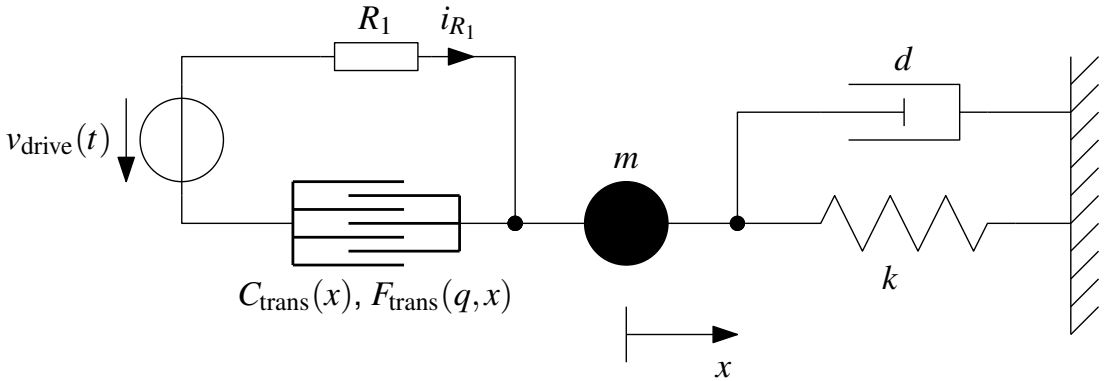
1. “Classic” domain-specific conservative model:



- Implementation of primitives per domain
- SystemC-AMS offers only linear electrical primitives

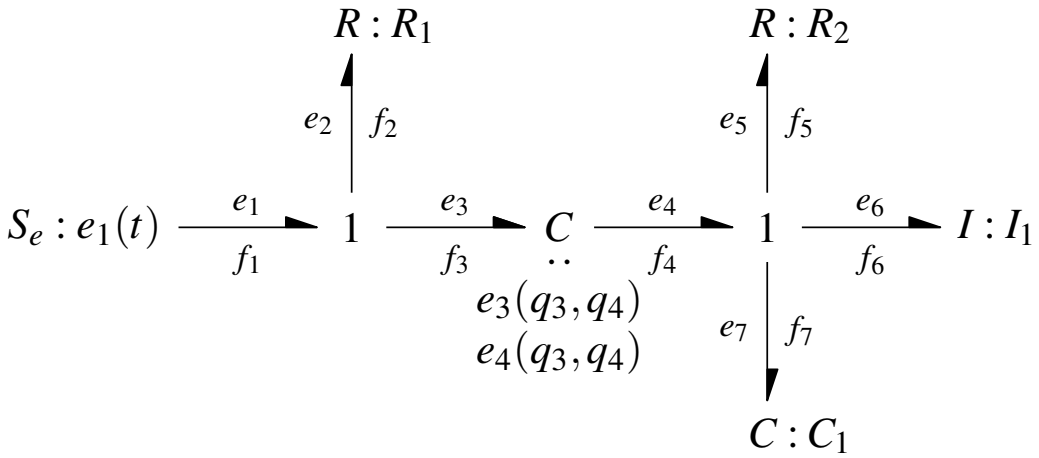
Modeling of Multi-Domain Systems (1/2)

1. "Classic" domain-specific conservative model:



- Implementation of primitives per domain
- SystemC-AMS offers only linear electrical primitives

2. Equivalent generic bond graph model:

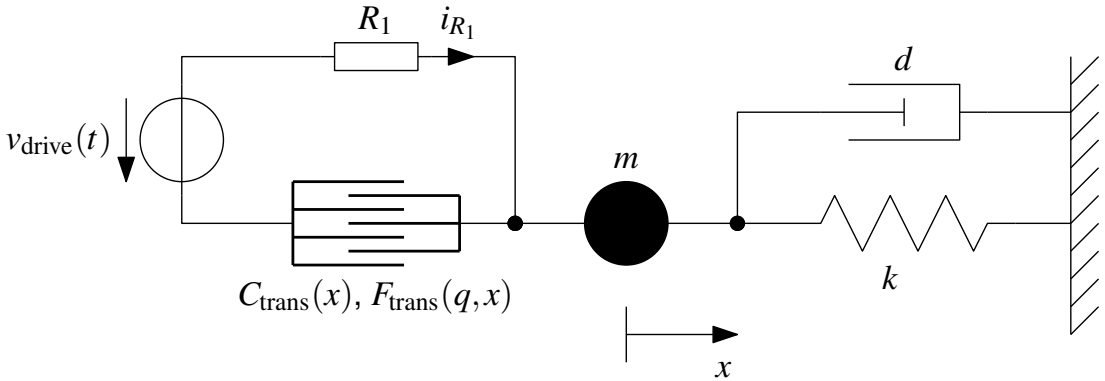


+ Primitives parametrizable to domain
+ Causality analysis extracts computational structure
- Not (yet) supported by SystemC-AMS

- Check model assembly, consistent equations $\rightsquigarrow$ dimensional analysis

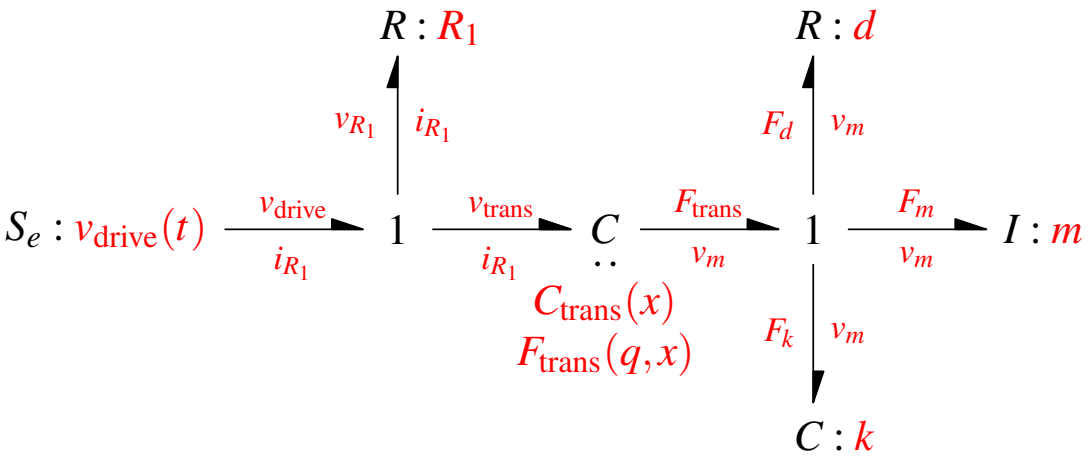
Modeling of Multi-Domain Systems (1/2)

1. "Classic" domain-specific conservative model:



- Implementation of primitives per domain
- SystemC-AMS offers only linear electrical primitives

2. Equivalent acausal bond graph model:

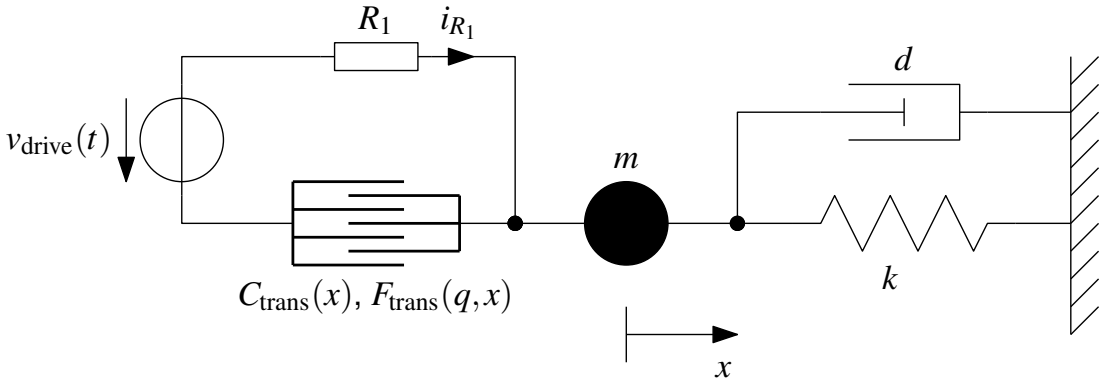


+ Primitives parametrizable to domain
+ Causality analysis extracts computational structure
- Not (yet) supported by SystemC-AMS

- Check model assembly, consistent equations $\rightsquigarrow$ dimensional analysis

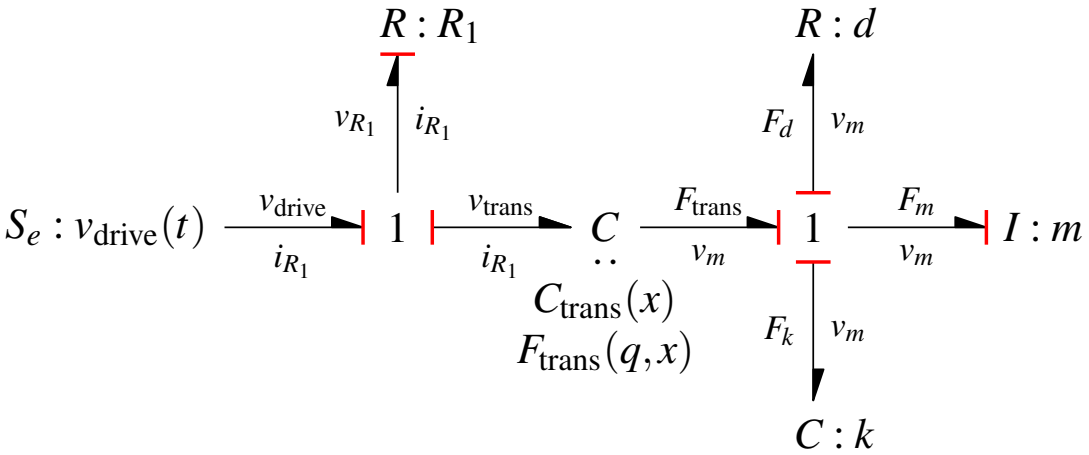
Modeling of Multi-Domain Systems (1/2)

1. “Classic” domain-specific conservative model:



- Implementation of primitives per domain
- SystemC-AMS offers only linear electrical primitives

2. Equivalent causal bond graph model:

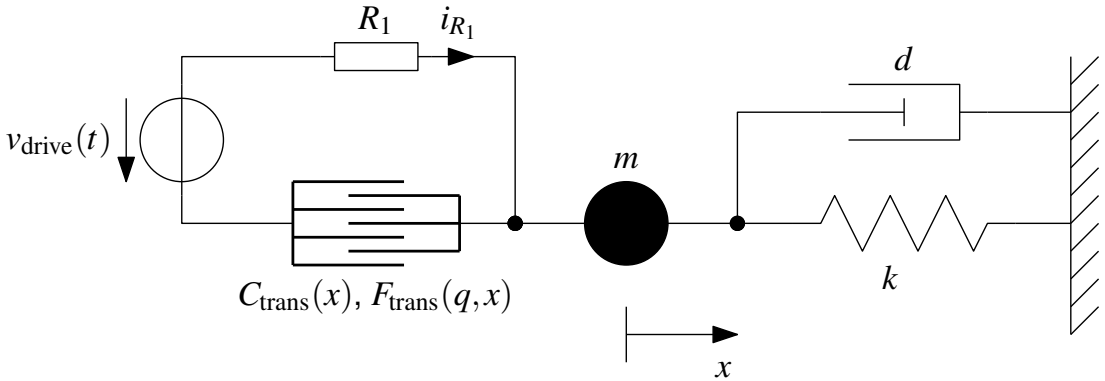


+ Primitives parametrizable to domain
+ Causality analysis extracts computational structure
- Not (yet) supported by SystemC-AMS

▸ Check model assembly, consistent equations ⇝ dimensional analysis

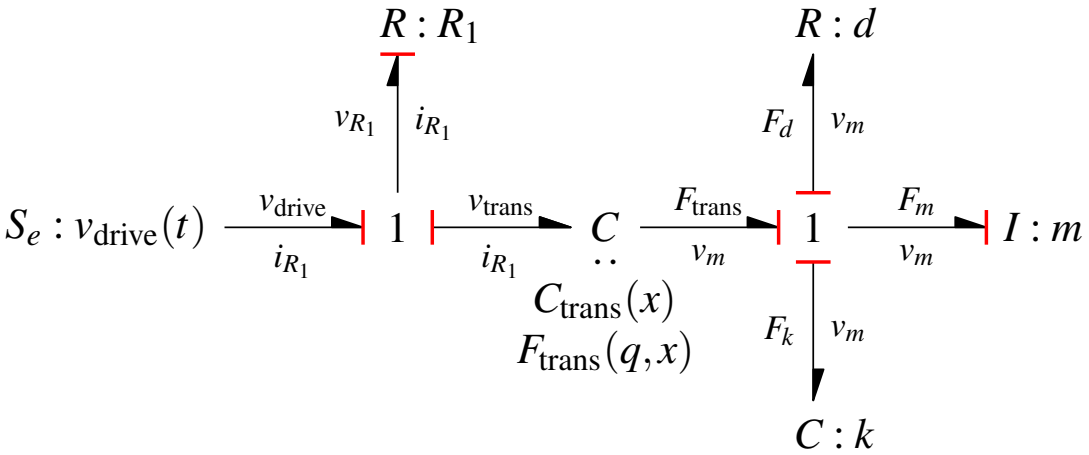
Modeling of Multi-Domain Systems (1/2)

1. “Classic” domain-specific conservative model:



- Implementation of primitives per domain
- SystemC-AMS offers only linear electrical primitives

2. Equivalent causal bond graph model:

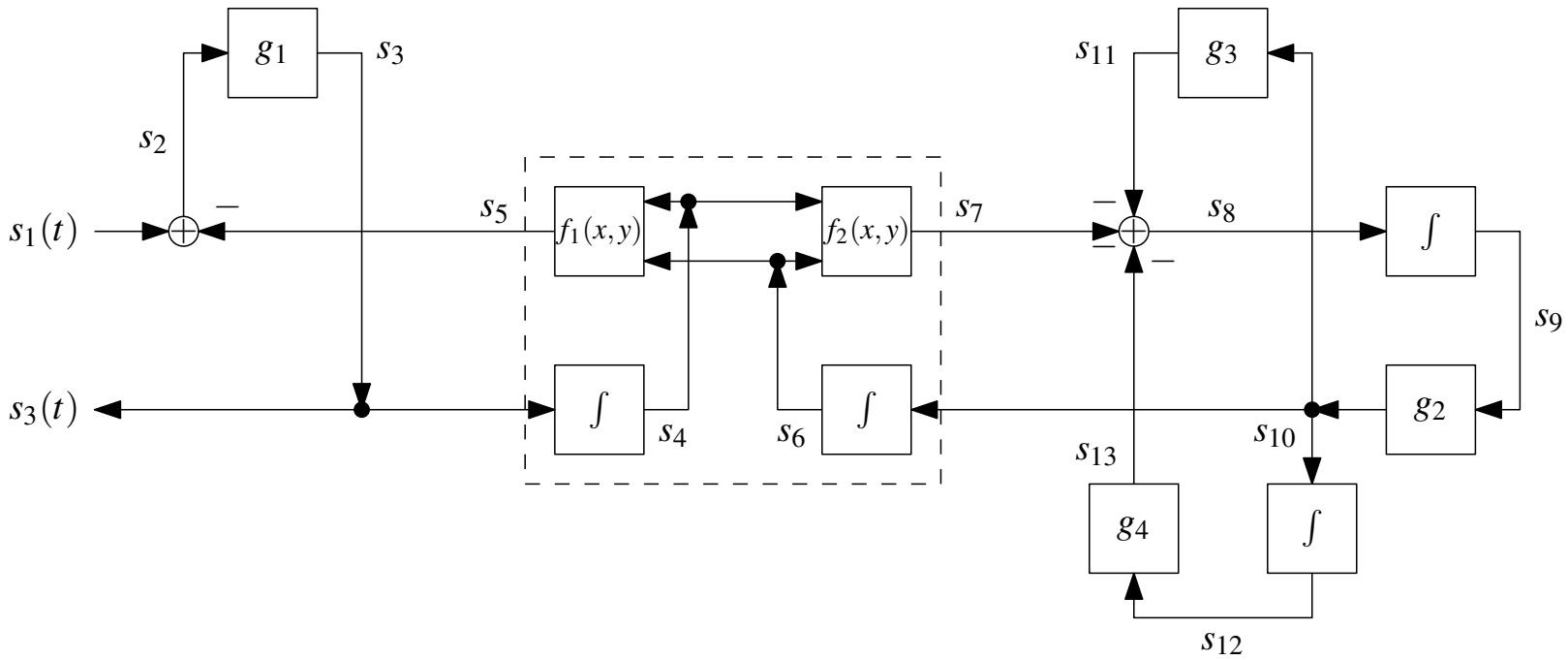


+ Primitives parametrizable to domain
+ Causality analysis extracts computational structure
- Not (yet) supported by SystemC-AMS

- Check model assembly, consistent equations $\rightsquigarrow$ dimensional analysis

Modeling of Multi-Domain Systems (2/2)

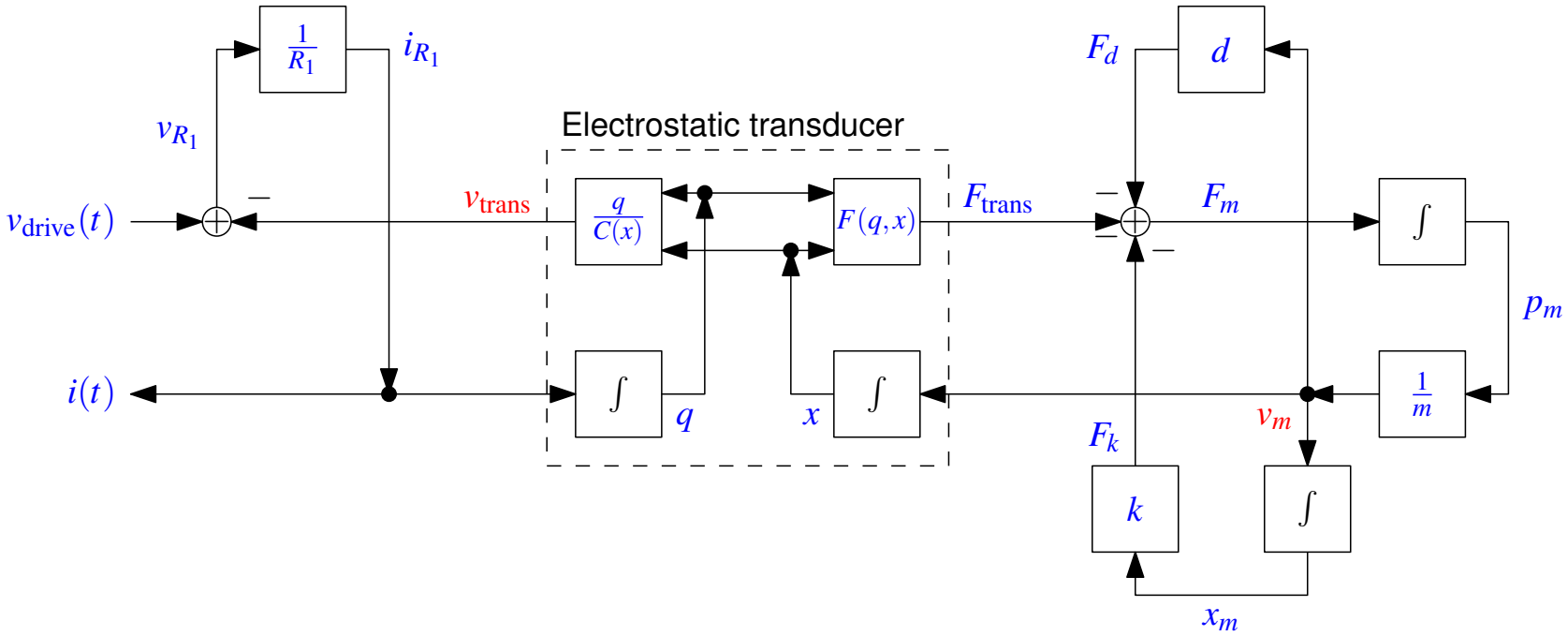
3. Derived block diagram:



+ Fully causal $\rightsquigarrow$ static scheduling $\rightsquigarrow$ procedural execution
+ Simulatable with SystemC-AMS's Synchronous Data Flow (SDF) MoC
– Energy conservation emulated
– Hard to reuse in other context with different input/output roles
$\rightsquigarrow$ Annotated units resolve symbol conflicts

Modeling of Multi-Domain Systems (2/2)

3. Derived block diagram:

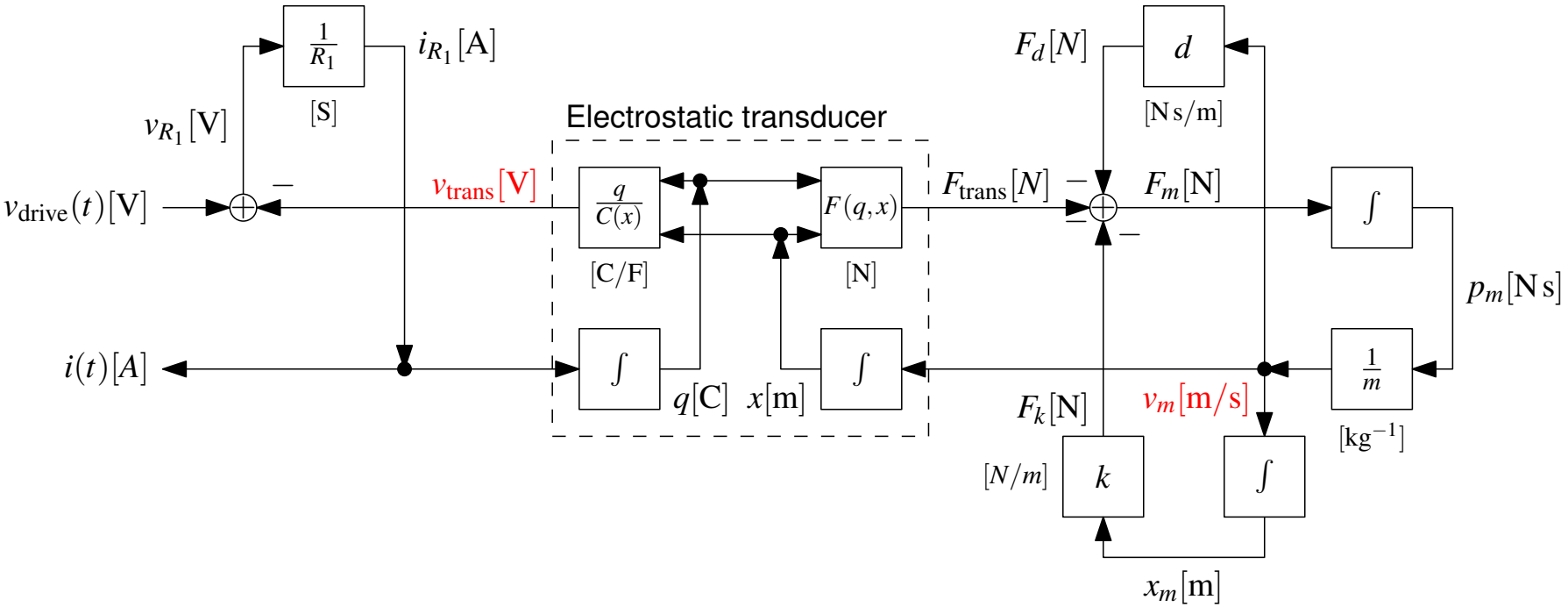


+ Fully causal $\rightsquigarrow$ static scheduling $\rightsquigarrow$ procedural execution
+ Simulatable with SystemC-AMS's Synchronous Data Flow (SDF) MoC
– Energy conservation emulated
– Hard to reuse in other context with different input/output roles

$\rightsquigarrow$ Annotated units resolve symbol conflicts

Modeling of Multi-Domain Systems (2/2)

3. Derived block diagram:



+ Fully causal ⇝ static scheduling ⇝ procedural execution
+ Simulatable with SystemC-AMS's Synchronous Data Flow (SDF) MoC
– Energy conservation emulated
– Hard to reuse in other context with different input/output roles

⇝ Annotated units resolve symbol conflicts

Boost.Units: C++ Library for Dimensional Analysis

- `unit<Dim, System>` type: to represent arbitrary composite unit
  - `Dim`: Static type list of base dimensions raised to a rational power
  - `System`: Set of base dimensions and their measures
  - Example: Energy is $[M]^1[L]^2[T]^{-2}$, in SI: $\mathrm{kg\,m^2\,s^{-2}} = \mathrm{N\,m} = \mathrm{J}$
- `quantity<U, V>` type: composed of unit `U` and value `V` types
- Code example:

```
// Current quantity with double value
quantity<si::current> i = 0.1 * si::ampere;
// Complex impedance quantity
quantity<si::resistance, complex<double> >
  Z = complex<double>(4.0, 3.0) * si::ohm;
// Assignment of product only to a complex voltage quantity
quantity<si::electric_potential, complex<double> >
  v = Z * i;
```

Boost.Units: C++ Library for Dimensional Analysis

- `unit<Dim, System>` type: to represent arbitrary composite unit
  - `Dim`: Static type list of base dimensions raised to a rational power
  - `System`: Set of base dimensions and their measures
  - Example: Energy is $[M]^1[L]^2[T]^{-2}$, in SI: $\mathrm{kg\,m^2\,s^{-2}} = \mathrm{N\,m} = \mathrm{J}$
- `quantity<U, V>` type: composed of unit `U` and value `V` types
- Code example:

```
// Current quantity with double value
quantity<si::current> i = 0.1 * si::ampere;
// Complex impedance quantity
quantity<si::resistance, complex<double> >
  Z = complex<double>(4.0, 3.0) * si::ohm;
// Assignment of product only to a complex voltage quantity
quantity<si::electric_potential, complex<double> >
  v = Z * i;
```

Boost.Units Dimensional Analysis Examples (1/2)

- Accidental Summing of Different Quantity Types:

```
quantity<si::electric_potential> v_1 = 5.0 * si::volt;
quantity<si::velocity> v_2 = 3.0 * si::meter / si::second;
// This won't compile:
quantity<si::electric_potential> v_tot = v_1 + v_2;
```

- Compiler error:

```
fail_bmas2009_examples.cpp:16: error: no match for 'operator+' in 'v_1 + v_2'
```

↝ Cannot sum voltage quantity v_1 and speed quantity v_2

Boost.Units Dimensional Analysis Examples (1/2)

- Accidental Summing of Different Quantity Types:

```
quantity<si::electric_potential> v_1 = 5.0 * si::volt;
quantity<si::velocity> v_2 = 3.0 * si::meter / si::second;
// This won't compile:
quantity<si::electric_potential> v_tot = v_1 + v_2;
```

- Compiler error:

```
fail_bmas2009_examples.cpp:16: error: no match for 'operator+' in 'v_1 + v_2'
```

↝ Cannot sum voltage quantity v_1 and speed quantity v_2

Boost.Units Dimensional Analysis Examples (2/2)

- Wrong implementation of Ohm's law:

```
quantity<si::resistance> R = 5.0e3 * si::ohm;
quantity<si::electric_potential> v = 10.0 * si::volt;
// This won't compile:
quantity<si::current> i = R * v;
```

- Compiler error (simplified, using namespace boost::units):

```
fail_bmas2009_examples.cpp:10: error: conversion from
'quantity<unit<list<dim<length_base_dimension, static_rational<4l, 1l> >,
                   list<dim<mass_base_dimension, static_rational<2l, 1l> >,
                        list<dim<time_base_dimension,
                                 static_rational<-6l, 1l> >,
                             list<dim<current_base_dimension,
                                      static_rational<-3l, 1l> >,
                                  dimensionless_type> > > >,
                 si::system, void>,
          double>'
to non-scalar type 'quantity<si::current, double>' requested
```

↝ Cannot convert from $[\Omega\,\mathrm{V}] = [\mathrm{m}^4\,\mathrm{kg}^2\,\mathrm{s}^{-6}\,\mathrm{A}^{-3}]$ to $[\mathrm{A}]$

Boost.Units Dimensional Analysis Examples (2/2)

- Wrong implementation of Ohm's law:

```
quantity<si::resistance> R = 5.0e3 * si::ohm;
quantity<si::electric_potential> v = 10.0 * si::volt;
// This won't compile:
quantity<si::current> i = R * v;
```

- Compiler error (simplified, `using namespace boost::units`):

```
fail_bmas2009_examples.cpp:10: error: conversion from
'quantity<unit<list<dim<length_base_dimension, static_rational<4l, 1l> >,
                    list<dim<mass_base_dimension, static_rational<2l, 1l> >,
                         list<dim<time_base_dimension,
                                  static_rational<-6l, 1l> >,
                              list<dim<current_base_dimension,
                                       static_rational<-3l, 1l> >,
                                   dimensionless_type> > > >,
               si::system, void>,
          double>'
to non-scalar type 'quantity<si::current, double>' requested
```

$\rightsquigarrow$ Cannot convert from $[\Omega\,\mathrm{V}] = [\mathrm{m}^4\,\mathrm{kg}^2\,\mathrm{s}^{-6}\,\mathrm{A}^{-3}]$ to $[\mathrm{A}]$

Integrating Boost.Units into SystemC-AMS

1. Parametrize signals and ports to new `quantity<U, V>` type:
   - Supported by DE and SDF MoCs, as ports and signals are templates
   - Example:

```
// Quantity SDF signals
sca_sdf_signal<quantity<si::current> > i_sig;
sca_sdf_signal<quantity<si::electric_charge> > q_sig;
```

2. Keep implemented modules generic enough:
   - Don't reimplement similar behavior for each quantity type
   - ⇝ Write module templates to parametrize ports and parameters
   - Demonstrated with generic module library library `scax_block_diagram`
   - Example:

```
// Integrator instantiation
scax_bd::scax_integ_trapez<quantity<si::current> >
  i_integ("i_integ", 0.0 * si::coulomb);
i_integ.in(i_sig);
i_integ.out(q_sig);
```

Integrating Boost.Units into SystemC-AMS

1. Parametrize signals and ports to new `quantity<U, V>` type:
   - Supported by DE and SDF MoCs, as ports and signals are templates
   - Example:

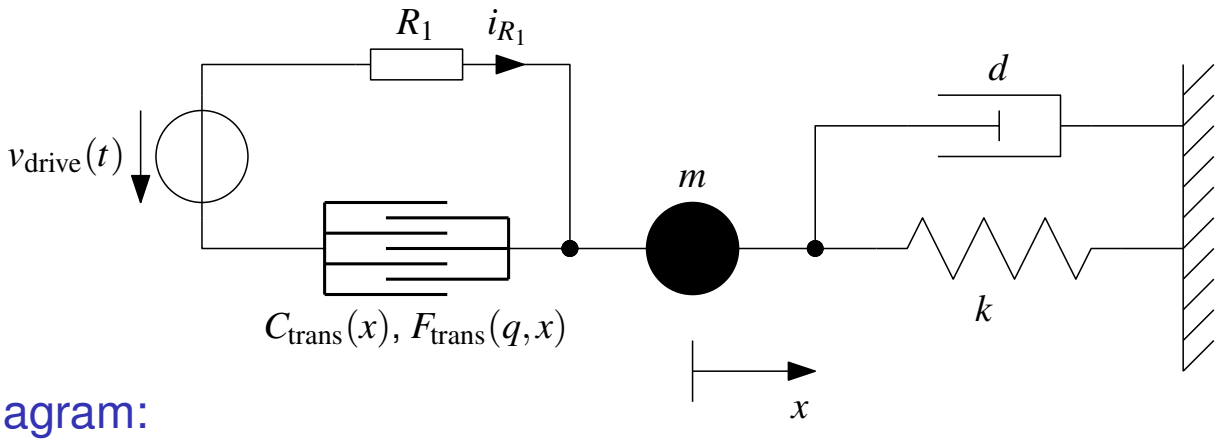
```
// Quantity SDF signals
sca_sdf_signal<quantity<si::current> > i_sig;
sca_sdf_signal<quantity<si::electric_charge> > q_sig;
```

2. Keep implemented modules generic enough:
   - Don't reimplement similar behavior for each quantity type
   - ⇝ Write module templates to parametrize ports and parameters
   - Demonstrated with generic module library library `scax_block_diagram`
   - Example:

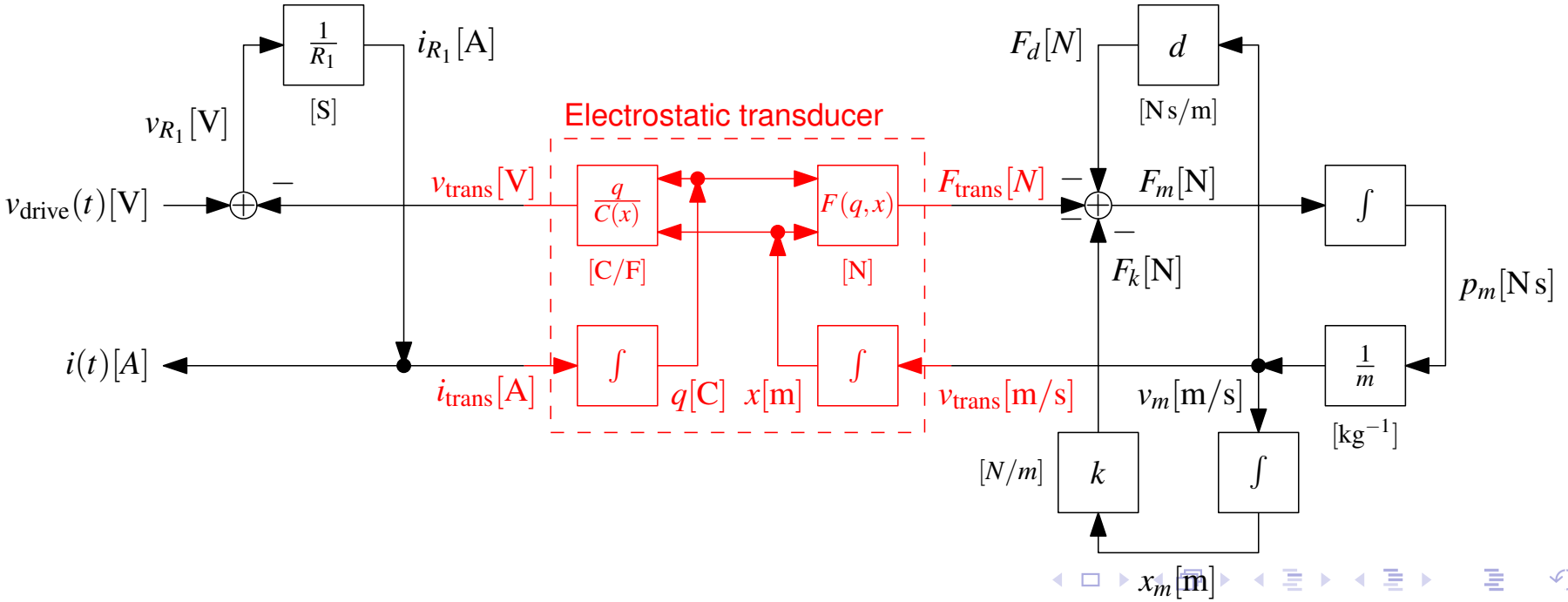
```
// Integrator instantiation
scax_bd::scax_integ_trapez<quantity<si::current> >
  i_integ("i_integ", 0.0 * si::coulomb);
i_integ.in(i_sig);
i_integ.out(q_sig);
```

Electromechanical Transducer with Resonator

Domain-specific conservative model:



Derived block diagram:



Electromechanical Transducer Using double

```
struct elmech_transducer : public sc_core::sc_module {
  // Electrical and mechanical ports
  sca_sdf_in<double> i_in; // Current input [A]
  sca_sdf_out<double> v_out; // Voltage output [V]
  sca_sdf_in<double> v_in; // Velocity input [m/s]
  sca_sdf_out<double> F_out; // Force output [N]
  // Parametrize module
  // v_func: Transducer voltage [V] function v = f(q, x)
  // F_func: Transducer force [N] function F = f(q, x)
  // q_0: Initial charge [C]
  // x_0: Initial displacement [m]
  elmech_transducer(
    const sc_core::sc_module_name& name,
    function<double (double, double)> v_func,
    function<double (double, double)> F_func,
    double q_0 = 0.0,
    double x_0 = 0.0); // ...
};
```

Electromechanical Transducer Using quantity<U, V>

```
struct elmech_transducer : public sc_core::sc_module {
  // Typedefs for common quantity types
  typedef quantity<si::electric_potential> voltage_t;
  typedef quantity<si::force> force_t; // ...
  // Electrical and mechanical ports
  sca_sdf_in<current_t> i_in;
  sca_sdf_out<voltage_t> v_out;
  sca_sdf_in<velocity_t> v_in;
  sca_sdf_out<force_t> F_out;
  // Parametrize module
  elmech_transducer(
    const sc_core::sc_module_name& name,
    function<voltage_t (charge_t, displacement_t)> v_func,
    function<force_t (charge_t, displacement_t)> F_func,
    charge_t q_0 = 0.0 * si::coulomb,
    displacement_t x_0 = 0.0 * si::meter); // ...
};
```

Transducer Voltage Function Defined in Testbench

- Version using `double`:

```
// Transducer capacitance formula v_trans(q, x)
// v_trans: Voltage [V]; q: Charge [F]; x: Displacement [m]
function<double (double, double)>
  v_trans_func = _1 / (C_trans_0 * (1.0 - (_2 / overlap)));
```

↝ Compiler cannot check formula consistency and parameter order!

- Version using `quantity<U, V>`:

```
typedef quantity<si::electric_potential> voltage_type;
typedef quantity<si::electric_charge> charge_type;
typedef quantity<si::length> displacement_type;
// Transducer capacitance formula v_trans(q, x)
function<voltage_type (charge_type, displacement_type)>
  v_trans_func = _1 / (C_trans_0 * (1.0 - (_2 / overlap)));
```

↝ Compiler can fully check formula consistency and parameter order!

Transducer Voltage Function Defined in Testbench

- Version using `double`:

```
// Transducer capacitance formula v_trans(q, x)
// v_trans: Voltage [V]; q: Charge [F]; x: Displacement [m]
function<double (double, double)>
  v_trans_func = _1 / (C_trans_0 * (1.0 - (_2 / overlap)));
```

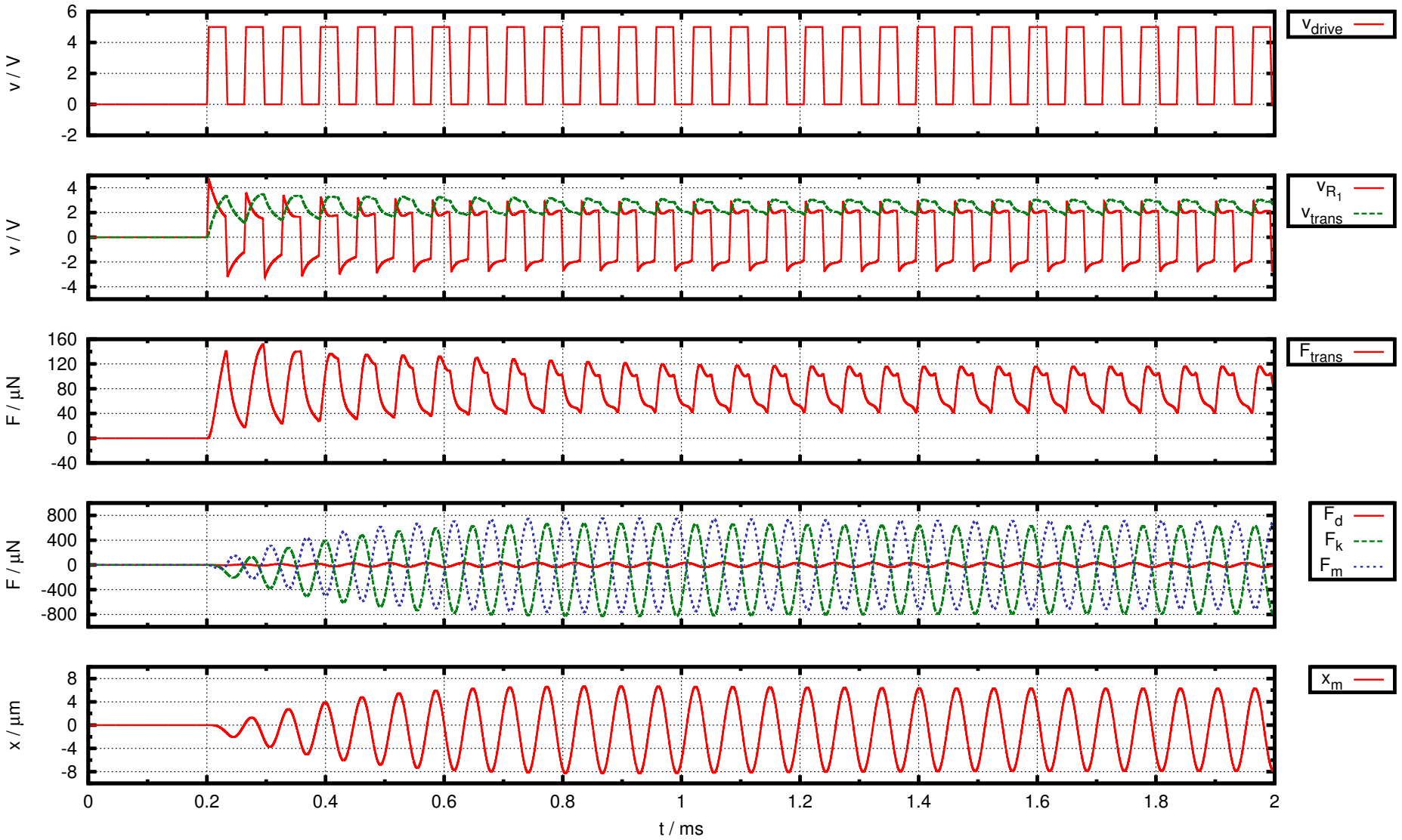
↝ Compiler cannot check formula consistency and parameter order!

- Version using `quantity<U, V>`:

```
typedef quantity<si::electric_potential> voltage_type;
typedef quantity<si::electric_charge> charge_type;
typedef quantity<si::length> displacement_type;
// Transducer capacitance formula v_trans(q, x)
function<voltage_type (charge_type, displacement_type)>
  v_trans_func = _1 / (C_trans_0 * (1.0 - (_2 / overlap)));
```

↝ Compiler can fully check formula consistency and parameter order!

Simulation Results



Compilation and Simulation Times Comparison

Language/Simulator	Feature	$t_{compile}$	$t_{simulate}$
SystemC-AMS 0.15 RC5	`quantity<U, V>`	19.07 s	11.76 s
(+ SystemC 2.2.0, g++ 4.3.2 -O2)	**`double`**	9.69 s	11.38 s
VHDL-AMS	free **`quantity`**	0.79 s	18.76 s
(ADVance MS 2008.2)	branch **`quantity`**	0.84 s	18.43 s

- Simulated time: 2.4 ms
- Fixed time step: 10 ns
- Platform: Intel Pentium 4 3 GHz, 1 MB Cache, 2 GB RAM, Debian Lenny, Linux 2.6.26 (i386)

Conclusions and Outlook

Conclusions:

- Quantities $\rightsquigarrow$ check model assembly and calculation consistency
- Boost.Units $\rightsquigarrow$ increases compilation time, not simulation time
- Generic block diagram library for SDF MoC enables high-level description of analog behavior and keeps link to physical domain

Current work and outlook:

- Scalability of demonstrated approach?
- Filter Boost.Units compiler errors
- Develop Bond Graph MoC applying these techniques

Conclusions and Outlook

Conclusions:

- Quantities $\rightsquigarrow$ check model assembly and calculation consistency
- Boost.Units $\rightsquigarrow$ increases compilation time, not simulation time
- Generic block diagram library for SDF MoC enables high-level description of analog behavior and keeps link to physical domain

Current work and outlook:

- Scalability of demonstrated approach?
- Filter Boost.Units compiler errors
- Develop Bond Graph MoC applying these techniques

References


Boost Library Documentation, 1998-2009.
http://www.boost.org/doc/libs.


Broman, D., *et al.*: *Design considerations for dimensional inference and unit consistency checking in Modelica*.
In *Proceedings of the 6th International Modelica Conference*, Bielefeld, Germany, 2008.


IEEE: *IEEE Standard 1666-2005, SystemC Language Reference Manual*, Mar. 2006.
http://standards.ieee.org/reading/ieee/std/dasc/1666-2005.pdf.


Karnopp, D.C., *et al.*: *System Dynamics: Modeling and Simulation of Mechatronic Systems*.
Wiley, 4th ed., Jan. 2006.


Maehne, T., *et al.*: *Development of a bond graph based model of computation for SystemC-AMS*.
In *Proc. 4th IEEE PRIME 2008 Conference*, Istanbul, Turkey, 2008. IEEE.


Mähne, T., *et al.*: *Creating virtual prototypes of complex MEMS transducers using reduced-order modelling methods and VHDL-AMS*.
In *Applications of Specification and Design Languages for SoCs*. Springer, 2006.


OSCI: *Draft Standard SystemC AMS Extensions Language Reference Manual*, Dec. 2008.
http://www.systemc.org/members/download_files/check_file?agreement=AMS_draft1_120308.

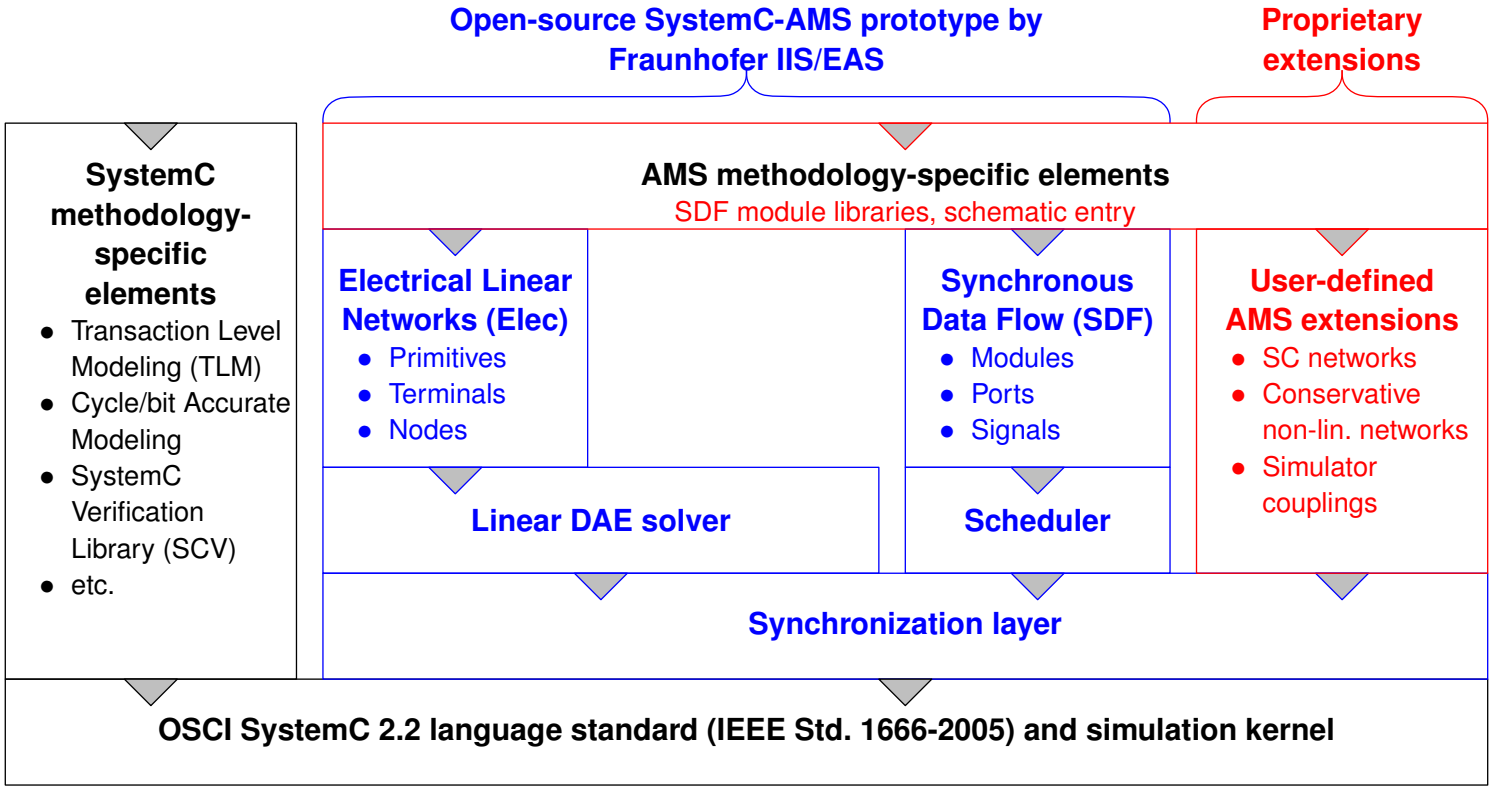

Vachoux, A., C. Grimm, and K. Einwich: *Extending SystemC to support mixed discrete-continuous system modeling and simulation*.
In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2005*.

The End

Thank you for your attention!

AMS Extensions to SystemC

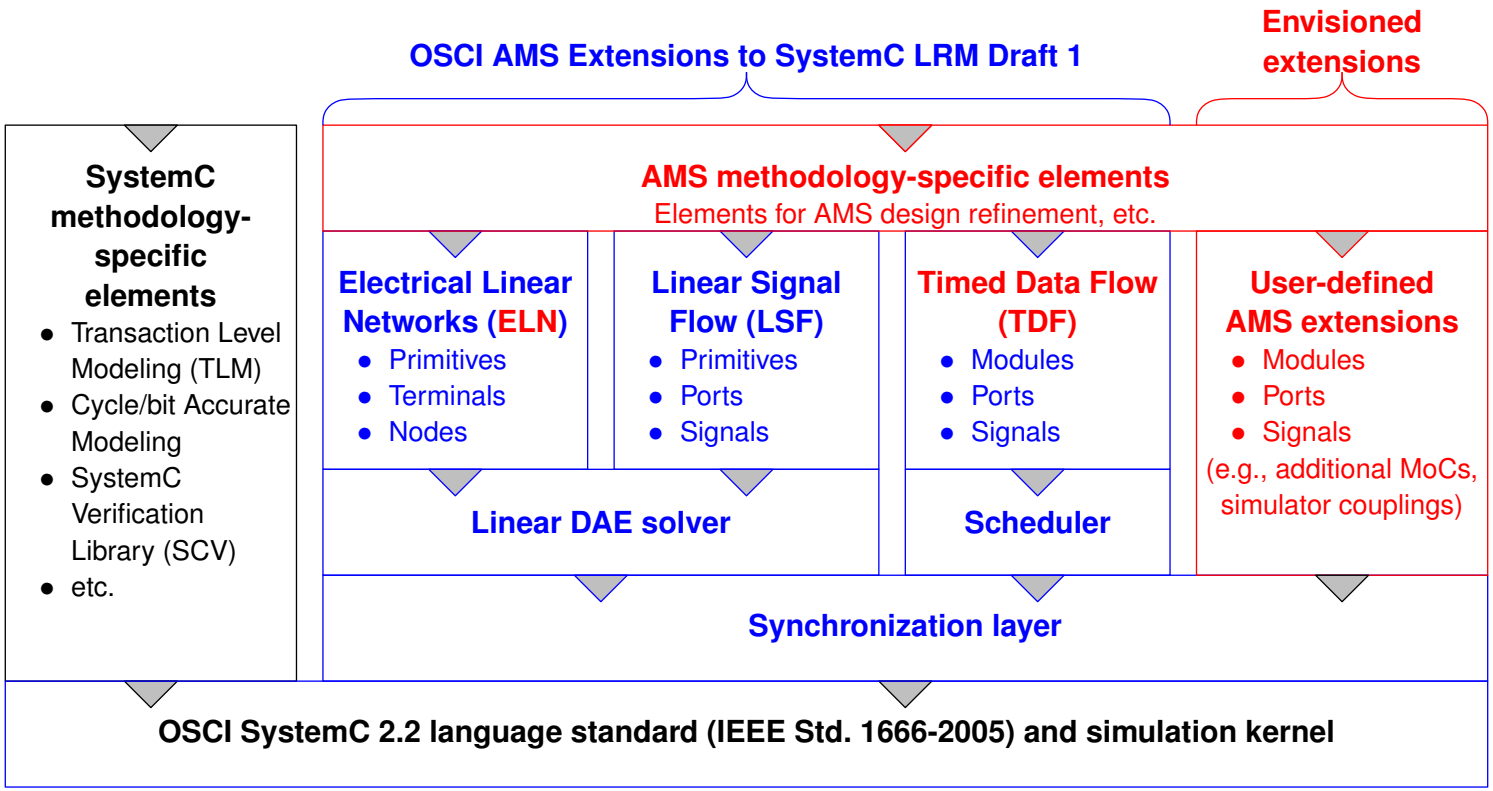
Goal: Simulation of heterogeneous SoCs using several synchronised MoCs
OSCI standardisation effort: based on SystemC-AMS (Fraunhofer IIS/EAS)



- White paper and Draft 1 LRM available
- Proof-of-concept implementation under way driven

AMS Extensions to SystemC

Goal: Simulation of heterogeneous SoCs using several synchronised MoCs
OSCI standardisation effort: based on SystemC-AMS (Fraunhofer IIS/EAS)



- White paper and Draft 1 LRM available
- Proof-of-concept implementation under way driven

scax_block_diagram Library Overview (1/2)

Name	Description
`scax_source<T,TimeType>`	SDF samples source module using a waveform function: f : `TimeType` $\rightarrow$ `T`
`scax_sink<T>`	SDF samples sink module
`scax_scale<T1,T2>`	Scale module
`scax_sum<T>`	Summing module with variable input number
`scax_mul<T1,T2>`	Multiplier module with two inputs
`scax_integ_trapez<T>`	Trapezoidal integrator module
`scax_dot_secant<T>`	Differentiator module using asymmetric evaluation of Newton's difference quotient

scax_block_diagram Library Overview (2/2)

Name	Description
`scax_func1<T1,T2>`	Time-independent function module with one input: $f : \mathtt{T1} \rightarrow \mathtt{T2}$
`scax_func2<T1,T2,T3>`	Time-independent function module with two inputs: $f : \mathtt{T1} \times \mathtt{T2} \rightarrow \mathtt{T3}$
`scax_func3<T1,T2,T3,T4>`	Time-independent function module with three inputs: $f : \mathtt{T1} \times \mathtt{T2} \times \mathtt{T3} \rightarrow \mathtt{T4}$
`scax_func1t<T1,T2,TimeType>`	Time-dependent function module with one input: $f : \mathtt{T1} \times \mathtt{TimeType} \rightarrow \mathtt{T2}$
`scax_func2t<T1,T2,T3,TimeType>`	Time-dependent function module with two inputs: $f : \mathtt{T1} \times \mathtt{T2} \times \mathtt{TimeType} \rightarrow \mathtt{T3}$