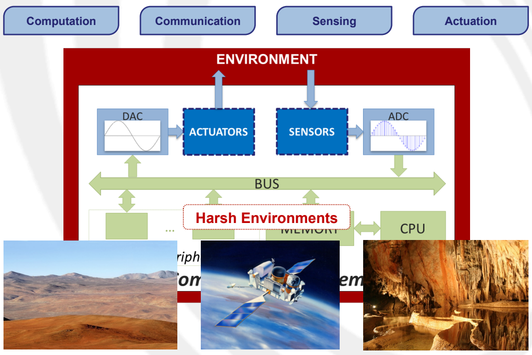
Virtual Prototyping of Smart Systems through *Automatic Abstraction* and *Mixed-Signal Scheduling*

Michele Lora, Enrico Fraccaroli, **Franco Fummi**
Department of Computer Science, University of Verona, Italy

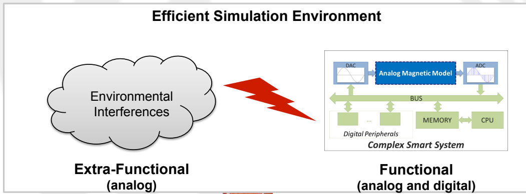
Outline

- Smart systems
- Objectives and motivations
- Guiding Idea:
  - State of the art digital model abstraction
  - An analog model abstraction technique
  - A mixed-signal scheduling methodology
- Analog abstraction
- Mixed-signal scheduling
- Experimental setup and results
- Conclusions

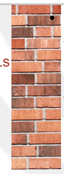
Smart Systems



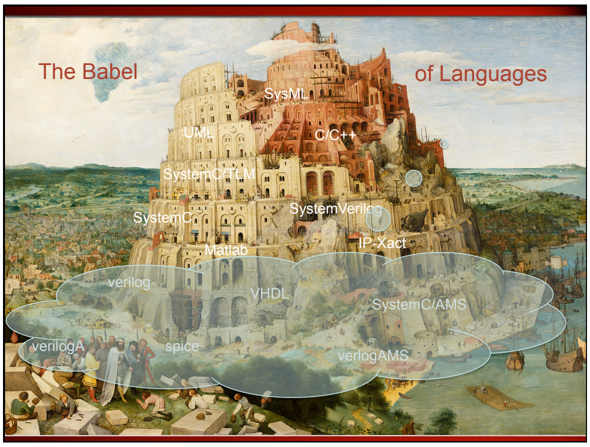
Objectives and Motivations



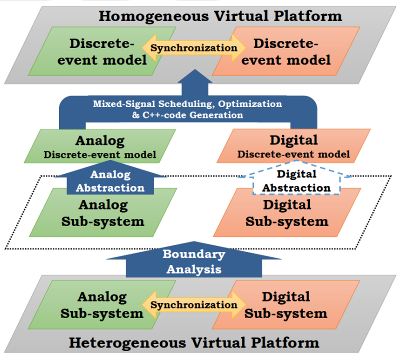
- Analog
  - Ad-Hoc Simulators
  - Analog extensions of HDLs
    - VHDL-AMS
    - Verilog-AMS
    - SystemC-AMS

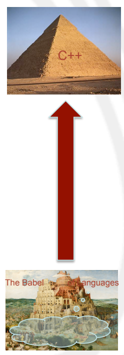


- Digital
  - Hardware Description Languages
    - VHDL
    - Verilog
  - System-Level Description Languages
    - SystemC

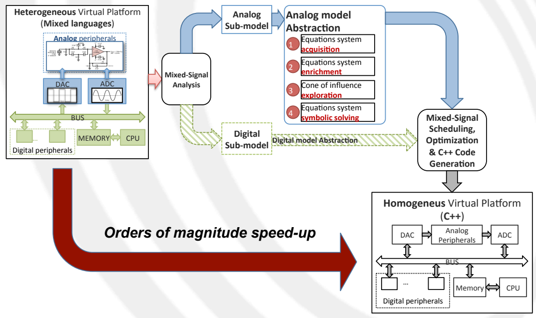


Methodology Overview (1/2)





Methodology Overview (2/2)



Digital Abstraction

- Guiding idea:
  - **RTL models are abstracted up to TLM / C++ code**
- Interface and protocol abstraction:
  - RTL interface is replaced with a TLM (or C++) interface
  - Communication protocol is abstracted so that a single transaction spans through multiple clock cycles
    - N. Bombieri, F. Fummi, G. Pravadelli, Automatic Abstraction of RTL IPs into Equivalent TLM Descriptions, *IEEE Transactions on Computers*, vol. 60, n. 12, 2011, pp. 1730-1743
- Data types abstraction :
  - HDL data types are mapped to C++ native data types
    - N. Bombieri, F. Fummi, V. Guarnieri, F. Stefanni, S. Vinco, HDTLib: an efficient implementation of SystemC data types for fast simulation at different abstraction levels, *An International Journal on Design Auto. for Emb. Systems*, vol. 16, n. 2, 2012, pp. 115-135
- Process scheduler abstraction:
  - HDL processes are abstracted to C++ functions by implementing static / dynamic and mixed scheduling policies
    - S.Vinco, V.Guarnieri, F.Fummi, Code Manipulation for Virtual Platform Integration, *IEEE Transactions on Computers*, vol. 65, n. 9, 2016, pp. 2694-2708

EDALab **HIFSuite** tools

Analog Abstraction (1/7)

- Guiding idea:
  - Abstraction from **Circuit Level** to **Functional Level**

Level	Modeling Primitives	Implications
Functional	Mathematical signal flow description per block, connected in signal flow diagram	No internal block structure; conservation laws need not be satisfied on pins
Behavioral	Mathematical description (equations, procedures) per block	No internal block structure; conservation laws must be satisfied on pins
Macromodel	Simplified circuit with controlled sources	Spatially unrelated to actual circuit; conservation laws must be satisfied
Circuit	Connection of SPICE primitives	Spatially one-to-one related to actual circuit; conservation laws must be satisfied

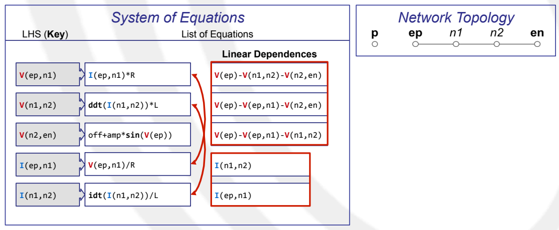
L. Scheffer, L. Lavagno, and G. Martin, *EDA for IC Implementation, Circuit Design, and Process Technology*. CRC Taylor & Francis, 2006.

  - Consider only relations with **semantic** meanings between a **value of interest** and the **inputs** of the model

Analog Abstraction (2/7)

Equation system enrichment

- Find relations between circuit branches by
  - solving each of the initial equations
  - applying Kirchhoff's flow law (KFL) and Kirchhoff's potential law (KPL)
- Some of these equations are in a linear dependency relation

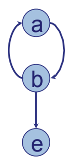


Analog Abstraction (3/7)

Cone of Influence Exploration

- Purpose: Value of Interest → Inputs of the Model
- Enriched system of Equation → Graph representation
  - A labeled node is associated to each equation
  - An edge connects a node A to a node B whenever
    - the lhs variable of the equation associated with the second node(B)
    - appears on the rhs of the equation associated with the first node(A)

```
V(ep,n1) = I(ep,n1)*R                  a
I(ep,n1) = V(ep,n1)/R                  b
V(n1,n2) = ddt(I(n1,n2))*L             c
I(n1,n2) = idt(V(n1,n2))/L             d
V(ep,n1) = V(ep)-V(n1,n2)-V(n2,en)     e
V(n1,n2) = V(ep)-V(ep,n1)-V(n2,en)     f
V(n2,en) = V(ep)-V(ep,n1)-V(n1,n2)     g
I(ep,n1) = I(n1,n2)                    h
I(n1,n2) = I(ep,n1)                    i
V(n2,en) = off+amp*sin(V(ep))          j
V(ep)                                  k
```



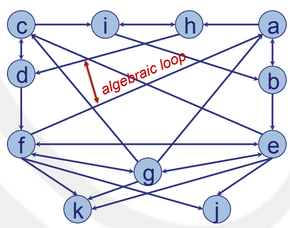
Analog Abstraction (4/7)

Cone of Influence Exploration (contd.)

- Select the minimum set of equations describing the Behavior of the Value of Interest
  - **Start**: Equation describing the Value of Interest
  - Equation of a visited node is stored inside the minimum set and the node is disabled
  - **End**: All the nodes are disabled

"Whenever an equation belonging to a linearly dependent set is selected, the nodes associated with the other equations of the set are disabled"

```
Value of interest
V(ep,n1) = I(ep,n1)*R                  a
I(ep,n1) = V(ep,n1)/R                  b
V(n1,n2) = ddt(I(n1,n2))*L             c
I(n1,n2) = idt(V(n1,n2))/L             d
V(ep,n1) = V(ep)-V(n1,n2)-V(n2,en)     e
V(n1,n2) = V(ep)-V(ep,n1)-V(n2,en)     f
V(n2,en) = V(ep)-V(ep,n1)-V(n1,n2)     g
I(ep,n1) = I(n1,n2)                    h
I(n1,n2) = I(ep,n1)                    i
V(n2,en) = off+amp*sin(V(ep))          j
V(ep)                                  k
Inputs of the Model
```



Analog Abstraction (5/7)

Equation System Symbolic Solving

- Purpose
  - Optimizes the model and breaks all algebraic loops
- How
  - Symbolically solve the minimal set of equations
  - Symbolic solver: GiNaC
- Before solving the system
  1. Access function are transformed into a C++ compliant naming
  2. Derivatives and Integrals are discretized (accuracy ↔ techniques)

```
V(ep,n1) = I(ep,n1)*R
I(n1,n2) = idt(V(n1,n2))/L
V(n1,n2) = V(ep)-V(ep,n1)-V(n2,en)
I(ep,n1) = I(n1,n2)
V(n2,en) = off+amp*sin(V(ep))
V(ep)
```

```
V_ep_n1 = I_ep_n1*R
I_n1_n2 = idt(V_n1_n2)/L
V_n1_n2 = V_ep-V_ep_n1-V_n2_en
I_ep_n1 = I_n1_n2
V_n2_en = off + amp*sin(V_ep)
V_ep
```

```
I_n1_n2 = Accum_V_n1_n2/L
```

`Accum_V_n1_n2` is called **auxiliary** variable

Analog Abstraction (6/7)

Equation System Symbolic Solving (contd.)

- The system of equations is composed by
  - **the circuit equations**
- and is solved for
  - the **Value of Interest** and **variable used to evaluate** the auxiliary variables

System of Equations

```
V_ep_n1 = I_ep_n1*R
I_n1_n2 = Accum_V_n1_n2/L
V_n1_n2 = V_ep-V_ep_n1-V_n2_en
I_ep_n1 = I_n1_n2
V_n2_en = off + amp*sin(V_ep)
V_ep
```

Unknowns

```
V_ep_n1
V_n1_n2
```

$$lhs = \sum_i (auxiliary_i * const_i) + \sum_j (input_j + const_j)$$

Analog Abstraction (7/7)

Abstracted Model Generation

- The solution is used to generate an **abstracted C++ simulation model**
- It contains
  - An **assignment** for the Value of Interest
  - An **assignment** for each Value used to update an Auxiliary Variable
  - An **update assignment** for each Auxiliary Variable

Solved Equations

```
V_ep_n1 = Accum_V_n1_n2*C1 + V_n2_en*C2 + V_ep*C3
V_n1_n2 = Accum_V_n1_n2*C4 + V_n2_en*C5 + V_ep*C6
V_n2_en = off + amp*sin(V_ep)
```

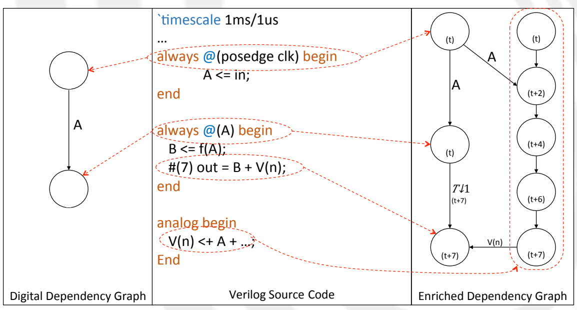
```
class Model {
  public:
    void process(double Ts) {
      V_ep_n1=…;
      V_n1_n2=…;
      // Update auxiliaries
      Accum_V_n1_n2   = …;
    }
  private:
    double R=…, L=…, amp=…, off=…;
}
```


Mixed-signal Scheduling (1/3)

- Guiding idea:
  - **Digital abstraction produces C++ functions**
  - **Analog abstraction produces C++ functions**
    - **synchronization and scheduling of these C++ functions is necessary to correctly reproduce the behavioral evolution of the initial AMS description**
- Dependency graph enrichment
  - dependencies between synchronous processes are modelled
  - enriched with the exact timing of asynchronous processes that have to be synchronized
- Temporal decoupling
  - edges among processes are a partial order relation
  - not related processes are decoupled and simulated independently
- Synchronization
  - the global time is managed accordingly to the dependencies

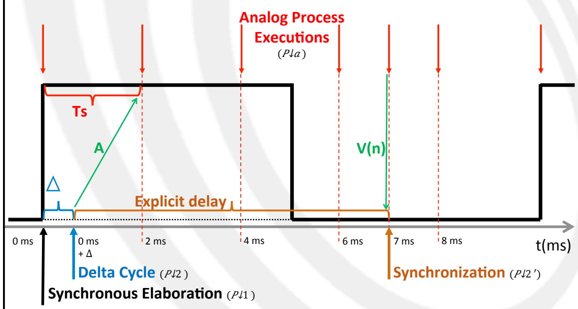
Mixed-signal Scheduling (2/3)

- Enriched dependency graph construction



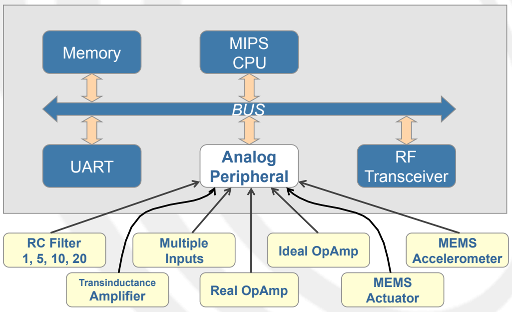
Mixed-signal Scheduling (3/3)

- Temporal decupling and synchronization



Experimental Platform

- A smart platform (OSTC) composed of analog and digital components:



Experimental Results (1/3)

- Proposed methodology has been implemented
  - in **OCCAM** (Ordinary C++-Code for Analog Models)
  - on top of EDALab's **HIFSuite** framework

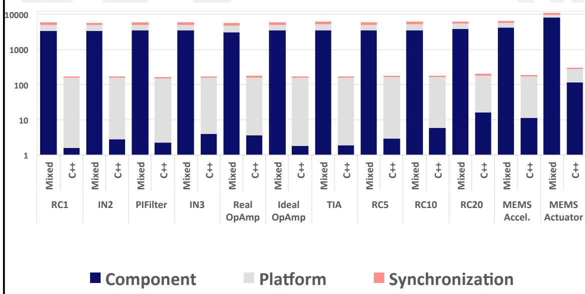
Benchmark	Relations	Inputs	Outputs	Lines of Code	Abstraction Time (s)
RC1	2	1	1	17	0.009
IN2	3	2	1	21	0.012
PIFilter	4	1	1	21	0.008
IN3	5	3	1	31	0.015
Voltage Limiting Operational Amplifier	2	1	1	33	0.007
Ideal Operational Amplifier	6	1	1	31	0.009
Transimpedence Amplifier	3	1	1	33	0.007
RC5	10	1	1	42	0.014
RC10	20	1	1	67	0.026
RC20	40	1	1	117	0.078
MEMS Accelerometer	66	10	8	123	0.020
MEMS Mechanical Actuator	28	1	1	408	0.084

Experimental Results (2/3)

Benchmark	Heterogeneous (Analog: Verilog-AMS)		SystemC-AMS/ELN (ABACUS automatic translation)				C++ (OCCAM automatic abstraction)			
	Component	Platform	Component		Platform		Component		Platform	
	time (s)	time (s)	time (s)	speed-up (x)	time (s)	speed-up (x)	time (s)	speed-up (x)	time (s)	speed-up (x)
RC1	3365.87	4972.86	20.67	162.84	1457.66	3.41	1.59	**2120.11**	154.73	**32.14**
IN2	3419.13	4934.58	25.23	135.51	1558.97	3.17	2.77	**1234.34**	157.02	**31.43**
PIFilter	3475.80	4990.13	31.14	111.61	1599.70	3.12	2.27	**1533.95**	153.34	**32.54**
IN3	3575.54	5070.32	31.25	114.43	1692.64	3.00	4.06	**879.83**	156.90	**32.32**
Voltage Limiting Operational Amplifier	3138.62	4712.55	Not applicable due to nonlinearities				3.62	**867.29**	159.73	**29.50**
Ideal Operational Amplifier	3499.48	5024.72	30.21	115.83	1599.52	3.14	1.77	**1982.30**	156.42	**32.12**
Transimpedence Amplifier	3438.67	5114.33	Not applicable due to nonlinearities				1.91	**1801.78**	154.92	**33.01**
RC5	3438.38	5051.94	43.46	79.11	1722.90	2.93	2.87	**1200.03**	160.61	**31.45**
RC10	3584.68	5165.78	76.15	47.07	2076.72	2.49	5.86	**612.00**	163.39	**31.62**
RC20	3879.56	5358.37	129.34	29.99	2631.01	2.04	16.36	**237.18**	177.93	**30.12**
MEMS Accelerometer	4196.61	5723.16	181.80	23.08	3936.65	1.45	11.60	**361.76**	168.78	**33.91**
MEMS Mechanical Actuator	8078.35	9769.48	Not applicable due to nonlinearities				114.64	**70.47**	275.53	**35.46**

Experimental Results (3/3)

- Relative impacts of the proposed techniques



Conclusions

- We presented a methodology for effectively generating Smart Systems virtual platforms through:
  - digital models abstraction
  - analog models abstraction
  - mixed-signal scheduling
- Homogeneous C++ virtual platforms can be thus generated
  - orders of magnitude **simulation speed-up** with respect to state-of-the-art mixed-signal language simulators

The **simulation barrier** between analog and digital models has been **sensibly reduced**