A Hierarchical Design Tool for SystemC AMS

Daniela Genius[1] and Ludovic Apvrille[2]

Sorbonne Universités, UPMC Paris 06, LIP6, CNRS UMR 7606, Paris, France,
LTCI, Télécom Paris, Institut Polytechnique de Paris, France,
daniela.genius@lip6.fr, ludovic.apvrille@telecom-paris.fr

Abstract. The splitting into functionalities and interaction between analog and digital parts should be considered as early as possible in the design phase. We extended this methodology in former work, in order to take cyber-physical systems, based upon analog / digital hardware and software components, into account. The Models of Computation in these systems are not frequently tackled by many researchers due to their significant differences. However, as we show in this article, a SysML model can be directly used to generate a virtual prototype representing both the analog and mixed-signal parts of cyber-physical systems. For this, we rely on a hierarchical methodology, as certain analog components demand highly detailed designs. In this study, we introduce the capability to perform extensive semi-automatic design space exploration for these parts as well. A way to achieve this is to use parameters to automatically adapt the number of subsystems considered in the prototype.

Keywords: Cyber Physical Systems, Analog/Mixed Signal Design, Virtual Prototyping

1 Introduction

Cyber-physical systems (CPS) are built upon analog / digital hardware and software components. Most often, off-the-shelf components can be used. Yet, when more specific requirements have to be met (such as: low-power, very small size, specific application, cost, etc.) off-the-shelf components may be too costly or not confirming the specific requirements, thus advocating for (partial) from-scratch designs, sometimes both for analog and digital parts.

For a custom design, the splitting of functionalities between analog and digital parts, and how they shall interact, is of prime importance, and should therefore be done as early as possible in the design phase. In early design phases, simulation or formal verification helps taking design decisions. But, in the case of mixed designed, i.e., designs with analog and digital parts, the Models of Computation (MoC) of these two aspects strongly differ. Moreover, they are commonly designed at different abstraction levels, depending on the design patterns already available. Last, because of the significant semantic difference between MoCs, similar models or tools can generally not be used to study the same system.

Thus, an interesting contribution is to offer a multi-level co-simulation of analog and digital aspects that can be executed from the same input models (in our case: SysML models), and featuring all necessary elements to capture analog and digital aspects. Abstraction levels should range from a near-circuit precision to more abstract analog and digital models (e.g., transactional models). In order to achieve this goal, we decided to rely on the multi-level design tool TTool [3].

Our previous contributions [20, 19] already highlighted how to jointly express in SysML analog and digital parts, and how to provide tool support for these features. in particular, we explained how SysML diagrams can efficiently capture digital and analog parts of CPS as well as their interactions. Moreover;, we have shown how to generate a virtual prototype of the analog and mixed-signal parts directly from these

SysML models. Note that TTool already offers extensive verification capabilities for the digital parts, but this is out-of-scope of this paper.

The most recent version of TTool-AMS combines three simulation semantics. For the digital part of the system, a Discrete Event (DE) MoC is used. In this MoC, the operation of a system is driven by a discrete sequence of events in time, each event changing the state of the system. For the analog part, we use a dataflow model with differential algebraic equations.

While previous contributions could be used to model and simulate mixed-signal systems, as shown in [18], the present contribution enhances system design with full semi-automatic design space exploration targeting analog/mixed signal systems. Indeed, for more advanced use cases, it is necessary to represent replicated resources in a compact way, and to vary the number of subsystems consisting of several electronic elements by a simple change of an integer parameter, for example when increasing the number of bits to increase the resolution of an analog-digital converter.

Section 2 presents related work of the two domains we cover: model-based design for cyber-physical systems and analog/mixed signal hardware design. Basic concepts of SystemC AMS are introduced in Section 3. Our contribution starts with the introduction of a model-based approach to SystemC AMS Electrical Linear Network Design (Section 4), and illustrated throughout the section by a toy example. The aforementioned approach is extended in Section 5 to handle multiple ports and replicated modules, so as to enable design space exploration. Section 6 shows a larger case study, an analog-to-digital converter where the number of control bits is scalable. We conclude by giving perspectives on future work.

2 Related Work

Our contribution bridges the gap between two research domains: model-based design methodologies for cyber-physical systems and the field of analog/mixed-signal hardware design.

2.1 Model-based design for cyber-physical systems

Embedded systems can be designed using different Models of Computation, with different notions of concurrency and time [26]. The most well-known tool in analog/mixed signal design is *Ptolemy II* [27] of the university of Berkeley, based upon a data flow model. Heterogeneous systems are addressed by defining several sub domains, instantiating elements controlling the time synchronization between domains – this part however is left to the responsibility of the designer.

Metro II [9] uses *Adaptors* for data synchronization between components belonging to different MoCs. The model designer however still has to implement time synchronization. As a common simulation kernel handles all process execution, but the MoCs are not well separated.

Modelica [15] is an object-oriented modeling language for component-oriented systems containing e.g. mechanical, electrical, electronic and hydraulic components. Modelica classes contain a set of equations that can be translated into objects running on a simulation engine. Yet, since time synchronization is not predefined, the simulation engine must manipulate objects in a symbolic way in order to determine an execution order between components of different MoCs.

Linking simulation with different MoCs can be done by using the Functional Mockup Interface [6]. Yet, in our former work [18], we nee to take into account three MoCs from initial high-level design onwards, breaking the system design into three levels from digital over abstract analog/mixed signal to circuit level.

Discrete Event System Specification (DEVS [7]) is a modular and hierarchical formalism which supports discrete events and continuous systems. Continuous functions can be described by differential equations. A dozen of platform implementations based on DEVS exist, ranging from Petri Net based over object oriented

to Python based. However, DEVS relies on a global homogeneous time. In that sense, DEVS approach is probably the most similar; it does however not generate SystemC based prototypes capable of full-system simulation.

Las but not least, UML/SysML based modeling techniques such as MARTE [10] and Gaspard2 [16] have been employed to model cyber-physical systems [36], but, with few exceptions [39, 29], they do not support refinement until a low level of abstraction nor provide full-system simulation.

Into-CPS [14] uses model-based formal methods by integrating discrete-event models of controllers with continuous-time models of their environments. Starting from a discrete-event model, approximations of continuous-time behavior are subsequently replaced by couplings to continuous-time models.

Another extension to the SDF formalism is called Polygraph, which includes frequency constraints and adjustable communication rates and ensures synchronization [11].

TTool [4], an open-source modeling and verification framework, provides to some extend analog/mixed signal modeling and virtual prototyping [20]. The determination of the schedule and causality between analog and digital parts of the system is based on timed Synchronous Data Flow (SDF) [28]. TTool already permits the generation of SystemC based virtual prototypes and has been extended with prototype generation from System C AMS TDF models. It is thus this last one which we choose to enhance by circuit design capabilities.

2.2 Analog/Mixed signal hardware design based on SystemC

The following tools target analog/mixed signal or multi-domain design and virtual prototyping based on SystemC [25], with or without alteration of the simulation kernel which was initially targeted only toward discrete event simulation.

Frameworks based on SystemC include HetSC [23], HetMoC [42] and ForSyDe [31], all having the disadvantage that instantiation of elements and synchronization control is totally left to designers.

SystemC-H [32] and SystemC-A [41] extend the SystemC simulation kernel. The former allows only one hierarchical level in the description of models, and execution is based on a master-slave relation: a modified discrete event simulation kernel initializes and simulates the processes described by means of different MoC-specific kernels. The SystemC scheduler preserves the initialization, evaluate, update, delta and time notification phases, but some of them are modified. Synchronization mechanisms are not available among components defined under different models of computation.

In SystemC-A, which allows hierarchy, the scheduler calls the analog kernel phases (iteration and verification) before SystemC scheduling. The independent analog kernel is able to synchronize with the DE simulation kernel.

SystemC AMS extensions [1] is a standard describing an extension with AMS and RF features [40]. They predefine several Models of Computation (MoC). In the scope of the project BeyondDreams [5], a mixed analog-digital systems proof-of-concept simulator has been developed [13], based on the SystemC AMS extension standard.

Another simulator, Multi Domain Virtual Prototyping (MDVP) is proposed in the H-Inception project[22]. The simulation phases of SystemC are not modified here. A major result of that work is that causality issues can be automatically checked *before* simulation [2].

Pushing the issue of handling causality and its interaction with scheduling event further, and using the above mentioned TTool framework for hardware and software modeling, is is now possible to detect and solve causality issues on the level of SysML design [8]. This approach, integrated in TTool under the name TTool-AMS, is based on the classical SystemC AMS simulation mechanism.

3 Introduction to SystemC-AMS modeling

SystemC AMS extensions [1] is an extension of SystemC with AMS (Analog/Mixed Signal) and RF (Radio Frequency) features [40]; several Models of Computation are predefined. A proof-of-concept implementation [13] evolved into an industrial tool [12] that also handles validation of hardware against software and generates Simulink or C Code.

Digital components are described by a Discrete Event (DE) MoC, while analog components follow the Timed Data Flow (TDF) MoC, based on the timeless Synchronous Data Flow semantics [28]. The most low-level MoC is called Electrical Linear Network (ELN). It relies on equations to capture the behavior of electrical circuits in a simplified way.

For the modeling of digital hardware/software systems (e.g., micro controllers and processors), interconnect and communication protocols, SystemC (DE) modeling approaches are most suitable. Signal processing functionality should be modeled by Linear Signal Flow. If sampling rates and time steps need to be described, either statically or dynamically, the TDF MoC is chosen. If non-linear elements such as diodes and transistors have to be precisely modeled, it is preferable to resort to Spice [34]. In combination with SystemC, SystemC AMS thus covers all of the above MoC except Spice. At the moment, our approach does not accommodate nonlinear models or Linear Signal Flow (LSF).

3.1 Discrete Event

A Discrete-Event (DE) simulation abstracts a system as a discrete sequence of events in time, where each event signals a change of state. This stands in contrast to continuous simulation, where the state of the system undergoes constant changes over time. DE modules are depicted with white rectangles in SystemC AMS. DE modules have input and output ports, represented as white squares. Last but not least, DE modules can contain SystemC code.

3.2 Timed Data Flow

Timed Data Flow (TDF), which is based on the timeless Synchronous Data Flow (SDF) semantics [28], samples continuous functions at discrete intervals. A TDF *module* is described with an attribute representing the time step and a processing function, a mathematical function depending on the module inputs and/or internal states.

Figure 1 shows a graphical representation of a TDF cluster, as defined in the SystemC AMS standard. DE modules are represented as white blocks, TDF modules as gray blocks, TDF ports as black squares, converter ports connecting the TDF and the DE domain as black and white squares, and finally TDF signals as arrows.

TDF modules have the following attributes:

1. Module time step (**Tm**) denotes the period of time during which the module is activated. A module is activated only if there are enough samples available at its input ports.
2. Rate (**R**). Each module reads or writes a fixed number of data samples each time it is activated, annotated to the port as port rate.
3. Port time step (**Tp**) denotes the time interval between two operations (read or write).
4. Delay (**D**). A delay can be assigned to a port and will make the port handle a fixed number of samples at each activation, and read or write them in the following activation of the port.

At each time step, a TDF module follows a structured process: initially, it reads a predetermined number of samples from its input ports, then it executes its processing function, and ultimately, it writes a predetermined number of samples to its output ports.

Figure 1, taken from the example given in the System C AMS standard notation [1], shows a *cluster* built of several modules. The DE module Y is represented as a white block, TDF modules A and B as gray blocks. TDF ports are black squares, TDF converter ports black and white squares, DE ports white squares and signals are represented as arrows. There are TDF ports between modules A and B, and B sends its output to a converter port. For the TDF modules, port rates, delays, timesteps, and module timesteps are all specified. For instance, the module timestep of A is 6 ms, its output port time step is 2 ms and rate 3. B has an input port time step of 2 ms and rate of 2, and module time step of 4ms, rate of 1 and time step of 4 ms in the output port, respectively. The output port of B is a *converter port*.

Schedulability denotes the correct and static execution-order of TDF modules in one cluster. A cluster is *schedulable* if the module time step is consistent with the rate and time step of any port within a module. Le T_M denote the module time step, T_{pi} and T_{po} respectively denote the input and output port time steps, R_{pi} and R_{po} respectively denote the input and output port rates:

$$T_M = T_{pi} \times R_{pi} = T_{po} \times R_{po}$$

Before the static schedule of a cluster can be computed, the *time steps* and *sampling rates* that are not indicated in the model need to be calculated by upstream and downstream propagation, as explained in [1]. In the example shown above, the schedule is ABABB: module A is activated, followed by B, again A and finally two times B.

TDF clusters run continuously according to their schedule, proposing values on their converter ports to the interface to/from the DE modules on their converter ports.

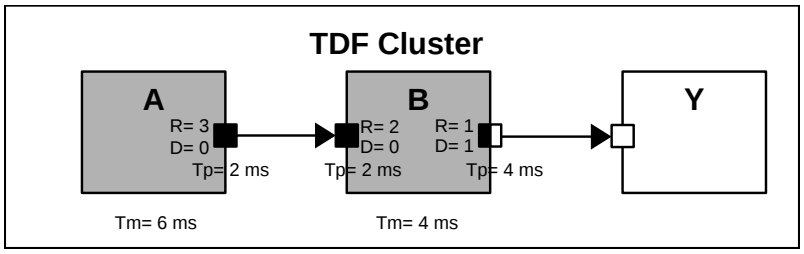

Fig. 1: TDF Cluster

3.3 Electrical Linear Networks

An ELN module gives a detailed representation of an electrical circuit. The Electrical Linear Networks (ELN) model of computation introduces the use of electrical primitives and their interconnections to model conservative, continuous-time behavior. ELN primitive modules such as resistors, capacitors or voltage/current sources are connected via *terminals* (positive or negative) —represented as circles— to *nodes* —represented as straight lines— and form an ELN *cluster*. The mathematical relations between the electrical primitives are defined at each node in the network. Equations of each primitive element are then combined

into an *equation system*, where both the potential (voltage) and flow (current) quantities are used according to Kirchhoff's laws. The electrical network is thus represented by a set of differential algebraic equations that can be taken into account at simulation.

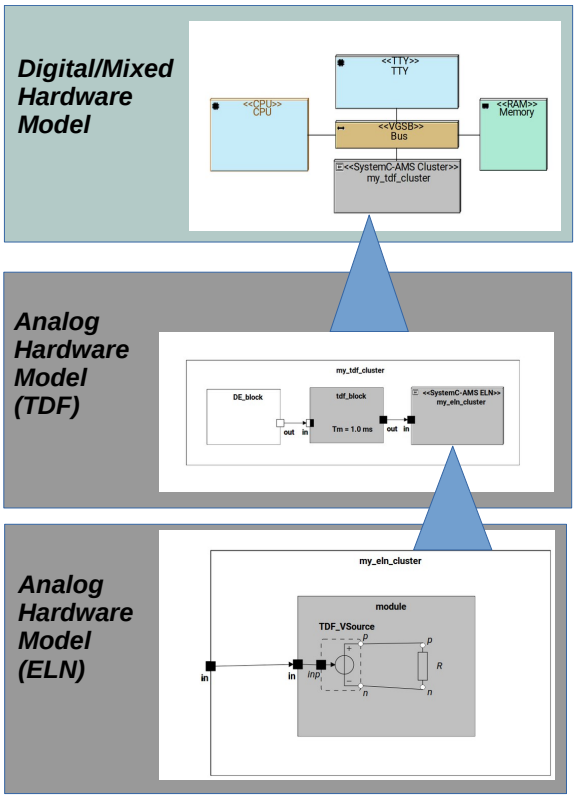

Fig. 2: Overview of the hierarchical method shown in [18])

SystemC AMS extensions offer a limited set of primitive modules: voltage or current sources, linear lumped elements (resistors, capacitors, inductors), transmission lines, ideal transformers and amplifiers, linear gyrators, and ideal switches. Unlike for TDF models, there is no possibility to implement user-defined electrical primitives. Non-linear behavior however cannot be represented. As a consequence, nonlinear elements such as diodes and transistors must be approximated with existing linear components.

Figure 3 depicts a module with only ELN elements, featuring a low-pass filter. The example is taken from [1] and redesigned with TTool. Even though the equation system can be solved, it's crucial to note that such a module must be encapsulated into a TDF module, which is then governed by its timesteps.

3.4 Combining and simulating several MoC in SystemC AMS

Converter *ports* are required to connect DE components to TDF components, and reciprocally. Converter *modules* can connect TLM components to TDF or DE modules. The timing and consistency issues between

their different MoC, in particular between TDF and DE, are tricky to handle, since situations where either the DE or the TDF simulation runs ahead must be avoided. Therefore, a correct schedule has to be calculated accordingly. Moreover, in some cases, additional delays have to be introduced [8, 2].

The mathematical equations for each ELN primitive module and their relationships defined at each node are part of the overall equation system: during simulation, these equation systems are solved numerically at appropriate time steps.

For ELN models connected to a TDF model, the time step from the connected TDF port(s) is propagated to the ELN model. Consistency between locally defined ELN module time steps and propagated time steps is checked by SystemC AMS. Otherwise, the time points for the solution of the ELN equation system or communication with the connected TDF model cannot be defined properly.

For ELN modules, a time step can be directly assigned to modules or propagated using the mechanism of the time step within an ELN equation system. In case an ELN model is connected to a TDF model, the time steps from the connected TDF ports are propagated to the ELN model.

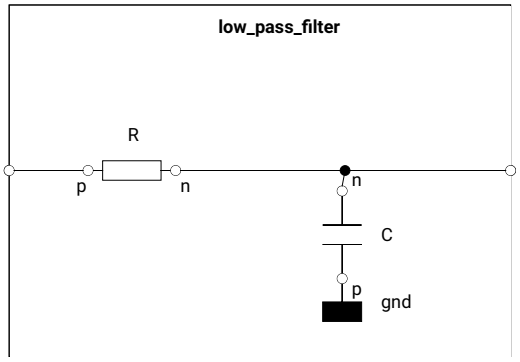

Fig. 3: Example ELN cluster (capture from TTool)

4 Hierarchical modeling of analog hardware components

In the following, we highlight our contribution from [18], where we stated that the Timed Data Flow (TDF) model of computation (the higher abstraction level) is sometimes insufficient when dealing with highly innovative circuit designs that require an important level of detail. As interactions with the environment are to be studied precisely as soon as possible, i.e., before the actual design is complete, we show how to efficiently use TDF descriptions for that purpose. This can be done in a top-down, hierarchical manner, using a customized SysML meta-model, as shown in this section.

Both [8] and [2] propose a simulation environment for SystemC-AMS where the simulation of DE components controls the TDF simulation. It seems logical that the TDF simulation should in turn control the ELN simulation. Adhering to this simulation hierarchy, we propose a three-level modeling. This approach permits designers to fluidly transition between levels, each represented by distinct diagrams encapsulating analog/mixed-signal hardware. The diagram at each successive level offers a bird's eye view of the lower one, abstracting certain components that have been modeled at a lower level for representation at the current level.

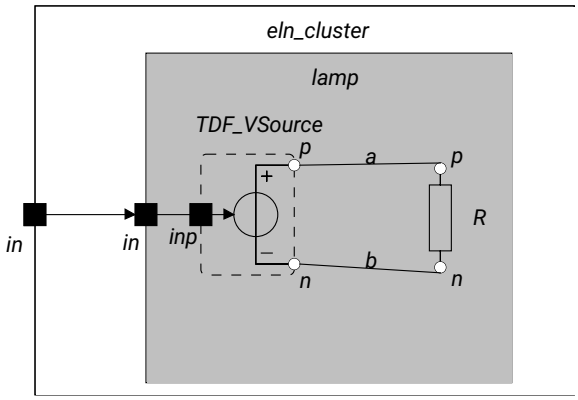

Fig. 4: TDF cluster encapsulating an ELN diagram (from [18])

Figure 2 gives an overview the hierarchical method we suggest. Our approach applies once a hardware/-software partitioning has been accomplished, i.e. digital functions of the systems have been split between hardware and software. The top part of the Figure displays a UML Deployment Diagram, which contains an analog cluster depicted as a grey UML node. The other nodes correspond to the digital parts of the Virtual Prototype: CPU, memory, bus and a terminal to support debugging. All these digital nodes have known translations to SystemC models in the free SoClib library [38]. The middle part of the Figure shows a zoom into the grey box (opened with a double-click in the top box). Last, the lowest hierarchical level is dedicated to the refinement of some of the modules in ELN representation.

4.1 Introductory example

Figure 5 shows a combination of a DE, a TDF and an ELN module, from left to right in a strongly simplified home automation system. SysML blocks are used to abstract a light bulb supplied with a voltage controlled by a dimmer, which in turn is controlled by the—digital—microcontroller of a home automation system. The DE module has one DE output port; the TDF module features a DE-to-TDF converter port on its left hand side and a TDF output port on the right hand side. The latter is connected to the TDF input port of the ELN module.

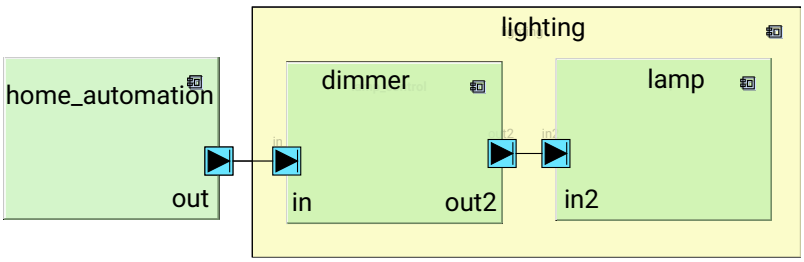

Fig. 5: Functional model of the lighting system (from [18])

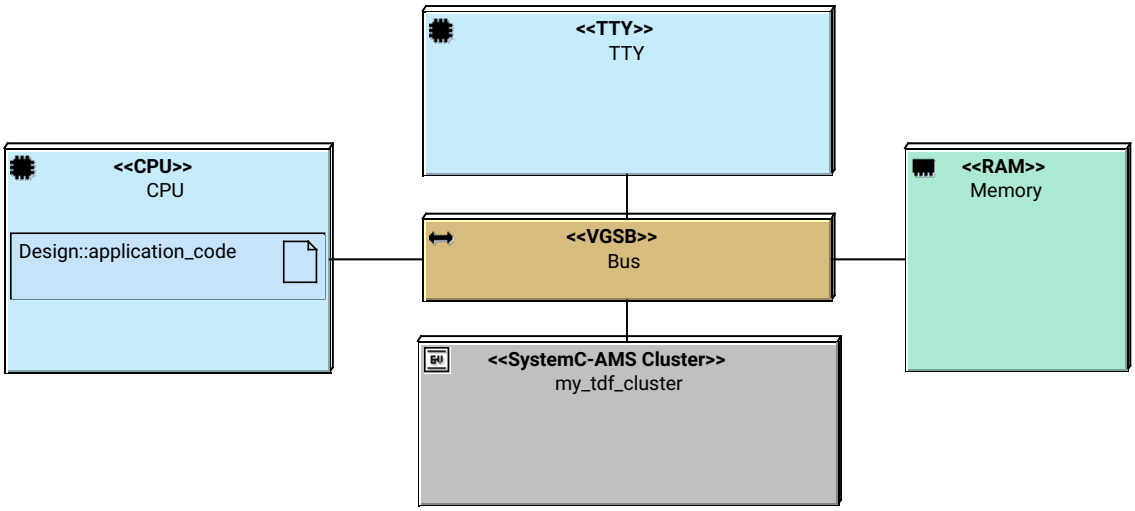

Fig. 6: Deployment Diagram (from [18])

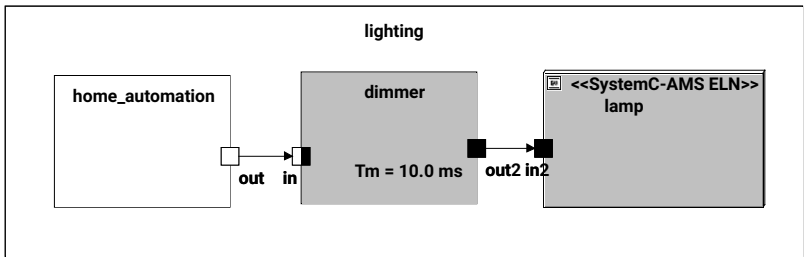

Fig. 7: TTool representation combining three different SystemC AMS MoC (from [18])

4.2 SysML-like hierarchical modeling of SystemC AMS modules

Digital hardware is represented in a UML Deployment Diagram (Figure 6 as defined in [17]. A micro controller and the application are shown in the light blue box on the left (named CPU and *application_code*, respectively), all connected to a bus. The platform also features RAM memory and a TTY to serve debug purposes. Figure 6 shows the TDF controller as a grey box at the bottom. The DE block on the left of Figure 7 represents the interface to the micro controller.

By selecting such a TDF module, the user opens a TDF panel like the one shown in Figure 7. The left part of this Figure, "DE_block", represents the interface to the digital hardware, for example a micro controller or general purpose processor running application code. This block is connected to a TDF block (in the middle) which samples at a given frequency (indicated by $T_m = 1.0\mu$s in tdf_block). Causality issues between the TDF and the DE MoC are explained in [8]. The right hand side of the Figure shows the encapsulation of a ELN cluster into a TDF cluster. Input and output are handled via TDF ports with sampling frequency of 1 μs, to which the scheduling algorithm of [8] is applied and whose sampling frequency is enforced on the encapsulated ELN modules. The precise way of handling inputs and outputs by ELN components is hidden at this abstraction level, a *processing function* handles the transfer.

By selecting the ELN cluster block, the corresponding ELN panel opens (Figure 4). In this toy example, the left hand module has a TDF input port connected to TDF_VSource, a TDF-to-ELN converter module used for the TDF module to control the voltage source. This module is in turn connected via its positive (p) and negative (n) *terminals* to an ELN resistor.

While multiples ports can be used for TDF modules,the SystemC AMS standard ELN specification does not define them. Thus, each TDF port array has to be split into individual ports connected to one TDF/ELN converter module each. As an improvement, we introduce ELN multiple ports at the entry and exit of each ELN cluster. Figure 8 shows the toolbar of the new ELN panel featuring all graphical operators which are supported in this diagram. These ELN elements are also listed in Table 1.

Currently, we support 20 elements out of the 29 defined in the SystemC AMS standard, not counting ports, connectors and terminals. The nullor and gyrator elements, and the TDF and DE controlled versions of variable lumped elements were less frequently required in our designs, but could be easily added to our framework.

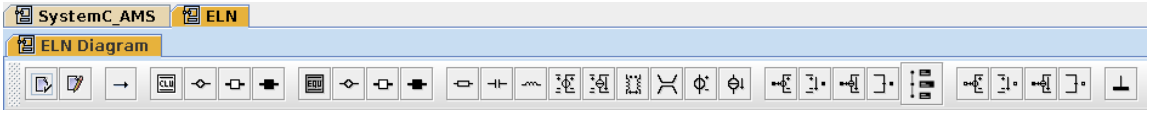

Fig. 8: ELN Panel Toolbar (from [18])

4.3 Integration into the overall method

Figure 9 displays the overall design method suggested for systems with digital and analog parts. The top of the figure focuses on hardware/software partitioning: a functional representation is mapped into a hardware platform. The functionality is partitioned into software tasks represented by the state machine diagram on the left and the block diagram on the right and hardware. The deployment diagram on the upper right represents all of the selected hardware. A virtual prototype is then generated using an assembly of components from the free SystemC library [38], and from analog hardware components given in SystemC AMS.

4.4 Virtual prototype generation

Note that an ELN cluster can never be simulated alone, it requires a TDF block which imposes its time step. For a virtual prototype containing three different Models of Computation, it is particularly important that interactions between the MoC are set up as early as possible in the design process. [8] has already shown how to efficiently generate TDF and DE parts of the prototype as well as their interaction, including validation of causality on the TDF/DE converter ports. In our tool, this can be done as follows:

1. Select one of the TDF clusters, including the ones which contain ELN clusters.
2. Activate the "Validation" button. This trigger the check for coherency of the time steps within the TDF cluster as well as for respect of temporal causality between TDF and DE models as described in [8], and proposes a valid schedule.
3. Activate "Code generation" button. For all ELN clusters, the correct SystemC AMS TDF module templates with the appropriate input and output ports are generated.

Element	Symbol
Cluster and module terminal	
TDF cluster and module port	
DE cluster and module port	
Tesistor	
Capacitor	
Inductor	
Voltage controlled voltage source	
Voltage controlled current source	
Ideal transformer	
Transmission line	
Independent voltage source	
Independent current source	
Voltage source driven by TDF input signal	
Voltage converted to TDF output signal	
Current source driven by TDF input signal	
Current source converted to a TDF output signal	
Switch driven by a TDF input signal	
Switch driven by a DE input signal	
Voltage source driven by DE input signal	
Voltage converted to DE output signal	
Current source driven by DE input signal	
Current source converted to DE output signal	
Reference node (ground)	

Table 1: ELN modules currently available in the panel (from [18])

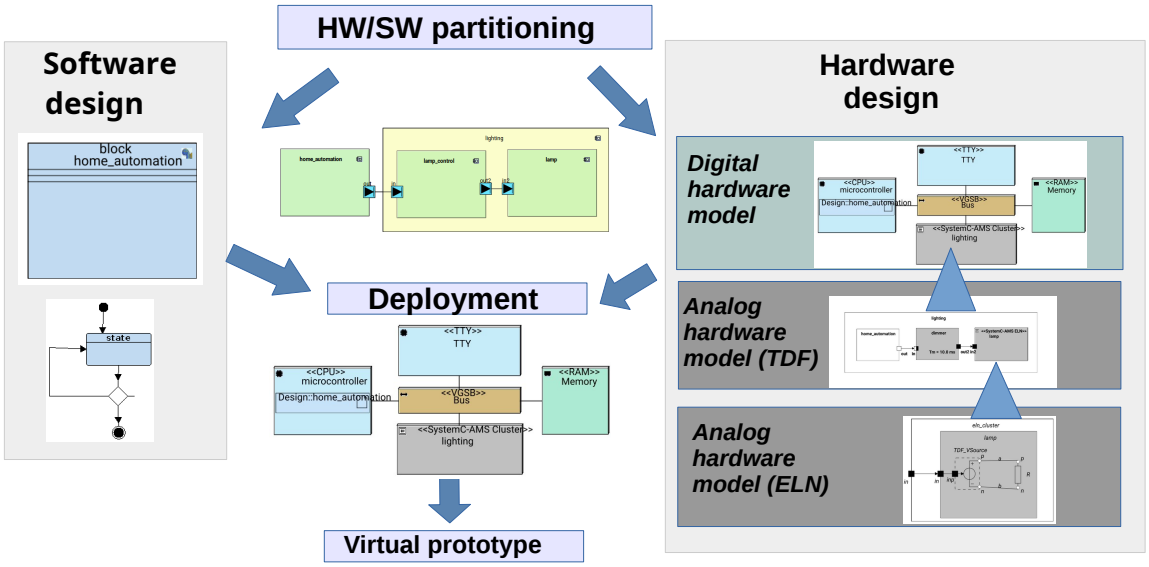

Fig. 9: Method with software design (from [18])

4. Select one of the ELN clusters within the TDF cluster.
5. Activate the "Code generation" button. This generates, for the selected ELN cluster representation, SystemC AMS code for all internal ELN modules. It updates the module template for the surrounding TDF block with the instantiation of the internal ELN blocks and the signals connected to the internal ports.

In our tool, the design choice was made that ELN clusters are always modeled inside TDF blocks. The algorithm given in [8] propagates the time steps and checks for schedulability and causality issues on the abstraction level where interaction between TDF/DE blocks is analyzed. An ELN cluster thus can never be simulated alone, it requires a TDF block that forces its time step. Thus, an ELN module has to be encapsulated within a TDF module.

Finally, the virtual prototyping code must be generated for:

- The top cell, containing the simulation entry point, TDF and DE block instantiation, code for starting and stopping the simulation and optional code for tracing.
- The ELN cluster encapsulation module. This is a TDF module instantiating the ELN modules, their connections among each other and to the TDF modules.
- The ELN module itself.

Algorithm 1.1 shows the transformation for code generation and scheduling, leaving out the DE part and using the scheduling algorithm CALCULATESCHEDULE of [8] for TDF clusters.

Figure 10 shows the generated code for the top cell which includes the code from files for the three parts: DE block, TDF block and ELN equation system. Figure 11 shows the generated SystemC AMS code for the ELN cluster encapsulation and ELN module of the introductory example, respectively. The TDF ports have to be connected to the sub modules of the ELN cluster. As a consequence, in the ELN cluster code, we find port-to-port binding: port "in" of the module is thus bound to the cluster port named "in". The ELN module contains a voltage source controlled by the TDF module via the "inp" port connected to the cluster port "in", a ground element, and two nodes a and b connecting them.

Listing 1.1 Code generation and scheduling algorithm from [18]

```
 1: procedure GENERATESYSTEMCAMSCODE
                                                  ▷ T time step, 𝒷 block,
                                                  ▷ 𝒸 cluster, 𝓂 module
 2:     for each TDF cluster 𝒸_TDF do
 3:         generate cluster code
 4:         for all TDF blocks 𝒷_TDF in 𝒸_TDF do
 5:             CALCULATESCHEDULE (𝒸_TDF)
 6:             if 𝒷_TDF simple TDF block then
 7:                 generate TDF block code
 8:             else
                                          ▷ 𝒷_TDF contains ELN cluster 𝒸_ELN
 9:                 for all 𝓂_ELN ∈ 𝒸_ELN do
10:                     set T_𝓂ELN from 𝒷_TDF
11:                     determine T_pi, T_po for 𝒷_TDF
12:                 end for
13:                 calculate T_𝒸ELN
14:                 CALCULATESCHEDULE (𝒸_TDF)
15:                 if 𝒸_TDF schedulable then
16:                     generate encapsulation code
17:                     for all 𝓂_ELN ∈ 𝒸_ELN do
18:                         generate ELN code
19:                     end for
20:                 end if
21:
22:             end if
23:         end for
24:     end for
25: end procedure
```

4.5 Simulation

Our simulation environment assumes that:

- The DE simulator controls the entire simulation via the converter ports [8, 2].
- TDF modules impose their timestep on the ELN modules, as described in [1].
- Even if possible, according to [1], direct assignment of a timestep to an ELN module is not allowed.

Figure 12 shows the SystemC AMS simulation of all three parts for the introductory toy example. Three modules, one DE, TDF and ELN module each, are instantiated, two so-called *views* correspond to the TDF and the ELN MoC. After this, the dataflow (TDF) cluster is analyzed by SystemC AMS, then the ELN cluster.

Figure 13 shows the waveforms of the signals exchanged between the ELN and TDF module in the example, obtained during simulation. This representation uses the Gtk Analog Wave Viewer [35], and can be compared to the prototype hardware implementation on FPGA (Field-programmable Gate Array) and analog circuit.

```
#include <systemc-ams>
#include "generated_H/lighting.h"
#include "generated_H/lamp.h"
#include "generated_H/home_automation.h"

// Simulation entry point
int sc_main(int argc, char *argv[]) {
        using namespace sc_core;
        using namespace sca_util;

        // Declare signals to interconnect.
        sc_core::sc_signal<int> sig_1;
        sca_tdf::sca_signal<double > sig_2;

        // Instantiate cluster's modules.
        dimmer dimmer_0("dimmer_0");
       lamp
          lamp_1("lamp_1");
          home_automation home_automation_2("home_automation_2");
                dimmer_0.out(sig_2);
                dimmer_0.in(sig_1);

                lamp.in(sig_2);
                home_automation_2.out(sig_1);

        // Start and stop simulation
        sc_start(10000000.0, SC_MS);
       sc_stop();
       return 0;
}
```

Fig. 10: Generated code for the TDF cluster top cell

5 Adding Configurability

The tool extension described above allows the integration of ELN and the generation of virtual prototypes combining all three MoC. But there is also commonly the need to evaluate designs with different parameters. The larger case study presented in Section 6 is a typical system that needs to be investigated for different configurations.

TTool already allows to configure many parameters for digital hardware components, such as cache associativity and size, bus speed, etc. Yet, dealing with a certain degree of replication, other than inserting a SysML block for each new hardware module (processor, memory bank, etc.) was not yet possible. The problem is exacerbated with the high complexity of ELN designs and their connection to the rest of the system. TTool diagrams thus neither take into account the number of bits/size of ports, identically replicated sub-circuits etc.. Unfortunately, this is an almost required feature for SystemC AMS low level ELN models.

```
#include <systemc-ams>
#include "lamp.h"
class lamp : public sc_core::sc_module {

public:
lamp lamp_0;
sca_tdf::sca_in<double> in;

SC_CTOR() :
lamp_0("lamp_0"),
in("in"){
lamp_0.in(in);
};
SC_MODULE(lamp){
        sca_tdf::sca_in<double> in;
        sca_eln::sca_r R;
        sca_eln::sca_tdf_vsource TDF_VSource;

        SC_CTOR(lamp)
        : in("in"), R("R", 1.0),
        TDF_VSource("TDF_VSource", 1.0),
        b("b") , a("a") {
                R.p(a);
                R.n(b);
                TDF_VSource.p(a);
                TDF_VSource.n(b);
                TDF_VSource.inp(in);
          }
private:
        sca_eln::sca_node b;
        sca_eln::sca_node a;
};
```

Fig. 11: Generated encapsulation code and ELN code for the module *lamp*

5.1 Introducing ELN Multi Modules

The number of ELN elements used in a design counts for the total surface of the circuit [1]. To express a configurable number of identical modules and thus better handle scalability, we define a new module which allows to configure the number of identical ELN circuits it contains.

As already mentioned, ELN modeling does not allow custom modules. We solve the problem by providing a more compact representation of replication in TTool-AMS, without however violating SystemC AMS semantics: transparently to the user, so-called *multi modules*, representing an aggregation of identical sub-circuits, are translated into single modules in the textual SystemC AMS code, each one individually connected through its "terminal".

[1] We do not consider layout issues in this paper. Yet, we are aware of its importance, especially when dealing with low-level steps of the design process.

```
Info: SystemC-AMS:
        3 SystemC-AMS modules instantiated
        2 SystemC-AMS views created
        2 SystemC-AMS synchronization objects/solvers instantiated

Info: SystemC-AMS:
        1 dataflow clusters instantiated
          cluster 0:
                2 dataflow modules/solver, contains e.g. module: dimmer_0
                2 elements in schedule list,
                10 ms cluster period,
                ratio to lowest:  1              e.g. module: dimmer_0
                ratio to highest: 1 sample time  e.g. module: dimmer_0
                0 connections to SystemC de, 1 connections from SystemC de

Info: SystemC-AMS:
        ELN solver instance: sca_linear_solver_0 (cluster 0)
                has 3 equations for 2 modules (e.g. lamp_0.R),
                1 inputs and 0 outputs to other (TDF) SystemC-AMS domains,
                0 inputs and 0 outputs to SystemC de.
                10 ms initial time step

Info: SystemC-AMS:
        ELN solver instance: sca_linear_solver_0 (cluster 0)
                has calculated 10000 time steps the equation system was 1 times re-initialized
                the following max. 10 modules requested the most re-initializations: lamp_0.R  1
```

Fig. 12: Simulation

Figure 14 shows the introductory example, featuring two light bulbs. Here, the two identical Vsource-Resistor circuits are replaced by a new multi module with two subsystems. The module thus has a port of arity 2. The multi module can then be named and used as a black box in future designs.

A multi module is constructed by designing the circuit and attaching it to the multi module, which in turn is attached to the proper module. Then, the arity is chosen, indicating how many instances of the circuit will be employed.

5.2 Introducing Multi Ports

In the initial, purely digital tool implementation, software modules were communicating via signals. Tasks running on processors were communicating through ports via 32-bit wide FIFOs channels, and all transfers were bursts of a multiple of 32 bit [17]. Those sizes were due to hardware restrictions imposed by SoCLib [38] and every communication was between performed via these channels.

Multiple TDF ports On the analog level, things much are less simple: The number of bits that are transferred at a time is important. In SystemC AMS TDF we thus have the notion of *arrays of ports* of identical size and type. For example, a write operation to the second of two identical ports of type boolean, expressed as an array of ports of size 2, named `out[2]`, is expressed as follows: `out[1].write(true)`.

Fig. 13: Co-simulation waveform result

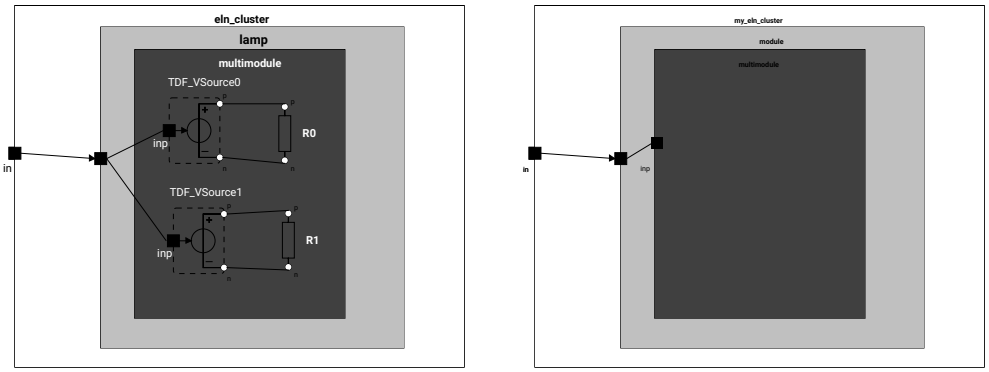

Fig. 14: Configuration of a multi module with TDF ports

While arrays of ports are defined both for DE and TDF modules, this is not the case for ELN modules. The reason is that ELN modules represent electrical circuits, connected among each other by "terminals". Lumped elements have two terminals top connect p and n, and so forth. A number of ELN modules are in addition controlled by TDF or DE ports, as shown in the lower part of Table 1. In order to connect arrays of TDF signals, each signal has to be connected to one port, hampering scalability.

Multiple ELN terminals The issue is more complex with ELN terminals. The SystemC AMS standard forbids multiple terminals. We thus define a new connection feature, displayed with a white circle. This connection feature is to be translated into multiple terminals in SystemC AMS.

In the example shown in Figure 15, three resistors are connected via their p terminals to the nodes from the p terminal of the source to the p terminal of the sink. The same connection is done for their n terminals. Let us now replace it by a multi module. The three terminals become one terminal of arity 3 (but which is translated into three separate terminals, each connected to their respective node).

6 Case Study: Scalable SAR ADC

Cyber-physical systems have two domains (analog, digital) interconnected with digital-to-analog (DAC) and analog-to digital (ADC) converters. These converters are expected to be of small size and designed with high energy efficiency in mind.

Successive approximation register (SAR) ADCs provide good power efficiency for medium-resolution applications [37]. Wireless local area network (WLAN) for example require resolutions above 10 bit and

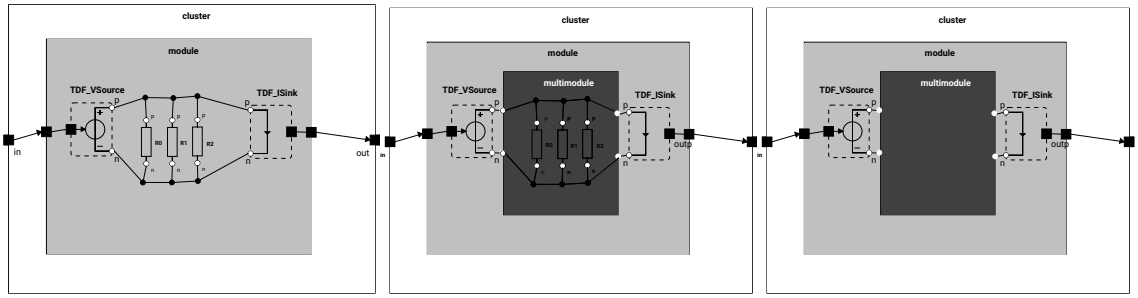

Fig. 15: Transformation example for a multi module with TDF ports and ELN terminals

sampling rates of about 40 MS/s, digital TV receivers up to 80 MS/s sampling rate, while Bluetooth applications require resolution and sampling rates of around 12-bit and 11 MS/s, respectively. ADC should support multiple applications, so their design is required to be easily reconfigurable for multi-standard systems.

Figure 17 shows an overview the general design of such a SAR-DAC circuit. [21] gives an overview of several possible designs. The basic idea of SAR ADCs is to approximate the actual voltage by several iterations, corresponding to the number of bits which are fed back to a DAC. The number of bits, the most essential parameter of the circuit, is usually selected between 3 to 12-bit precision.

We consider a SAR ADC designed in an ongoing project [30], mostly consisting of analog components, with a digital component controlling the device. The implementation was performed with the *Oceane* toolchain [33]. One interesting challenge is to couple the implementation process with a system-level modeling approach. In the beginning, a very detailed model of the ADC circuit was not yet available, the number of bits of precision required was not yet known, and in consequence the system-level model had to be easily modified. In the scope of this project, we could thus evaluate the extensions to our SysML-based modeling tool [3].

Figure 16 shows the main algorithm of the ADC [30]. The idea is to iteratively quantify an incoming voltage V_x by setting an initial voltage value V_in; control all bits are initially set to 0. The algorithm starts by setting the most significant bit (MSB) B_n to 1. At a given time t, V_x is successively compared to a voltage V_dac generated by the DAC, setting the control bit to 1 if the voltage is higher, 0 if the voltage is lower. This algorithm was implemented in VHDL [24] targeting an FPGA (Field-programmable Gate Array). The control bits are used to steer a digital analog converter (DAC) producing a more precise voltage for the next iteration.

Figure 18 shows the detailed hardware implementation proposed by [30]: a non-differential ADC with implicit sampling using capacitor top plates. On the upper center, we find a comparator (CMP) which compares zero/ground voltage (VSS) to the voltage generated in each cycle by the DAC (VDAC). Shown on the lower left hand side, the DAC produces this voltage from $i+1$ capacitors which are either activated (switch closed) when the control bit S_i is 1, deactivated when it is 0. B_i has the same value as S_i but is destined for digital output as a bit vector. Thus, during n iterations, the incoming voltage is approximated with n bit precision. An additional capacitor on the right sets the starting capacity, the others then yield 2^0,2^1, etc ... 2^{N-1} times that capacity. The implementation of this design is a typical challenging test case for our design parametrization proposal, and our subsequent tool extension.

Figure 19 shows the overall SysML based representation of the SystemC AMS TDF design, created using our tool. For simplicity, the entire general purpose digital platform is represented on the lower right in form of a DE block, whose only task here is to provide the start of conversion (SoC) signal on a single output

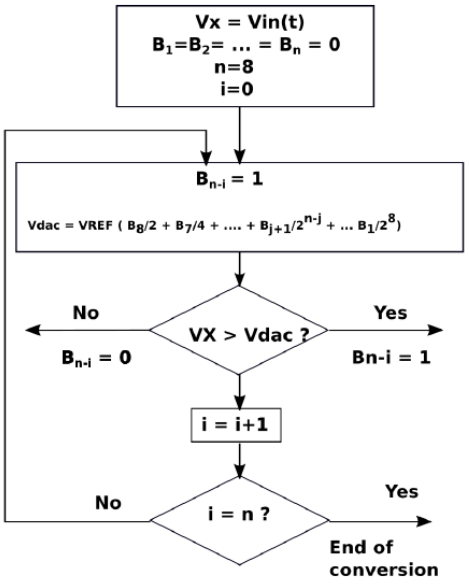

Fig. 16: Conversion algorithm from [18],[30]

port. In current experimentation, code on the digital platform is essentially limited to start/stop signals, and to I/O functionality, which are abstracted in a DE module, shown in white.

In fact, the control logic has been moved into this TDF block and considered as mixed-signal despite its digital nature (originally described in the VHDL language) as currently the tool allows only TDF blocks to contain ELN. The conversion algorithm (Figure 16) thus has been implemented in the *processing* function of the *control_logic* block as shown in Figure 20).

The sampling algorithm is implemented in SystemC in the Control Logic component. It was translated from VHDL and precisely reflects the functionality shown in the algorithm of figure 16.

The comparator-and-DAC block has two TDF entry ports called *start_conversion* and *in_bits*. The *start_conversion* TDF signal is received from the *control_logic* block, the configurable *in_bits* signal contains the *n* bits controlling the switches in the DAC. As the number of control bits is configurable, we make use of the multiport mechanism: the arity of a TDF port can be configured. The block also features an output port *VDD_out*, providing the voltage calculated after each iteration of the algorithm (output of CMP block in Figure 18), a floating point value.

Double clicking on the *comparator_and_DAC* block opens the most detailed view (Figure 21). The comparator-and-DAC block contains the two analog ADC blocks *Comparator (CMP)* and *DAC* (representing the entire left part of Figure 18). These blocks are described in SystemC AMS ELN and form the cluster *comparator_and_DAC*.

The comparator on the top receives *V_dac*, via an ELN terminal (bottom of the block,) from the DAC that produced it with the SAR method, as explained beforehand. It is compared to the incoming voltage to be measured, *V_x*, modeled by an independent voltage source on the left of the comparator. At each iteration, the result of the successive approximations is transmitted to the outer world by TDF port *VDD_out* (a *double* value).

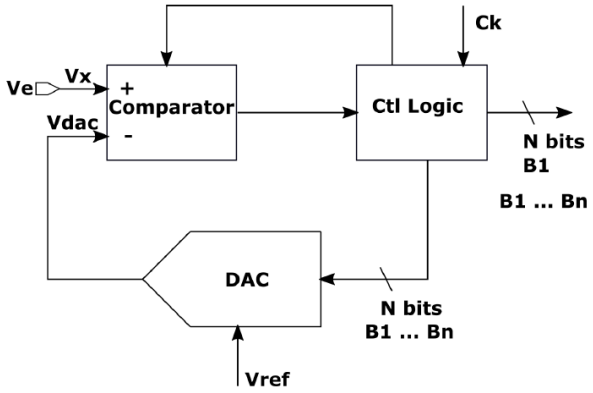

Fig. 17: SAR Analog-Digital converter design overview from [18], [30]

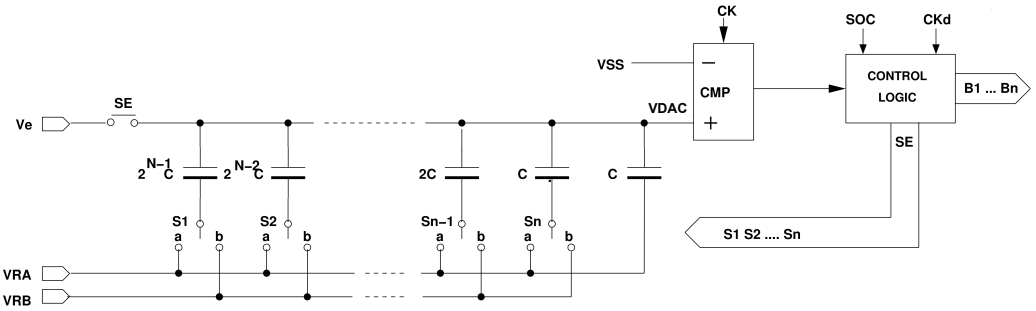

Fig. 18: Analog-Digital converter: electronic design (simplified) from [30]

The lower part of the figure represents the DAC. The voltage *Ve* (modeling V_x, the voltage to be determined) is generated by the independent voltage source on the upper left of the DAC model.

As ELN cannot handle arrays of ports, the ports in the cluster port *in_bits* (in our example of arity 3) were initially split into individual TDF module ports.

Two rows of three TDF controlled Switches (TDF_Switch_a1 to TDF_Switch_a3 and TDF_Switch_b1 to TDF_Switch_b3) take up the central part of the design and are connected, by their control ports, to one of three TDF in_bits signals, each representing one of the control bits for one switch of each row. There are four capacitors, three to be controlled by two switches each; C0 is not controlled by a switch and gives the initial value which is then doubled, quadrupled etc. as described in the algorithm above. A seventh switch *TDF_Switch_*0 is connected to the start_conversion port. Voltage sources Va and Vb on the right are set to 2V and 0V, respectively.

Figure 21 thus shows the full complexity of modeling the design of Figure 18. The voltage to be determined is represented by the independent voltage source V_in on the left of the comparator block.

This case study somehow triggered the research for a representation of multi modules: thanks to our contribution, the ELN model can now be rewritten in a more compact form (right hand side of Figure 21). Design space exploration can now be easily done by varying only the parameter of the number of bits, for example, increase it progressively from 3 to 12. While our contribution was to setup the ability to parame-

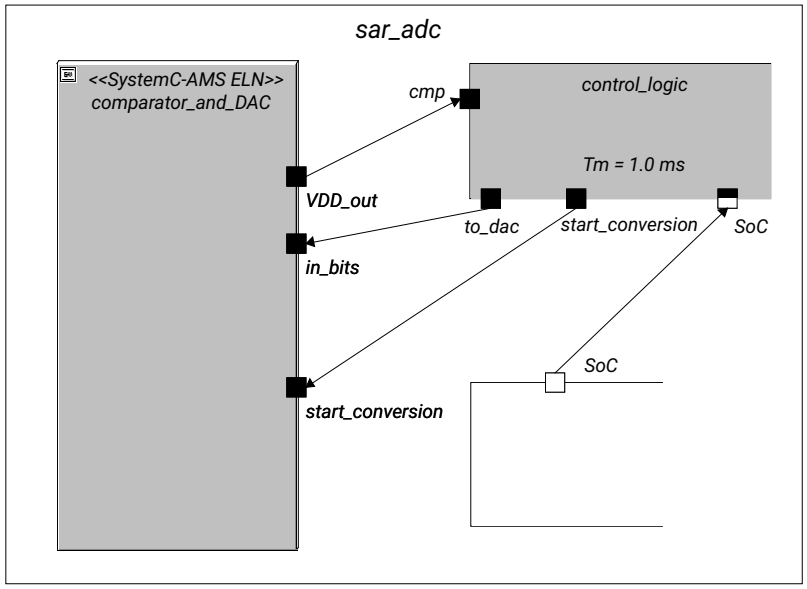

Fig. 19: TDF model view of the SAR-ADC in the tool (from [18])

terize ELN models, which is now easily possible, the exploration itself is still to be done in the scope of this case study.

7 Conclusion and future work

SystemC AMS based hierarchical design of cyber-physical systems is now supported by our tool, with a concrete and effective support for both digital and analog parts. Currently, the consistency between ELN and TDF is checked by SystemC AMS at simulation time, but the hypotheses from section 4.5 could be used to validate schedulability and causality between TDF and ELN *before* simulation, at prototyping time, similar to what was already shown in [8] for these properties between TDF and DE.

Yet, even after defining multi ports and multi modules, ELN diagrams still tend to quickly become complex to read; we are thus working on further visual improvements, such as use of colors and a better representation for line crossing.

The idea of multi modules should evidently be extended to multiple SystemC AMS blocks in general, like TDF and DE blocks. By doing so, identical modules running in parallel could be modeled, and a more compact representation also of the digital part of the system would be possible, for example for replicated software tasks and multi-core hardware.

Direct integration from ELN modules into DE modules is not yet supported; we adhere to the approaches of [2] and [5], which always require a TDF module beween ELN and DE.

Our tool also provides comfortable possibilities to model, verify and simulate embedded software on a virtual prototype: this aspect has been left out in the present paper in order to focus on the hardware design part. In a near future, we also would like to interface our tool with other free CAD software such as YOSYS, CORIOLIS, ... to obtain actual mixed-signal designs featuring FPGA and analog parts.

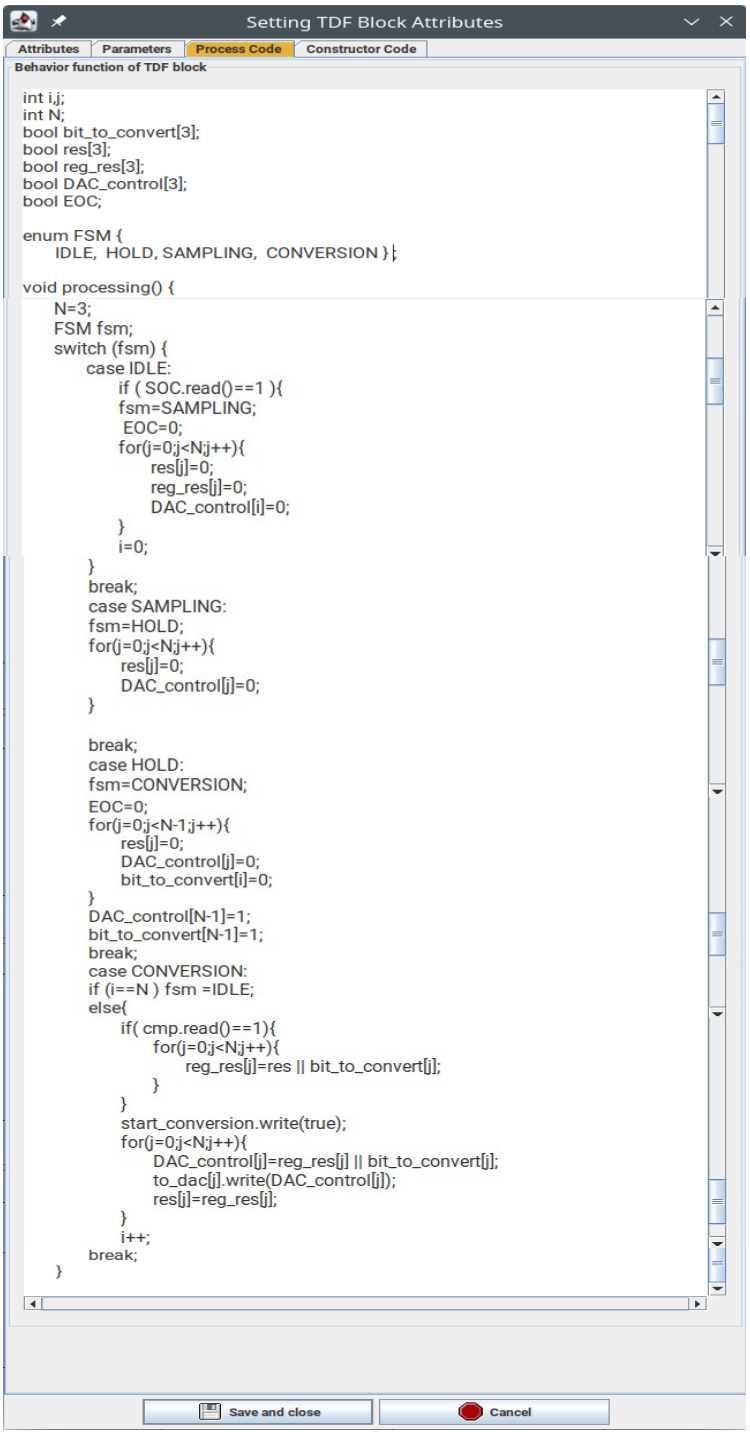

Fig. 20: Analog-Digital converter: complete processing function

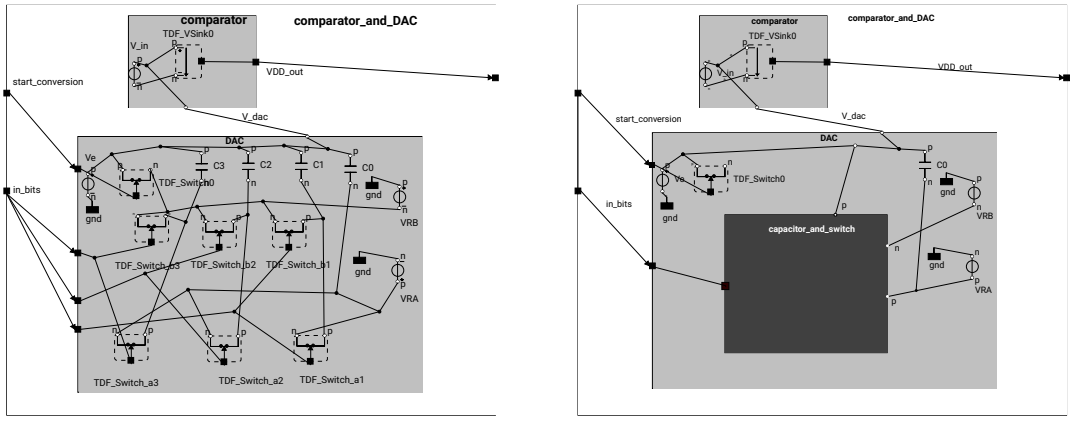

Fig. 21: Overview of the ELN Model of the comparator and DAC module: 3 bit conversion from [18] (left) and rewritten configurable version (right)

References

1. Accellera Systems Initiative: SystemC AMS extensions Users Guide, Version 1.0 (March 2010)
2. Andrade, L., Maehne, T., Vachoux, A., Ben Aoun, C., Pêcheux, F., Louërat, M.M.: Pre-Simulation Formal Analysis of Synchronization Issues between Discrete Event and Timed Data Flow Models of Computation. In: Design, Automation and Test in Europe, DATE Conference (Mar 2015)
3. Apvrille, L.: TTool, an open-source toolkit for the modeling and verification of embedded systems, http://ttool.telecom-paristech.fr/ (2011)
4. Apvrille, L., Muhammad, W., Ameur-Boulifa, R., Coudert, S., Pacalet, R.: A uml-based environment for system design space exploration. In: 2006 13th IEEE International Conference on Electronics, Circuits and Systems. pp. 1272–1275. IEEE (2006)
5. Beyond Dreams Consortium: Beyond Dreams (Design Refinement of Embedded Analogue and Mixed-Signal Systems) (2008-2011), projects.eas.iis.fraunhofer.de/beyonddreams
6. Blochwitz, T., et al.: The functional mockup interface for tool independent exchange of simulation models. In: 8th Int. Modelica Conference, Dresden, Germany. pp. 105–114 (2011)
7. Concepcion, A.I., Zeigler, B.P.: DEVS formalism: A framework for hierarchical model development. IEEE Transactions on Software Engineering **14**(2), 228–241 (1988)
8. Cortés Porto, R., Genius, D., Apvrille, L.: Handling causality and schedulability when designing and prototyping cyber-physical systems. Software and Systems Modeling pp. 1–17 (2021)
9. Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Yang, G., Zeng, H., Zhu, Q.: A next-generation design framework for platform-based design. In: DVCon. vol. 152 (2007)
10. Demathieu, S., Thomas, F., André, C., Gérard, S., Terrier, F.: First experiments using the uml profile for marte. pp. 50–57. IEEE (2008)
11. Dubrulle, P., Gaston, C., Kosmatov, N., Lapitre, A., Louise, S.: A data flow model with frequency arithmetic. In: International Conference on Fundamental Approaches to Software Engineering. pp. 369–385. Springer, Cham (2019)
12. Einwich, K.: Coside,https://www.coseda-tech.com, `https://www.coseda-tech.com`
13. Einwich, K.: SystemC AMS PoC2.1 Library, COSEDA, Dresden (2016)
14. Fitzgerald, J.S., Larsen, P.G., Pierce, K.G., Verhoef, M.H.G.: A formal approach to collaborative modelling and co-simulation for embedded systems. Mathematical Structures in Computer Science **23**(4), 726–750 (2013)
15. Fritzson, P., Engelson, V.: Modelica—a unified object-oriented language for system modeling and simulation. In: European Conference on Object-Oriented Programming. pp. 67–90. Springer (1998)

16. Gamatié, A., Beux, S.L., Piel, É., Atitallah, R.B., Etien, A., Marquet, P., Dekeyser, J.L.: A model-driven design framework for massively parallel embedded systems. ACM Trans. Embedded Comput. Syst **10**(4), 39 (2011)
17. Genius, D., Apvrille, L.: Virtual yet precise prototyping: An automotive case study. In: ERTSS'2016. Toulouse (Jan 2016)
18. Genius, D., Apvrille, L.: Hierarchical design of cyber-physical systems. In: Modelsward (2023)
19. Genius, D., Bournias, I., Apvrille, L., Chotin, R.: High-level partitioning and design space exploration for cyber physical systems. In: MODELSWARD (2020)
20. Genius, D., Cortés Porto, R., Apvrille, L., Pêcheux, F.: A tool for high-level modeling of analog/mixed signal embedded systems. In: MODELSWARD (2019)
21. Gylling, V., Olsson, R.: Implementation of a 200 msps 12-bit sar adc (2015)
22. H-Inception Consortium: Heterogeneous Inception Project (2012-2015), https://www-soc.lip6.fr/trac/hinception
23. Herrera, F., Villar, E.: A framework for heterogeneous specification and design of electronic embedded systems in systemc. ACM Transactions on Design Automation of Electronic Systems (TODAES) **12**(3), 22 (2007)
24. IEEE: IEEE Standard VHDL Language Reference Manual (1987)
25. IEEE: SystemC. IEEE Standard 1666-2011 (2011)
26. Jantsch, A.: Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation. Elsevier (2003)
27. Lee, E.A.: Disciplined heterogeneous modeling. In: Petriu, D., Rouquette, N., Haugen, O. (eds.) Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering, Languages, and Systems (MODELS). pp. 273–287. LNCS 6395, Springer-Verlag (Oct 2010)
28. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
29. Li, L.W., Genius, D., Apvrille, L.: Formal and virtual multi-level design space exploration. In: MODELSWARD, Springer CCIS vol 880. pp. 47–71 (2018)
30. Louërat, M.M., Porte, J.: scalable sar adc, technicat report, chips4makers.io (2022)
31. Niaki, S.H.A., Jakobsen, M.K., Sulonen, T., Sander, I.: Formal heterogeneous system modeling with systemc. In: Specification and Design Languages (FDL), 2012 Forum on. pp. 160–167. IEEE (2012)
32. Patel, H.D., Shukla, S.K.: Towards a heterogeneous simulation kernel for system-level models: a systemc kernel for SDF models. TCAD **24**(8), 1261–1271 (2005)
33. Porte, J.: Oceane: Software tool for analog design and education". https://www-soc.lip6.fr/en/team-cian/softwares/oceane/ (2008)
34. Quarles, T., Pederson, D., Newton, R., Sangiovanni-Vincentelli, A., Wayne, C.: Spice home page. Go online to http://bwrc. eecs. berkeley. edu/Classes/IcBook/SPICE (2003)
35. Quillevere, H.: Gtk Analog Wave viewer (2019), "`http://www.rvq.fr/linux/gaw.php`"
36. Selic, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. Elsevier (2013)
37. Shen, Y., Zhu, Z., Liu, S., Yang, Y.: A reconfigurable 10-to-12-b 80-to-20-ms/s bandwidth scalable sar adc. IEEE Transactions on Circuits and Systems I **65**(1), 51–60 (2017)
38. SocLib consortium: The SoCLib project: An Integrated System-on-Chip Modelling and Simulation Platform, www.soclib.fr (2003), `www.soclib.fr`
39. Taha, S., Radermacher, A., Gérard, S.: An entirely model-based framework for hardware design and simulation. In: DIPES/BICC. IFIP Advances in Information and Communication Technology, vol. 329, pp. 31–42. Springer (2010)
40. Vachoux, A., Grimm, C., Einwich, K.: Analog and mixed signal modelling with SystemC-AMS. In: ISCAS (3). pp. 914–917. IEEE (2003), `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8570`
41. Zhao, C., Kazmierski, T.J.: An extension to SystemC-A to support mixed-technology systems with distributed components. In: DATE. pp. 1–6. IEEE (2011)
42. Zhu, J., Sander, I., Jantsch, A.: Hetmoc: Heterogeneous modelling in systemc. In: Specification & Design Languages (FDL 2010), 2010 Forum on. pp. 1–6. IET (2010)