A Framework for Multi-level Modeling of Analog/Mixed Signal Embedded Systems

Daniela Genius[1(✉)], Rodrigo Cortés Porto[1,3], Ludovic Apvrille[2], and François Pêcheux[1]

[1] Sorbonne Université, LIP6, CNRS UMR 7606, Paris, France
daniela.genius@lip6.fr

[2] LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia Antipolis, France

[3] Technische Universität Kaiserslautern, Kaiserslautern, Germany

Abstract. Embedded systems are commonly built upon heterogeneous digital and analog integrated circuits, including sensors and actuators. Model-driven approaches for designing software and hardware are generally limited to the digital parts of systems. In the present paper, we adopt a global view on the extensions made to an integrated modeling and simulation tool, TTool. In this tool, the verification and virtual prototyping of embedded systems is described at different abstraction levels and extended in order to handle analog/mixed-signal systems. An extensive case study spans these levels and illustrates the usefulness of our approach.

1 Introduction

Many model-driven techniques have been proposed for designing both digital software and hardware. High level models are employed to specify the functionality of the system, and subsequent model transformations are applied until a virtual prototype containing software and hardware can be generated. However, embedded systems—e.g. robotics, automotive and medical systems—are frequently built upon heterogeneous hardware components such as processors, FPGAs, DSPs, hardware accelerators, digital and analog analog/mixed signal (AMS) and radio frequency (RF) circuits. In early design phases, a high-level representation that includes both digital and analog descriptions is necessary in order to quickly explore the design space, taking into account both digital and AMS/RF components. Obviously, at such a high level of abstraction, speed of design space exploration prevents us from using precise models.

The paper gives an overview of our recent contribution [27] and completes several aspects that have not yet been treated beforehand. Our model-driven approach offers operators and views in order to capture digital and analog domains at several abstraction levels. This approach is supported by the free software TTool [6]. TTool can capture digital/analog aspects and generate a virtual prototype combining SystemC and SystemC-AMS in order to evaluate

S. Hammoudi et al. (Eds.): MODELSWARD 2019, CCIS 1161, pp. 201–224, 2020.
https://doi.org/10.1007/978-3-030-37873-8_9

the system under design. The paper focuses both on modeling capabilities and simulation aspects e.g. ways to combine AMS simulation with event-based (SystemC) simulation. An important aspect regarding simulation which is addressed in the paper is the problem of synchronization between time domains. The overall approach is explained with toy examples before being demonstrated with an automotive braking application.

In the next section, we give an overview of existing approaches targeting the modeling and/or co-simulation of cyber-physical systems. Section 3 presents the basic concepts behind the simulation of analog components. Section 4 explains how digital and analog components can be modeled and evaluated altogether. Section 5 illustrates the usefulness of the approach with a realistic system. Finally, Sect. 6 concludes the paper and gives a perspective on future work.

2 Related Work

Several well established tools in analog/mixed signal design, like *Ptolemy II* [35] [42], are based upon a data-flow model. They target heterogeneous system design by defining several sub domains [21] using hierarchical models. Instantiation of elements controlling the time synchronization between domains is left to the responsibility of designers. Recently, a co-simulation framework for timing verification of cyber-physical systems [29] from Ptolemy models, named *Metronomy*, has been developed.

Metropolis [7] is also based on high level models and facilitates the separation of concerns between computation and communication aspects. Heterogeneous systems are taken into consideration, yet heterogeneity can only be represented using processes, mediums, quantities and constraints. Hierarchical models are not allowed. Metro II [18] introduces hierarchy and allows so-called *Adaptors* for data synchronization, which serve as a bridge between the semantics of components belonging to different Models of Computation (MoCs). The model designer still has to implement time synchronization by means of constraints, assertions, annotators and schedulers. As a common simulation kernel handles the entire process execution (digital and analog), MoCs are not well separated.

From the Micro Electro Mechanical Systems (MEMS) community [10] stems an approach which can transform structural SysML diagrams into VHDL-AMS code. It is thus closely related to our work, but limited to its domain and generates VHDL specifiations, which are less flexible than most other approaches for expressing different Models of Computation, VHDL being essentially a hardware description language on register transfer level.

Discrete Event System Specification (DEVS [14]) is a modular and hierarchical formalism for modeling and analyzing general systems. DEVS supports discrete events and continuous systems. Continuous functions can be described by differential equations, or hybrid systems. A dozen of platform implementations based on DEVS exist, ranging from Petri Net over object oriented to Python based [12,41,50].

Modelica [22] is an object-oriented modeling language for component-oriented systems containing e.g. mechanical, electrical, electronic and hydraulic components. 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.

UML/SysML based modeling techniques such as MARTE and Gaspard2 [23,48] are extremely popular for capturing the behavior of embedded systems, but less widely used for heterogeneous system design [44]. Furthermore, with very few exceptions such as [39,46], they do not support refinement until cycle/bit accurate level virtual prototypes nor provide OS support for full-system simulation. Co-simulation between different Models of Computation is usually out of scope, too.

The B method [1] and more recently Event-B [2] model systems at different abstraction levels and makes it possible to mathematically prove consistency between refinement levels. Based on set theory and the B language, the B method is well established in large-scale public/private projects (urban transports etc.). To our knowledge, no extensions to cyber-physical systems have been proposed.

Several frameworks based on SystemC [32], a library of C++ classes, makes it possible to model (digital) hardware. For instanfe, HetSC [31], HetMoC [51] and ForSyDe [40] all have the disadvantage that instantiation of elements and controlling the synchronization have to be managed by the designer.

The following works stem from the analog/mixed signal hardware design domain, where SystemC-AMS extensions [3] is about to become a standard, describing an extension of SystemC with AMS and RF features [47]. The usual approach for linking the digital part of a heterogeneous system with SystemC-AMS is to rely on the *Discrete Event* (DE) parts of SystemC AMS extensions. For instance, *Timed data Flow* (TDF) adds support for signals where data values are sampled with a constant time step.

In the scope of the BeyondDreams project [9], a mixed analog-digital systems proof-of-concept simulator has been developed, based on the SystemC AMS extension standard. Another simulator is proposed in the H-Inception project [30]. All of these approaches rely on SystemC AMS code i.e. they do not provide a high-level interface for specifying the application.

3 Basic Concepts

First, let us briefly introduce two fundamental concepts and two associated tools. On the one hand, *Timed data Flow* as implemented in [19], on the other hand, multi-level modeling and virtual prototyping as implemented in TTool [6].

3.1 Timed Data Flow

SystemC AMS predefines several Models of Computation, e.g. the Timed Data Flow (TDF) Model of Computation, which is based on the timeless Synchronous

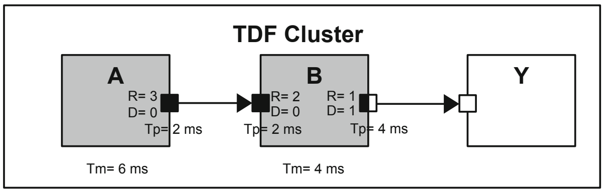


Fig. 1. TDF cluster [27].

Data Flow (SDF) semantics [36]. At each time step, a TDF module reads a fixed number of samples from each of its input ports, then executes the processing function, and finally writes a fixed number of samples to each of its output ports. TDF modules can interact with the discrete world (such as digital MPSoC platforms) using converter ports.

Figure 1 shows a graphical representation TDF cluster. Discrete DE modules are represented as white blocks, TDF modules as gray blocks, TDF ports as black squares, TDF converter ports as black and white squares, and finally TDF signals as arrows. So-called **converter ports**, shown as black-and white squares, serve as interface between the TDF and DE MoC. For the SysML-like notation supported by TTool, we will adhere to this representation.

TDF modules have the following attributes:

– Module Timestep (**Tm**) denotes the period during which a module is activated. One module is activated only if there are enough samples available at its input ports.
– Rate (**R**). A module reads or writes a fixed number of data samples each time it is activated. This number is annotated to the ports and it is known as the *Port Rate*.

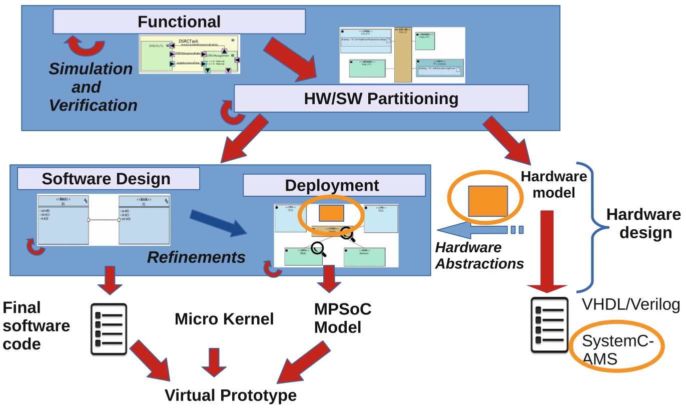


Fig. 2. Hardware/Software partitioning and Code generation for MPSoC platforms [27]. (Color figure online)

- Port Timestep (**Tp**) is the period during which each port of a module is activated. It also denotes the time interval between two samples that are being read or written.
- Delay (**D**). A Delay D can be assigned to a port to make it store a given number of samples each time it is activated, and read or write them in the next activation.

SystemC-AMS extensions, already mentioned in Sect. 2, define models of computations e.g. for TDF modules. We rely on a reference implementation [19] for generating the simulation code of the analog parts.

3.2 Modeling Tool

TTool [6] is a SysML based, free and open-source software initially designed for model-based engineering of (digital) embedded systems at different abstraction levels: functional, partitioning, software design, and deployment. To each of these levels, as shown in Fig. 2 taken from [27], is associated separate *panels*, which allow designers to model systems using a SysML-like notation. The method underlying these levels explains how to take hardware/software partitioning decisions at a high level of abstraction and to regularly validate them during software development [39].

Software and hardware tasks to be partitioned are first captured within the functional abstraction level. Software tasks used in deployments are captured in the software design abstraction level. In both partitioning and deployment, the computation part of tasks is deployed to processors or hardware accelerators, and the communication and storage parts are deployed to communication and storage elements e.g. buses and memories.

TTool allows verification and fast (and high-level) simulation of digital parts. It also supports cycle/bit accurate virtual prototyping on a Multi-Processor System-on-Chip (MPSoC) based on the *SoCLib* [45] public domain library written in SystemC. As SystemC-AMS is an extension to SystemC, relying on TTool for integrating analog/mixed signal components was natural. The next section discusses this integration.

4 Integration of Analog Components

In the following, we show how TDF concepts can be integrated into SysML-like models and in TTool, while keeping in mind our objective to generate correct-by-construction simulation code i.e. handling potential synchronization problems between domains before simulation starts. The philosophy of TTool also requires that all parts of the model are check against syntax (and against a few semantic aspects as well) **before** any code is generated.

Figure 2 uses orange circles to explain how the methodology described before have been adapted in order to support AMS components in TTool. Hardware parts, shown on the lower right, can be simulated with a cycle-accurate precision.

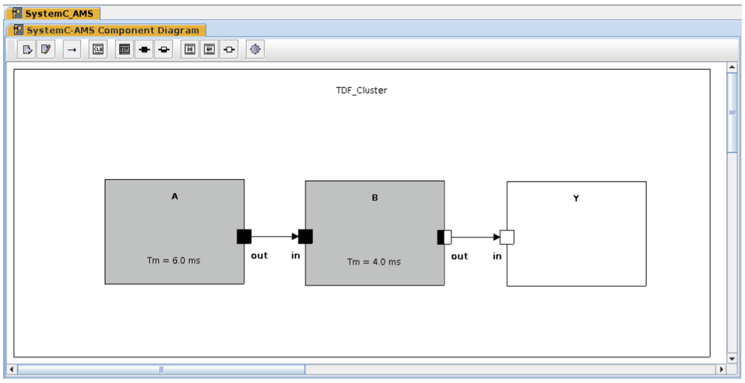
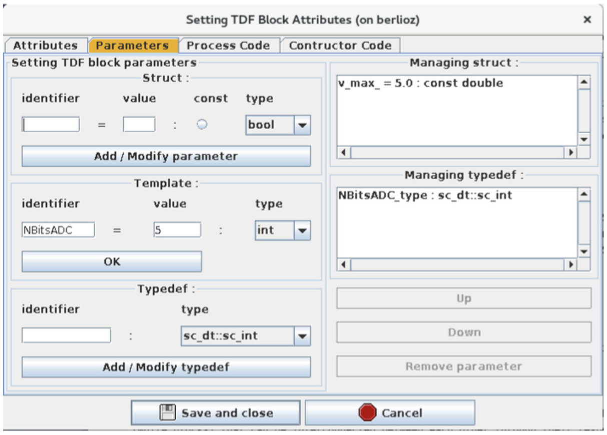


Fig. 3. SystemC-AMS diagram of Fig. 1 in TTool SystemC-AMS panel.



Setting TDF Block Attributes (on berlioz)

Attributes | Parameters | Process Code | Contructor Code

Setting TDF block parameters

Struct :

identifier		value		const	type
	=		:	○	bool

Add / Modify parameter

Template :

identifier		value		type
NBitsADC	=	5	:	int

OK

Typedef :

identifier		type
	:	sc_dt::sc_int

Add / Modify typedef

Managing struct :

v_max_ = 5.0 : const double

Managing typedef :

NBitsADC_type : sc_dt::sc_int

Up

Down

Remove parameter

Save and close

Cancel

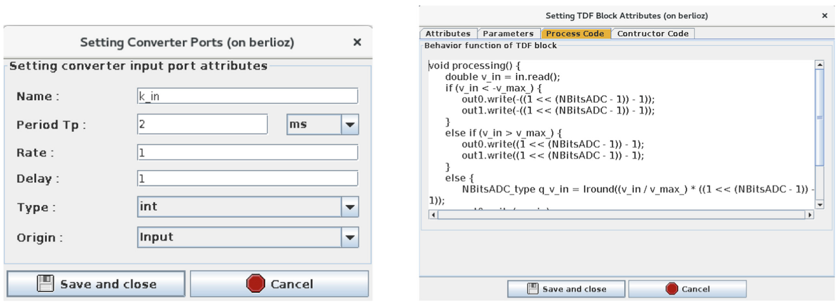
Fig. 4. TDF module parameters [27].

Analog/Mixed Signal components are not represented on the partitioning level since the decision to have them implemented in hardware or software is not in the hands of the designer of the embedded platforms. AMS components are thus captured in deployment diagrams, from which the hardware top cells and the descriptions of the mapping of software objects to processors, memories and communication elements are generated for simulation purpose.

Our contribution is twofold: we represent SystemC-AMS components in Deployment Diagrams and are able to generate the communication between digital and analog parts in the simulation/prototyping code.

4.1 Representing Analog Components

In our extension to TTool, analog and digital parts of a system are first designed in different **panels**. As a consequence, we have enhanced the graphical interface


Setting Converter Ports (on berlioz)

Setting converter input port attributes

Name :	k_in
Period Tp :	2 ms
Rate :	1
Delay :	1
Type :	int
Origin :	Input

Setting TDF Block Attributes (on berlioz)

Behavior function of TDF block

```
void processing() {
    double v_in = in.read();
    if (v_in < -v_max_) {
        out0.write(-((1 << (NBitsADC - 1)) - 1));
        out1.write(-((1 << (NBitsADC - 1)) - 1));
    }
    else if (v_in > v_max_) {
        out0.write((1 << (NBitsADC - 1)) - 1);
        out1.write((1 << (NBitsADC - 1)) - 1);
    }
    else {
        NBitsADC_type q_v_in = lround((v_in / v_max_) * ((1 << (NBitsADC - 1)) -
1));
```

Fig. 5. TDF port parameters (left) and processing function (right) [27].

of TTool with an abstract way to capture SystemC-AMS blocks with DE components, TDF modules and converter ports. Each TDF cluster must designed in its own panel because SystemC-AMS must calculate a separate schedule [3] for each of them.

As mentioned before, TDF modules can be connected together or with DE modules relying on TDF, DE and ports, respectively. The panel provides graphical representations of these elements. The graphical interface also offers a toolbar to select the different components (modules, ports) and connectors between ports.

Figure 3 shows a TTool AMS panel for the design of the introductory example, which contains two TDF modules (gray blocks) and a DE module (white block) interconnected through their respective ports and signals.

Module Parameters. The name and Timestep of a module can be set and its time resolution selected (s, μs, ns). The parameters of a TDF module such as its internal variables or template parameters can also be set up, as shown in Fig. 4.

A TDF block has as its attributes name and Timestep. As attributes, variables and constants can be declared.

Port Parameters. Port parameters can be captured as shown in Fig. 5. For readability, the port Timestep and Delay do not appear in the TDF block visible on the panel but can be obtained and specified by clicking on the port (Fig. 5).

Converter ports have the same attributes, while attributes of DE ports are slightly different (no need to specify Rate and Delay, but indicating the sensitivity to a clock signal is required). TDF and converter ports have a name, a Timestep, a Rate and a Delay. Furthermore, it has to be specified whether it is an input or an output port (called *origin* in the window), and which is the type of data to be transmitted.

Processing Function. Representing analog components in an abstract way is quite difficult since most components are more or less unique. Thus, we decided

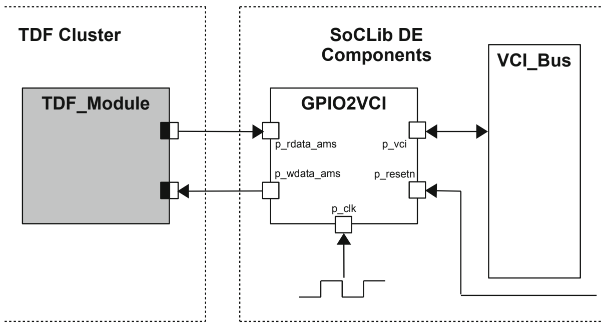


Fig. 6. GPIO2VCI component.

that it would be best if users could directly enter a code to describe functions' behavior. For instance, Fig. 5 shows on the right the processing function for a n-bit analog-digital converter as described by [5].

Valid Schedule. TTool takes as input a SysML system representation to compute a valid *schedule* for each cluster. This determines the correct execution order of TDF modules within the cluster, such that data flow characteristics (sampling rate, sampling period, etc.) are consistent. To compute this schedule, TTool relies on the classical sequential scheduling algorithm of [37] known as *list scheduling*. This algorithm uses an ordered list of the nodes to generate the schedule. Nodes are the TDF blocks and arcs are the signals. TTool builds this list based on the order in which the TDF blocks are created on the panel. Note that there can be several valid schedules. In the example of Fig. 1, a valid schedule would be ABABB.

4.2 Connecting AMS Components to the MPSoC

If the deployment model contains only SystemC-AMS clusters, TTool generates stand-alone SystemC AMS TDF code of the components as well as the SystemC-AMS top cells from the mixed graphical/textual descriptions, and supplies a Makefile. In case software code is also deployed, processors, buses and memories must also be generated. In order to run application software, we thus combine TDF clusters with a MPSoC suitable for full-system simulation.

For this purpose, the SoCLib library provides hardware models, written in SystemC. In particular, it allows the use of a micro kernel [8], able to load and execute cross-compiled software for several processor cores (MIPS, ARM, ...). SoCLib is based on the shared memory paradigm. Components are interconnected based on the *Virtual Component Interconnect* (VCI) [49] protocol. These components can be initiators i.e. they issue requests (e.g. CPUs) or targets that respond to these requests (e.g. RAM memory), sometimes both (DMA, coprocessor wrapper).

The main idea for the integration of SystemC-AMS and SoCLib components into TTool is that the analog components will act as **targets** for the SoCLib initiator digital components (CPUs, hardwareaccelerators, DMA, ...). The generated top cell is thus composed of SoCLib modules and interfaces to the SystemC-AMS clusters. It is also important to mention that a TDF cluster may contain custom DE modules which are not part of the SoCLib library.

In order to connect both worlds, we have introduced a generic *adapter* module that can be used as an interface between SystemC-AMS modules and SoCLib interconnect components [16]. This component is modeled as a **general-purpose input/output** (GPIO) adapter to VCI, called **GPIO2VCI** in the following.

Figure 6 shows the model of the GPIO2VCI component which plays the role an interface between the SystemC-AMS modules (*TDF_Module* belonging to a TDF Cluster) and the SoCLib VCI interconnect component (*VCI_Bus*). Data are exchanged via ports *p_rdata_ams* and *p_wdata_ams*, respectively, *p_vci* communicates with the SoCLib/VCI world. There is also a clock and a reset port. The component is manually inserted in the graphical interface of the panel, then its instantiation and connection, in particular the required lines in the top cell, are automatically generated.

The GPIO2VCI fulfills the rules for writing cycle-bit precise SystemC simulation models of SoCLib. These writing rules, listed in [28], specify that cycle-bit accurate components are built by one or several **Finite State Machines** (FSM) and have clearly defined internal registers. The FSM can be described by three types of functions. The `transition` function, which is triggered once per cycle on the rising edge of the clock, computes the next values of the registers, depending on their current values and the values of the input signals. The `genMoore` function, which is triggered once per cycle on the falling edge of the clock, computes the values of output signals that depend on the internal registers. Finally, the `genMealy` function, which is triggered once per cycle on the falling edge of the clock, computes the values of output signals that depend on the internal registers and the values of the input signals.

4.3 Solving Causality Problems

Due to their different Model of Computation, AMS components require to execute their simulated behavior apart from the rest of the system: yet, they regularly have to synchronize with the digital platform. The SystemC kernel is thus **controlling** the AMS kernel which runs continuously until it is interrupted by an access to a converter port by a TDF cluster.

When a TDF module accesses its input converter port, the DE simulation time advances until it is equal to the TDF simulation time of the input converter port. Later, if an access to an output converter port occurs with a TDF simulation time that is less than the new DE simulation time, a time synchronization issue occurs. To avoid this situation, the TDF simulation time of the output converter ports always needs to be greater or equal than the DE simulation time.

This problem was exposed in [4] and resolved with the help of colored timed Petri Nets [33] derived from the SystemC AMS code.

According to [17], when a SystemC-AMS simulation is being executed, the execution of the SystemC DE simulation kernel is blocked while the SystemC-AMS simulation kernel continues running. As a consequence, during this period the DE simulation time (t_{DE}) does not advance at all, while the TDF simulation time (t_{TDF}) runs according to the Timesteps of the TDF modules and ports. On access to a TDF converter port, the SystemC-AMS simulation kernel is interrupted and yields to the SystemC DE simulation kernel. This way, t_{DE} advances until it is equal to t_{TDF}. In general, t_{TDF} runs ahead of t_{DE}, but in some scenarios, $t_{TDF} \geq t_{DE}$ i.e. t_{DE} may be greater than t_{TDF}: this is a causality problem.

In [4], synchronization at converter ports is modeled with the help of Colored Timed Petri Nets derived from the SystemC-AMS code. Causality issues between TDF and DE MoC are then automatically checked. However, this is done on SystemC-AMS code, whereas [15] proposes a way to detect causality issues from SysML models and also shows that only accesses to TDF *input* converter ports affect synchronization.

The following algorithm presented in [16], of which a detailed version is shown in [15], solves causality issues by iterating over additional Delays and recomputing schedules until all causality issues are solved.

```
 1: procedure DETECTTIMESYNCISSUES
 2:     for each Module in Static Schedule do
 3:         for each Converter Port do
 4:             if Input Converter Port then
 5:                 advance $t_{\mathrm{DE}}$
 6:                 compute $max\_t_{\mathrm{DE}}$
 7:             else if Output Converter Port then
 8:                 compute $t_{\mathrm{TDF}}$ of port
 9:                 if !($t_{\mathrm{TDF}} \geq max\_t_{\mathrm{DE}}$) then
10:                     Time synchronization issue detected
11:                     Suggest port Delay to fix it
12:                 end if
13:             end if
14:         end for
15:     end for
16: end procedure
```

Based on the static schedule for one complete TDF cluster period, each time a TDF module is executed, for each accessed input converter port, the DE simulation time (t_{DE}) advance as shown in line 5, and the maximum t_{DE} is stored as shown in line 6. Then, for each accessed output converter port, the TDF simulation time (t_{TDF}) is computed (see line 8). The t_{TDF} of each port should be greater than or equal to the maximum stored DE simulation time, as shown in line 9. If this condition fails, there is a causality problem and a Delay in the output converter port where the issue was detected is suggested.

4.4 MPSoC Virtual Prototype

GPIO2VCI components are visible in the AMS diagram, as shown in Fig. 7, where our initial cluster is connected to a mono processor platform. Yet, only the connection is represented on the AMS panel by the GPIO2VCI. Also, there can be more than one such connections, one for each TDF cluster. Clicking on one of the GPIO2VCI components opens the corresponding TDF cluster.

Conversely, TDF clusters are displayed in the Deployment Diagram, see Fig. 8. Here, we map a monolithic toy software (a hello world message followed by the printout of *values* generated by a sine wave generator in the AMS cluster), represented by a block named *software* on a mono processor named *CPU0*.

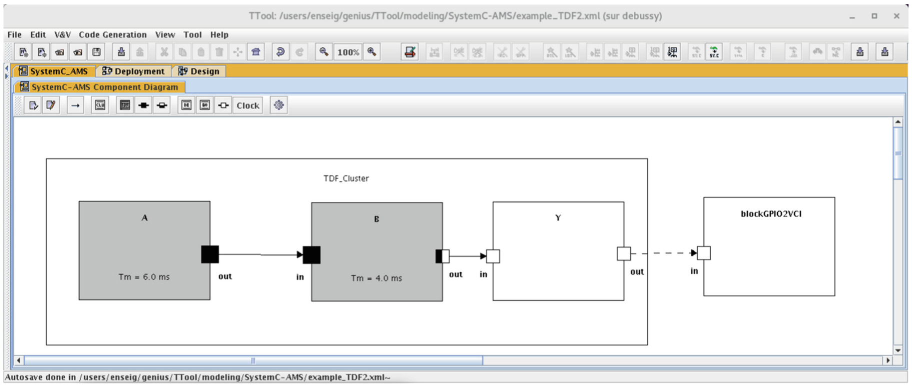

TTool: /users/enseig/genius/TTool/modeling/SystemC-AMS/example_TDF2.xml (sur debussy)
File Edit V&V Code Generation View Tool Help
100%
SystemC_AMS
Deployment
Design
SystemC-AMS Component Diagram
Clock
TDF_Cluster
A
Tm = 6.0 ms
out
in
B
Tm = 4.0 ms
out
in
Y
out
in
blockGPIO2VCI
Autosave done in /users/enseig/genius/TTool/modeling/SystemC-AMS/example_TDF2.xml~

Fig. 7. Adding a GPIO2VCI component.

4.5 Simulation of the Virtual Prototype

Since model-driven approaches expect to ideally provide model validation **before** code generation (and thus simulation), we propose a way to statically identify synchronization problems [15]. Basically, based on the static schedule for one complete TDF cluster period, each time a TDF module is executed, for each accessed input converter port, the DE simulation time (t_{DE}) advances, and the maximum t_{DE} is stored. Then, for each accessed output converter port, the TDF simulation time (t_{TDF}) of each port should be greater or equal than the maximum stored t_{DE}. If this condition fails, it means there is a causality problem with the time synchronization and a delay in the output converter port where the issue was detected will be suggested to the designer in order to resolve the problem. The schedulability of the analog part is validated using the schedulability check of SystemC-AMS [37], thus before code is generated.

Figures 9 and 10 show the simulation of the integration of SystemC-AMS and SoCLib SoC components: a write operation to the GPIO2VCI thus to the analog part, followed by a read from the GPIO2VCI.

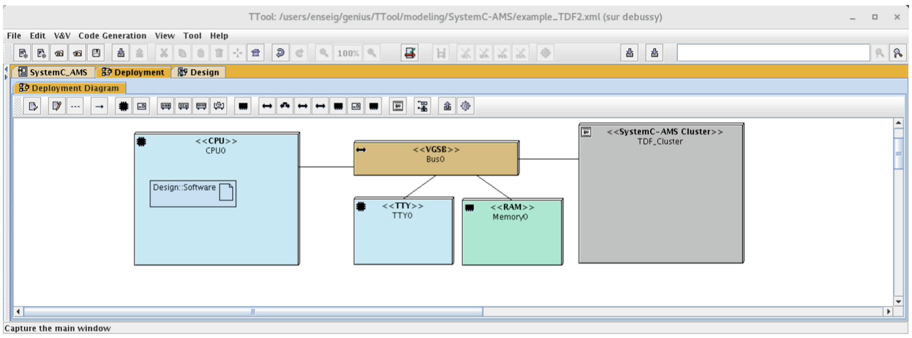
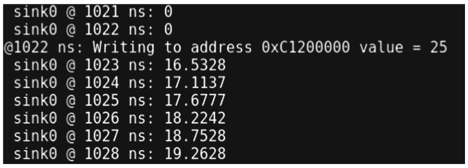
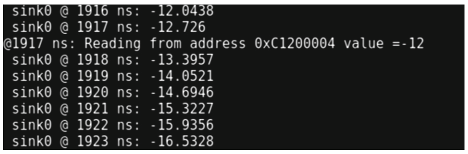


Fig. 8. TTool deployment panel featuring a TDF cluster.



```
 sink0 @ 1021 ns: 0
 sink0 @ 1022 ns: 0
@1022 ns: Writing to address 0xC1200000 value = 25
 sink0 @ 1023 ns: 16.5328
 sink0 @ 1024 ns: 17.1137
 sink0 @ 1025 ns: 17.6777
 sink0 @ 1026 ns: 18.2242
 sink0 @ 1027 ns: 18.7528
 sink0 @ 1028 ns: 19.2628
```

Fig. 9. Host machine console: Write to the GPIO2VCI component.



```
 sink0 @ 1916 ns: -12.0438
 sink0 @ 1917 ns: -12.726
@1917 ns: Reading from address 0xC1200004 value =-12
 sink0 @ 1918 ns: -13.3957
 sink0 @ 1919 ns: -14.0521
 sink0 @ 1920 ns: -14.6946
 sink0 @ 1921 ns: -15.3227
 sink0 @ 1922 ns: -15.9356
 sink0 @ 1923 ns: -16.5328
```

Fig. 10. Host machine console: Read from the GPIO2VCI component.

4.6 Trace Generation

While it possible to generate cycle accurate *vcd* traces of the digital signals in the original version of TTool, the integration of SystemC-AMS necessitates the tracing of the analog, thus continuous, signals. Thus, our tool contains additional mechanisms for trace generation of the analog part of the simulation.

SystemC-AMS tracing using the `sca_trace` primitives is invoked for each analog cluster. This function, if activated from the TTool graphical interface, allows to create one trace file per cluster. Code lines are generated and inserted in the SystemC-AMS code of the cluster.

Listing 1.1 shows how tracing is handled for the top cell under consideration A tabular trace file is created with a given name. signals connecting the GPIO2VCI component to the TDF cluster are added to the trace, then the tracing functions that have been created in the cluster's SystemC-AMS code are invoked. Traces can then be displayed with a tool adapted to analog traces, like

GAW - Gtk Analog Wave viewer [43]. As usual, traces of the SystemC digital part can displayed with e.g. gtkwave [11].

```
sca_util::sca_trace_file *tfp = sca_util::sca_create_tabular_trace_file("analog_trace");
sca_util::sca_trace(tfp,signal_to_ams0,"signal_to_ams0");
sca_util::sca_trace(tfp,signal_from_ams0,"signal_from_ams0");
Cluster0_0.trace_Cluster0(tfp);
...
sca_util::sca_close_tabular_trace_file(tfp);
```

Listing 1.1. Tracing for the AMS components invoked in the top cell.

5 Case Study

Our contribution to tackle digital and analog systems is illustrated by an automotive embedded system designed in the scope of the EVITA European project [20] and for which code generation was presented in [38]. Recent on-board Intelligent Transport (IT) architectures comprise a very heterogeneous landscape of communication network technologies (e.g., LIN, CAN, MOST, and FlexRay) that interconnect in-car Electronic Control Units (ECUs).

We apply in the following, step by step, the general methodology developed in [25] concerning the digital part along with the new techniques introduced in [27].

Among the use cases addressed by EVITA, we selected the automatic braking function [34]. Basically, this function works as follows: an obstacle is detected by another automotive system which broadcasts that information to neighboring cars. A car receiving such information has to decide whether it is concerned with this obstacle. This decision includes a plausibility check function that takes into account various parameters, such as the direction and speed of the car, and also information previously received from neighboring cars. Once the decision to brake has been taken, the braking order is forwarded to relevant ECUs. Last but not least, the presence of this obstacle is forwarded to other neighboring cars in case they have not yet received this information.

5.1 Partitioning

The *functional view* in Fig. 11 describes of a set of abstract communicating tasks; green boxes representing TLM modules). Functional abstraction allows us to avoid capturing the exact data processing algorithms, but rather to consider only abstract computation complexity. Each individual task describes its abstract functional behavior using communication operators, computation elements, and control elements. Thanks to data abstraction, we consider only the size of the data sent or received, and ignore details such as type, values, or names.

Then, mapping intends to partition functions between software and hardware implementations. Figure 12 shows the deployment diagram. The architecture is modeled as a graph built upon execution (light blue), communication (orange), and storage (light turquoise) nodes. Execution nodes are for example CPUs and hardware accelerators. Our extension allows a representation of analog/mixed

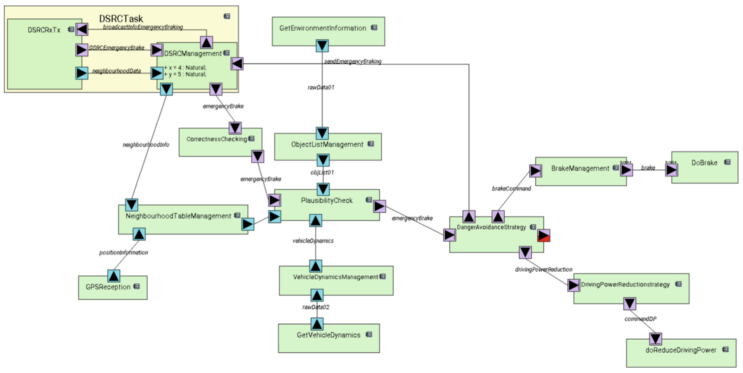


Fig. 11. Functional view. (Color figure online)

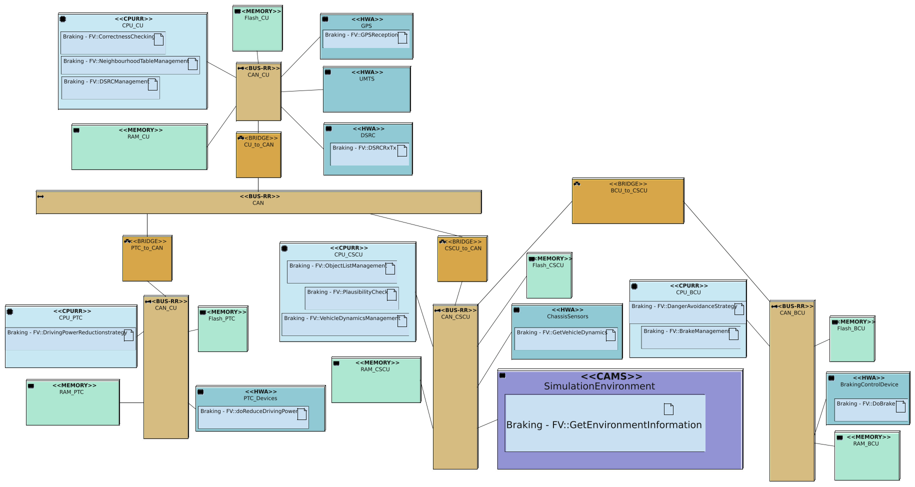


Fig. 12. Partitioning level mapping view. (Color figure online)

signal modules, which are execution nodes too. Communication nodes include bridges and buses, storage nodes are memories.

A function mapped onto a processor will be implemented in software, and a function mapped onto a hardware accelerator (darker turquoise) is implemented in hardware. Functions to be implemented in hardware are either digital or analog functions. In our example, all sensors obtaining information from the environment are modeled as analog blocks.

An evolution with regards to [27] is that analog blocks can now be made explicit on the partitioning level as a particular kind of hardware accelerators, named *CAMS* (abbreviating SystemC-AMS) as shown on the bottom center of Fig. 12: the simulation environment is subsumed in the light purple block named *SimulationEnvironment*, which is slightly enlarged for better readability.

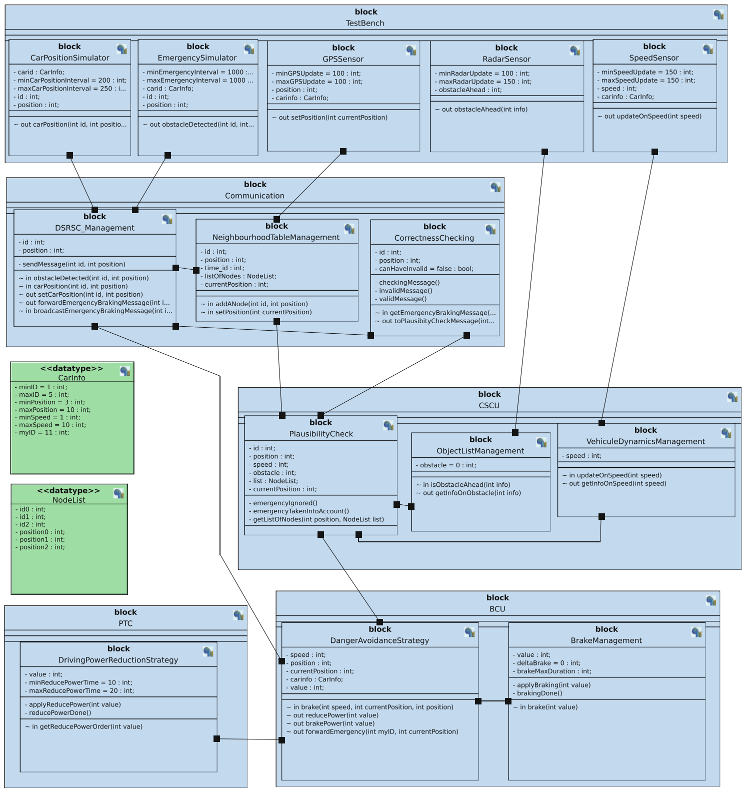

block TestBench
block CarPositionSimulator
- carid : CarInfo;
- minCarPositionInterval = 200 : int;
- maxCarPositionInterval = 250 : i...
- id : int;
- position : int;
~ out carPosition(int id, int positio...
block EmergencySimulator
- minEmergencyInterval = 1000 :...
- maxEmergencyInterval = 1000 ...
- carid : CarInfo;
- id : int;
- position : int;
~ out obstacleDetected(int id, int...
block GPSSensor
- minGPSUpdate = 100 : int;
- maxGPSUpdate = 100 : int;
- position : int;
- carinfo : CarInfo;
~ out setPosition(int currentPosition)
block RadarSensor
- minRadarUpdate = 100 : int;
- maxRadarUpdate = 150 : int;
- obstacleAhead : int;
~ out obstacleAhead(int info)
block SpeedSensor
- minSpeedUpdate = 150 : int;
- maxSpeedUpdate = 150 : int;
- speed : int;
- carinfo : CarInfo;
~ out updateOnSpeed(int speed)
block Communication
block DSRSC_Management
- id : int;
- position : int;
- sendMessage(int id, int position)
~ in obstacleDetected(int id, int position)
~ in carPosition(int id, int position)
~ out setCarPosition(int id, int position)
~ out forwardEmergencyBrakingMessage(int i...
~ in broadcastEmergencyBrakingMessage(int i...
block NeighbourhoodTableManagement
- id : int;
- position : int;
- time_id : int;
- listOfNodes : NodeList;
- currentPosition : int;
~ in addANode(int id, int position)
~ in setPosition(int currentPosition)
block CorrectnessChecking
- id : int;
- position : int;
- canHaveInvalid = false : bool;
- checkingMessage()
- invalidMessage()
- validMessage()
~ in getEmergencyBrakingMessage(...
~ out toPlausibityCheckMessage(int...
<<datatype>> CarInfo
- minID = 1 : int;
- maxID = 5 : int;
- minPosition = 3 : int;
- maxPosition = 10 : int;
- minSpeed = 1 : int;
- maxSpeed = 10 : int;
- myID = 11 : int;
<<datatype>> NodeList
- id0 : int;
- id1 : int;
- id2 : int;
- position0 : int;
- position1 : int;
- position2 : int;
block CSCU
block PlausibilityCheck
- id : int;
- position : int;
- speed : int;
- obstacle : int;
- list : NodeList;
- currentPosition : int;
- emergencyIgnored()
- emergencyTakenIntoAccount()
- getListOfNodes(int position, NodeList list)
block ObjectListManagement
- obstacle = 0 : int;
~ in isObstacleAhead(int info)
~ out getInfoOnObstacle(int info)
block VehiculeDynamicsManagement
- speed : int;
~ in updateOnSpeed(int speed)
~ out getInfoOnSpeed(int speed)
block PTC
block DrivingPowerReductionStrategy
- value : int;
- minReducePowerTime = 10 : int;
- maxReducePowerTime = 20 : int;
- applyReducePower(int value)
- reducePowerDone()
~ in getReducePowerOrder(int value)
block BCU
block DangerAvoidanceStrategy
- speed : int;
- position : int;
- currentPosition : int;
- carinfo : CarInfo;
- value : int;
~ in brake(int speed, int currentPosition, int position)
~ out reducePower(int value)
~ out brakePower(int value)
~ out forwardEmergency(int myID, int currentPosition)
block BrakeManagement
- value : int;
- deltaBrake = 0 : int;
- brakeMaxDuration : int;
- applyBraking(int value)
- brakingDone()
~ in brake(int value)

Fig. 13. Block diagram from [38].

5.2 Software Design

Once the partitioning is done, software can be designed and verified with TTool. Figure 13 shows the former software block diagram taken from [38], with the five sensors at the top: there, sensors are captured as software components. Now, they can be removed from the software design diagram since the partitioning decision has already been taken for the analog blocks: they do not need to be considered any more during software design.

Other software components are grouped according to their destination ECU:

- **Communication ECU** manages communication with neighboring vehicles.
- **Chassis Safety Controller ECU (CSCU)** processes emergency messages and sends orders to brake to ECUs.
- **Braking Controller ECU (BCU)** contains two blocks: *DangerAvoidanceStrategy* determines how to efficiently and safely reduce the vehicle speed, or brake if necessary.
- **Power Train Controller ECU (PTC)** enforces the engine torque modification request.

To prototype the software components with the other platform elements (hardware components, operating system), we must map the software components to a model of the target system. Mapping can be performed using the deployment features introduced in [24]: such a **deployment diagram** is a SysML representation of hardware components, their interconnection, software tasks and communication channels between software tasks.

5.3 Modeling Sensors

Before, since sensors were captured as software tasks, code generation from software design resulted in having a C/POSIX code representing the behavior of these sensors, leading to too unrealistic simulations All five sensors are now replaced by more realistic analog models in the form of five independent TDF clusters.

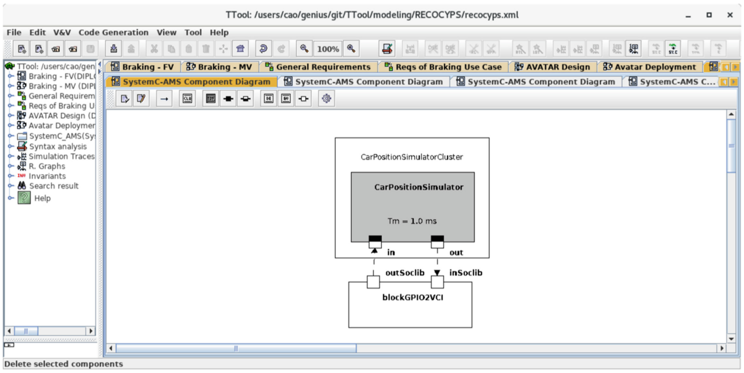

TTool: /users/cao/genius/git/TTool/modeling/RECOCYPS/recocyps.xml
File Edit V&V Code Generation View Tool Help
Braking - FV
Braking - MV
General Requirements
Reqs of Braking Use Case
AVATAR Design
Avatar Deployment
SystemC-AMS Component Diagram
SystemC-AMS Component Diagram
SystemC-AMS C...
CarPositionSimulatorCluster
CarPositionSimulator
Tm = 1.0 ms
in
out
outSoclib
inSoclib
blockGPIO2VCI
Delete selected components

Fig. 14. TTool panel with model of the *CarPositionSimulator* sensor.

Figure 14 shows the AMS panel of the textitCarPositionSimulator sensor that gives information on surrounding cars *id* (e.g., car position).

From TDF information (Rate, Delay, ...), TTool infers, if possible, missing parameters, and then computes a coherent schedule, and finally generates SystemC-AMS code, comprising ports, Delays and interfaces [15]. Cluster output is read by the *DSRSC_Management* block (see Fig. 13). Often, complex data structures of more than one parameter are transmitted in channels (here, *id* and *position*). Currently, they have to be transmitted one by one, basic type by basic type. Thus, *id* and *position* require two sequential write operations to the out port in the processing code and two corresponding read operations in the entry code.

We can easily model the randomized choice of an integer between 1 and 5 (*id*) and between 3 and 10 (*position*) stemming from the data type of Fig. 13. The code of this simple processing function is shown on the right of the figure in a separate window. The *write* primitive sends one integer value to the *out* converter port.

5.4 Interaction of Analog Blocks with the Software Design Level

In contrast to the purely digital model of the same application, the functional blocks pertaining to the sensors are no longer represented in the software design level block diagram, since they are represented by analog blocks captured in five separate SystemC-AMS panels. In Fig. 13, thus, the upper row of tasks named *TestBench* disappears.

A library named *libsyscams* has been provided to contain read and write primitives on the side of the MPSoC, the *read_gpio2vci* and *write_gpio2vci* functions. As shown above, *CarPositionSimulator* issues two random values from its output port, *EmergencySimulator* does the same. By executing these software functions, the CPU of the digital platform is able to exchange (i.e. read or write) values with the analog components.

On the side of the MPSoC platform, according to TTool's semantics, the *DSRSC_Management* block nondeterministically reads from either block, or read a *broadcastEmergencyBrakingMessage* from a third, the *DangerAvoidanceStrategy* block. In the current version, the first two blocks being replaced by sensors modeled in SystemC-AMS, this semantics should be preserved.

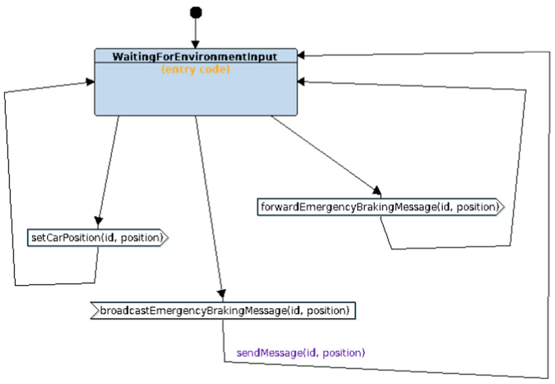
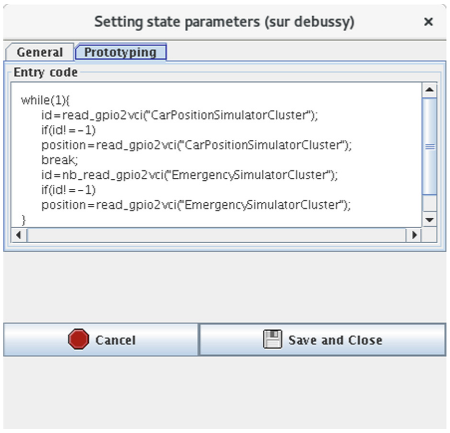

WaitingForEnvironmentInput
{entry code}
setCarPosition(id, position)
forwardEmergencyBrakingMessage(id, position)
broadcastEmergencyBrakingMessage(id, position)
sendMessage(id, position)

Fig. 15. *DSRSC_Management* block state machine containing link to the entry code.

Let us now consider the state machine of the *DSRSC_Management* block (Fig. 15). In [27], we show how to use **entry code** that can be contained in a state to call *libsyscams*. This is the case of the *WaitForEnvironmentInput* state. We read nondeterministically either the input from *CarPositionSimulator* or *EmergencySimulator*, whenever values are available on either. This nondeterminism, which was in the past expressed by the semantics of TTool's channels between software blocks, must now be reflected in the entry code of the software block's state machine as well. Figure 16 shows the successive operations: we call the *read_gpio2vci* primitive and check whether data was successfully read and in that case, go on to the next operation. If there are several parameters (here *id* and *position*), they must be read sequentially.



Entry code

```
while(1){
    id=read_gpio2vci("CarPositionSimulatorCluster");
    if(id!=-1)
    position=read_gpio2vci("CarPositionSimulatorCluster");
    break;
    id=nb_read_gpio2vci("EmergencySimulatorCluster");
    if(id!=-1)
    position=read_gpio2vci("EmergencySimulatorCluster");
}
```

Fig. 16. *WaitingForEnvironmentInput* state entry code.

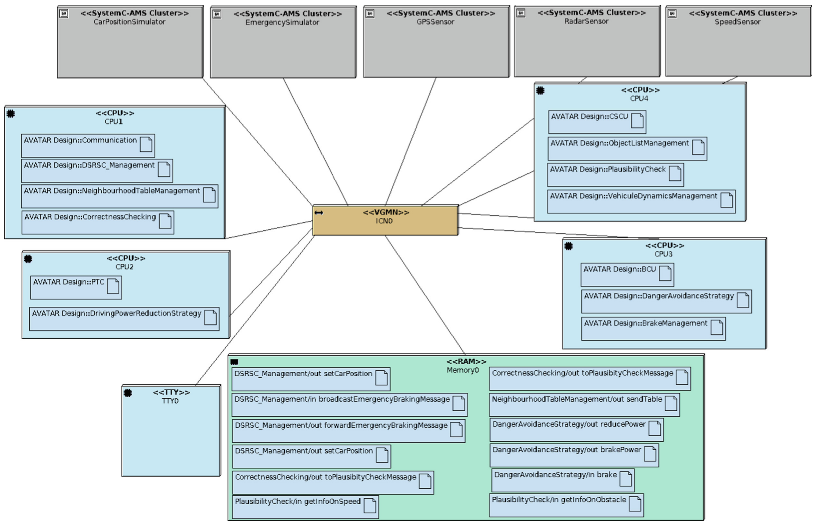
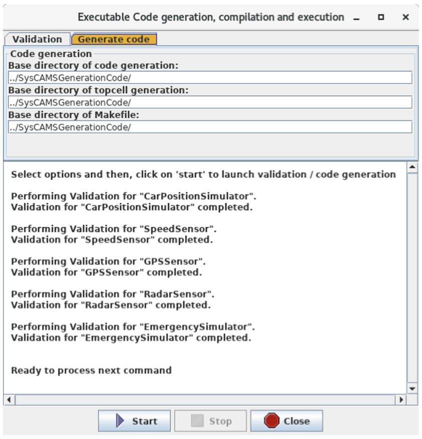

<<SystemC-AMS Cluster>> CarPositionSimulator
<<SystemC-AMS Cluster>> EmergencySimulator
<<SystemC-AMS Cluster>> GPSSensor
<<SystemC-AMS Cluster>> RadarSensor
<<SystemC-AMS Cluster>> SpeedSensor
<<CPU>> CPU1
AVATAR Design::Communication
AVATAR Design::DSRSC_Management
AVATAR Design::NeighbourhoodTableManagement
AVATAR Design::CorrectnessChecking
<<CPU>> CPU4
AVATAR Design::CSCU
AVATAR Design::ObjectListManagement
AVATAR Design::PlausibilityCheck
AVATAR Design::VehiculeDynamicsManagement
<<VGMN>> ICN0
<<CPU>> CPU2
AVATAR Design::PTC
AVATAR Design::DrivingPowerReductionStrategy
<<CPU>> CPU3
AVATAR Design::BCU
AVATAR Design::DangerAvoidanceStrategy
AVATAR Design::BrakeManagement
<<TTY>> TTY0
<<RAM>> Memory0
DSRSC_Management/out setCarPosition
DSRSC_Management/in broadcastEmergencyBrakingMessage
DSRSC_Management/out forwardEmergencyBrakingMessage
DSRSC_Management/out setCarPosition
CorrectnessChecking/out toPlausibityCheckMessage
PlausibilityCheck/in getInfoOnSpeed
CorrectnessChecking/out toPlausibityCheckMessage
NeighbourhoodTableManagement/out sendTable
DangerAvoidanceStrategy/out reducePower
DangerAvoidanceStrategy/out brakePower
DangerAvoidanceStrategy/in brake
PlausibilityCheck/in getInfoOnObstacle

Fig. 17. Deployment diagram of the active braking application.

5.5 Deployment

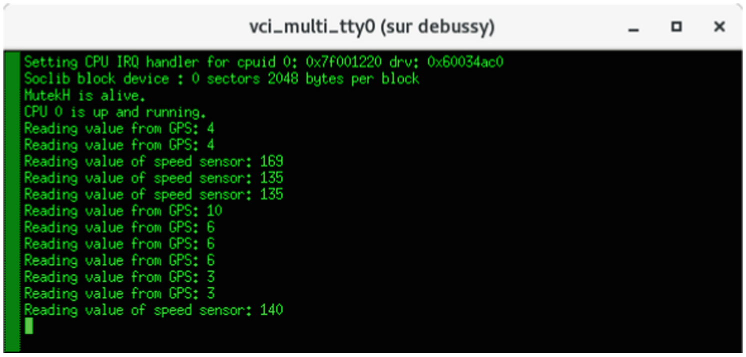
Figure 17 shows the extended *deployment diagram* giving an overview of the mapping of software tasks and channels. Where the software tasks are mapped onto the CPU, the channels between the tasks on the memory. TDF clusters are displayed as gray boxes along with digital components, interconnected to the central (digital) interconnect through GPIO2VCI components as detailed below. For a better overview, the diagram contains sensors as gray boxes, each one corresponding to a SystemC-AMS cluster connected via a GPIO2VCI. Clicking



Code generation
Base directory of code generation:
../SysCAMSGenerationCode/
Base directory of topcell generation:
../SysCAMSGenerationCode/
Base directory of Makefile:
../SysCAMSGenerationCode/

```
Select options and then, click on 'start' to launch validation / code generation

Performing Validation for "CarPositionSimulator".
Validation for "CarPositionSimulator" completed.

Performing Validation for "SpeedSensor".
Validation for "SpeedSensor" completed.

Performing Validation for "GPSSensor".
Validation for "GPSSensor" completed.

Performing Validation for "RadarSensor".
Validation for "RadarSensor" completed.

Performing Validation for "EmergencySimulator".
Validation for "EmergencySimulator" completed.


Ready to process next command
```

Fig. 18. Validation and code generation window.



```
Setting CPU IRQ handler for cpuid 0: 0x7f001220 drv: 0x60034ac0
Soclib block device : 0 sectors 2048 bytes per block
MutekH is alive.
CPU 0 is up and running.
Reading value from GPS: 4
Reading value from GPS: 4
Reading value of speed sensor: 169
Reading value of speed sensor: 135
Reading value of speed sensor: 135
Reading value from GPS: 10
Reading value from GPS: 6
Reading value from GPS: 6
Reading value from GPS: 6
Reading value from GPS: 3
Reading value from GPS: 3
Reading value of speed sensor: 140
```

Fig. 19. TTY of the SoCLib simulation showing system boot and first input from the sensors.

on the box opens the corresponding SystemC-AMS panel. A fifth CPU which used to simulate the sensor execution is no longer in use.

The generated MPSoC platform consists of a digital SoC based on SoCLib components connected to the analog hardware components, modeled using SystemC-AMS code. On the SoCLib side, a MIPS32 CPU, a 1 MB RAM memory and a TTY terminal are modeled in SystemC. This virtual prototype is capable of running **software** (limited to communicating some values and command in the case study) and a lightweight **operating system** [8].

5.6 Running the Application

TTool first checks the coherency of the block and port parameters before computing a valid TDF schedule for each TDF cluster, taking into account synchronization issues between the TDF and DE world [27]. This is done in a so-called

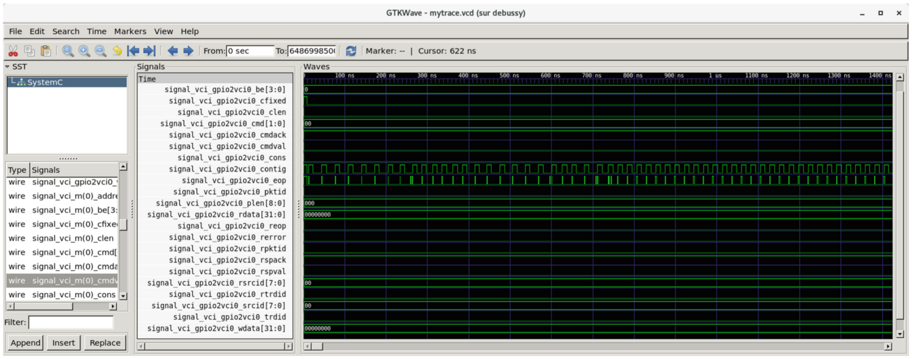

GTKWave - mytrace.vcd (sur debussy)
File Edit Search Time Markers View Help
From: 0 sec
To: 6486998500
Marker: -- | Cursor: 622 ns
SST
SystemC
Type Signals
wire signal_vci_gpio2vci0_
wire signal_vci_m(0)_addr
wire signal_vci_m(0)_be[3:
wire signal_vci_m(0)_cfixe
wire signal_vci_m(0)_clen
wire signal_vci_m(0)_cmd[
wire signal_vci_m(0)_cmda
wire signal_vci_m(0)_cmdv
wire signal_vci_m(0)_cons
Filter:
Append Insert Replace
Signals
Time
signal_vci_gpio2vci0_be[3:0]
signal_vci_gpio2vci0_cfixed
signal_vci_gpio2vci0_clen
signal_vci_gpio2vci0_cmd[1:0]
signal_vci_gpio2vci0_cmdack
signal_vci_gpio2vci0_cmdval
signal_vci_gpio2vci0_cons
signal_vci_gpio2vci0_contig
signal_vci_gpio2vci0_eop
signal_vci_gpio2vci0_pktid
signal_vci_gpio2vci0_plen[8:0]
signal_vci_gpio2vci0_rdata[31:0]
signal_vci_gpio2vci0_reop
signal_vci_gpio2vci0_rerror
signal_vci_gpio2vci0_rpktid
signal_vci_gpio2vci0_rspack
signal_vci_gpio2vci0_rspval
signal_vci_gpio2vci0_rsrcid[7:0]
signal_vci_gpio2vci0_rtrdid
signal_vci_gpio2vci0_srcid[7:0]
signal_vci_gpio2vci0_trdid
signal_vci_gpio2vci0_wdata[31:0]
Waves
100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns 900 ns 1 us 1100 ns 1200 ns 1300 ns 1400 ns

Fig. 20. Digital trace generated from TTool's simulation.

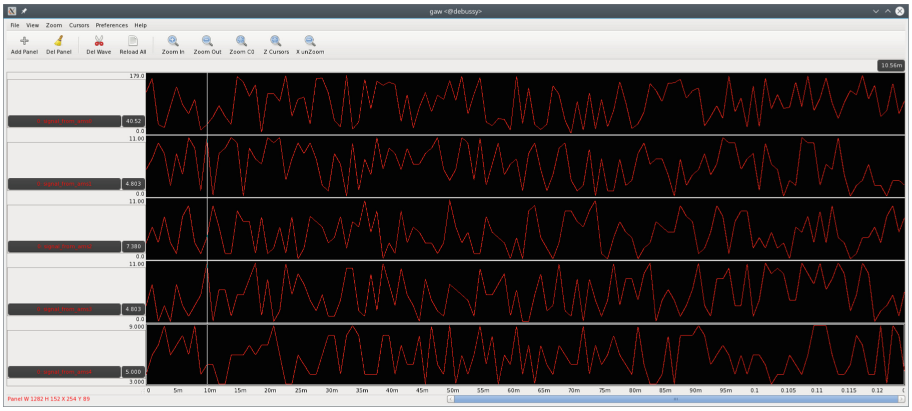

gaw <@debussy>
File View Zoom Cursors Preferences Help
Add Panel Del Panel Del Wave Reload All Zoom In Zoom Out Zoom C0 Z Cursors X unZoom
10.56m
179.0
40.52
0.0
11.00
4.803
0.0
11.00
7.380
0.0
11.00
4.803
0.0
9.000
5.000
3.000
0 5m 10m 15m 20m 25m 30m 35m 40m 45m 50m 55m 60m 65m 70m 75m 80m 85m 90m 95m 0.1 0.105 0.11 0.115 0.12
Panel W 1282 H 152 X 254 Y 89

Fig. 21. Analog trace generated from TTool's simulation.

validation window (Fig. 18). Once the cluster schedule is validated, code generation can be started from another dialog window.

Figure 19 shows the start up of the software application and first incoming measurements of the sensors (randomized values in plausible ranges were used for simulation). Figure 20 shows part of the simulation vcd trace of the AMS version, containing the digital signals; we focus on the signals on one of the *gpio2vci* interfaces. In Fig. 21, the five incoming signals from the five sensors are traced with GAW.

In the light of former work, we examine latencies [26]. The most important latency is the one between the detection of an emergency situation and the moment when the braking really occurs.

As can be seen in Fig. 22, the message is issued by *ForwardEmergencyBrakingMessage*, left hand side of the figure) and received by reading from channel *brake* (right hand side). In the simulated situation, the latency is of $1918 - 1681 = 237\,\mu$sec. Figure 23 finally shows the sequence diagram obtained

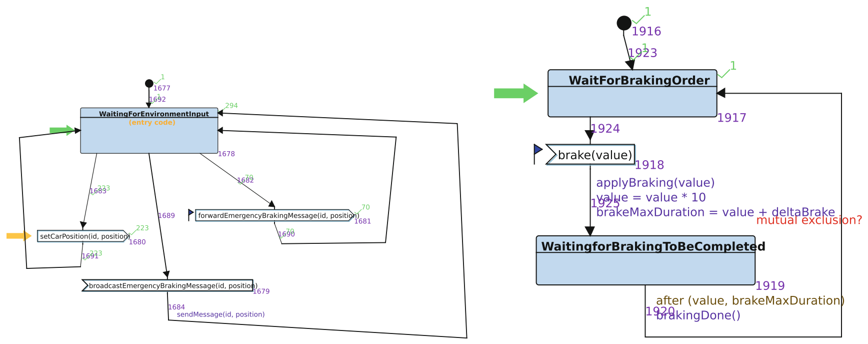

WaitingForEnvironmentInput
(entry code)
setCarPosition(id, position)
forwardEmergencyBrakingMessage(id, position)
broadcastEmergencyBrakingMessage(id, position)
sendMessage(id, position)
WaitForBrakingOrder
brake(value)
applyBraking(value)
value = value * 10
brakeMaxDuration = value + deltaBrake
mutual exclusion?
WaitingforBrakingToBeCompleted
after (value, brakeMaxDuration)
brakingDone()
1916
1923
1917
1924
1918
1925
1919
1920
1677
1692
1678
1682
1683
1689
1690
1681
1680
1691
1679
1684

Fig. 22. Latency checkpoints between emergency detection (write to channel *ForwardEmergencyBrakingMessage*) and braking (reading from channel *brake*). Automates are annotated with values obtained by running the interactive simulation.

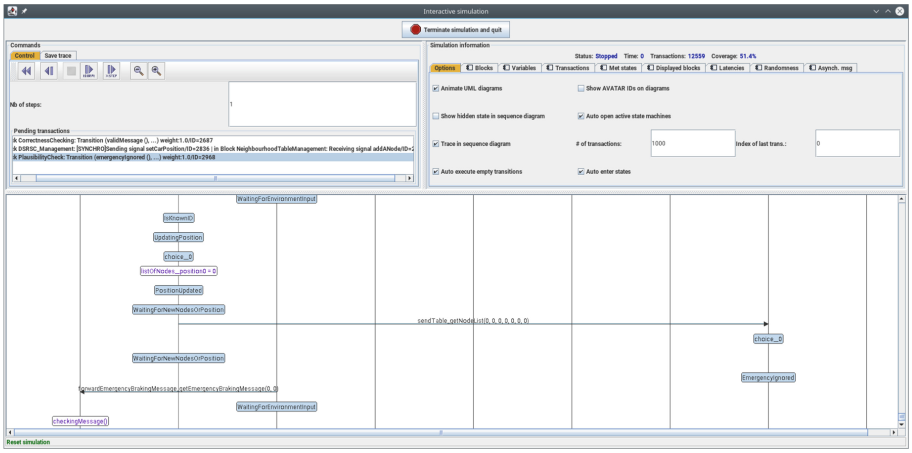

Interactive simulation
Terminate simulation and quit
Commands
Control
Save trace
Simulation information
Status: Stopped Time: 0 Transactions: 12559 Coverage: 51.4%
Options
Blocks
Variables
Transactions
Met states
Displayed blocks
Latencies
Randomness
Asynch. msg
Nb of steps: 1
Pending transactions
k CorrectnessChecking: Transition (validMessage (), ...) weight:1.0/ID=2687
k DSRSC_Management: [SYNCHRO]Sending signal setCarPosition/ID=2836 | in Block NeighbourhoodTableManagement: Receiving signal addANode/ID=2
k PlausibilityCheck: Transition (emergencyIgnored (), ...) weight:1.0/ID=2968
[✓] Animate UML diagrams
[ ] Show AVATAR IDs on diagrams
[ ] Show hidden state in sequence diagram
[✓] Auto open active state machines
[✓] Trace in sequence diagram
# of transactions: 1000
Index of last trans.: 0
[✓] Auto execute empty transitions
[✓] Auto enter states
WaitingForEnvironmentInput
isKnownID
UpdatingPosition
choice__0
listOfNodes__position0 = 0
PositionUpdated
WaitingForNewNodesOrPosition
sendTable_getNodeList(0, 0, 0, 0, 0, 0, 0)
choice__0
WaitingForNewNodesOrPosition
EmergencyIgnored
forwardEmergencyBrakingMessage_getEmergencyBrakingMessage(0, 0)
WaitingForEnvironmentInput
checkingMessage()
Reset simulation

Fig. 23. Emergency situation in a sequence diagram.

at software design level, indicating that there is an emergency message, but that particular message can be ignored.

6 Conclusion and Perspectives

The paper shows the integration of SystemC-AMS (TDF) components into a multi-level modeling tool for complex embedded systems. Starting from a SysML-like representation and progressively refining, we obtain, by model transformation, a cycle-accurate virtual prototype.

Virtual prototyping can be obtained from the last refinement stage, taking into account both analog and digital parts of the system. To this end, a library was created to provide read and write functions between digital and analog components.

Yet, in order to use analog components, C code needs to be inserted in order to capture analog functions. The resulting code is thus no longer correct by construction. In the future, this should be replaced by specific read and write operators. Also, it should be possible to transmit structured data types and multiple parameters more conveniently.

Even if analog components tend to be unique, we think that it will be possible to select a set of typical components such as filters, analog/digital converters, sine sources, and sinks. We plan to provide a library of parametrizable versions of such building blocks.

Yet, TDF models are still strongly oversimplified as in the EVITA industrial case study, further detail was not available. We are currently modeling a medical appliance with a strong proportion of analog blocks, stemming from an Open Source project [13], for which we have access to full implementation details.

Latency measurements are currently limited to the digital part. The feedback of simulation results is still only semi-automatic. Automating and extending this mechanism to the entire system should enable us to propose a full design space exploration environment for Analog/Mixed Signal systems.

References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Accellera Systems Initiative: SystemC AMS extensions Users Guide, Version 1.0. Accellera Systems Initiative, March 2010
4. 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, March 2015
5. Andrade Porras, L.: Principles and implementation of a generic synchronization interface between SystemC AMS models of computation for the virtual prototyping of multi-disciplinary systems. Ph.D. thesis, Université Pierre et Marie Curie (2016)
6. Apvrille, L.: Webpage of TTool (2011)
7. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A.L.: Metropolis: an integrated electronic system design environment. IEEE Comput. **36**(4), 45–52 (2003)
8. Becoulet, A.: Mutekh. http://www.mutekh.org
9. Beyond Dreams Consortium: Beyond Dreams (Design Refinement of Embedded Analogue and Mixed-Signal Systems) (2008–2011). http://projects.eas.iis.fraunhofer.de/beyonddreams
10. Bouquet, F., Gauthier, J.M., Hammad, A., Peureux, F.: Transformation of SysML structure diagrams to VHDL-AMS. In: 2012 Second Workshop on Design, Control and Software Implementation for Distributed MEMS, pp. 74–81. IEEE (2012)
11. Bybell, T.: GTKWave Viewer (2019). http://gtkwave.sourceforge.net
12. Capocchi, L., Santucci, J.F., Poggi, B., Nicolai, C.: DEVSimPY: a collaborative python software for modeling and simulation of DEVS systems. In: 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 170–175. IEEE (2011)

13. echOpen Community: Designing an open-source and low-cost echo-stethoscope (2017). http://www.echopen.org/
14. Concepcion, A.I., Zeigler, B.P.: DEVS formalism: a framework for hierarchical model development. IEEE Trans. Softw. Eng. **14**(2), 228–241 (1988)
15. Porto, R.C.: Integration of SystemC-AMS simulation platforms into TTool. Master's thesis, Technische Universität Kaiserslautern (2018)
16. Porto, R.C., Genius, D., Apvrille, L.: Modeling and virtual prototyping for embedded systems on mixed-signal multicores. In: RAPIDO (2019)
17. Damm, M., Grimm, C., Haas, J., Herrholz, A., Nebel, W.: Connecting SystemC-AMS models with OSCI TLM 2.0 models using temporal decoupling. In: FDL, pp. 25–30 (2008)
18. Davare, A.: A next-generation design framework for platform-based design. In: DVCon, vol. 152 (2007)
19. Einwich, K.: SystemC AMS PoC2.1 Library, COSEDA, Dresden (2016)
20. EVITA: E-safety vehicle intrusion protected applications. http://www.evita-project.org/
21. Fong, C.: Discrete-time dataflow models for visual simulation in ptolemy II. Master's report, Memorandum UCB/ERL M 1 (2001)
22. Fritzson, P., Engelson, V.: Modelica—a unified object-oriented language for system modeling and simulation. In: Jul, E. (ed.) ECOOP 1998. LNCS, vol. 1445, pp. 67–90. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054087
23. Gamatié, A., et al.: A model-driven design framework for massively parallel embedded systems. ACM Trans. Embed. Comput. Syst. **10**(4), 39 (2011)
24. Genius, D., Apvrille, L.: Virtual yet precise prototyping: an automotive case study. In: ERTSS 2016, Toulouse, January 2016
25. Genius, D., Li, L.W., Apvrille, L.: Model-driven performance evaluation and formal verification for multi-level embedded system design. In: 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017), Porto, Portugal (2017)
26. Genius, D., Li, L.W., Apvrille, L.: Multi-level latency evaluation with an MDE approach. In: 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018), Funchal, Portugal (2018)
27. Genius, D., Cortés Porto, R., Apvrille, L., Pêcheux, F.: A tool for high-level modeling of analog/mixed signal embedded systems. In: 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), Prague, Czech Republic (2019)
28. Greiner, A.: Writing efficient cycle-accurate, bit-accurate SystemC simulation models for SoCLib, September 2017. http://www.soclib.fr/trac/dev/wiki/WritingRules/Caba. http://www.soclib.fr/trac/dev/wiki/WritingRules/Caba. As of: 16 October 2018
29. Guo, L., Zhu, Q., Nuzzo, P., Passerone, R., Sangiovanni-Vincentelli, A., Lee, E.A.: Metronomy: a function-architecture co-simulation framework for timing verification of cyber-physical systems. In: Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, p. 24. ACM (2014)
30. H-Inception Consortium: Heterogeneous Inception Project (2012–2015). https://www-soc.lip6.fr/trac/hinception
31. Herrera, F., Villar, E.: A framework for heterogeneous specification and design of electronic embedded systems in SystemC. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **12**(3), 22 (2007)
32. IEEE: SystemC. IEEE Standard 1666-2011 (2011)

33. Jensen, K., Kristensen, L.M.: Coloured Petri Nets. Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009). https://doi.org/10.1007/b95112
34. Kelling, E., et al.: Specification and evaluation of e-security relevant use cases. Technical report, Deliverable D2.1, EVITA Project (2009)
35. Lee, E.A.: Disciplined heterogeneous modeling. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6395, pp. 273–287. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16129-2_20
36. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proc. IEEE **75**(9), 1235–1245 (1987)
37. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. **C–36**(1), 24–35 (1987). https://doi.org/10.1109/TC.1987.5009446
38. Li, L., Apvrille, L., Genius, D.: Virtual prototyping of automotive systems: towards multi-level design space exploration. In: DASIP (2016)
39. Li, L.W., Genius, D., Apvrille, L.: Formal and virtual multi-level design space exploration. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) MODELSWARD 2017. CCIS, vol. 880, pp. 47–71. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94764-8_3
40. Niaki, S.H.A., Jakobsen, M.K., Sulonen, T., Sander, I.: Formal heterogeneous system modeling with SystemC. In: 2012 Forum on Specification and Design Languages (FDL), pp. 160–167. IEEE (2012)
41. Ninios, P., Vlahos, K., Bunn, D.W.: OO/DEVS: a platform for industry simulation and strategic modelling. Decis. Support Syst. **15**(3), 229–245 (1995)
42. Ptolemy.org (ed.): System Design, Modeling, and Simulation using Ptolemy II (2014)
43. Quillevere, H.: Gtk Analog Wave Viewer (2019). http://www.rvq.fr/linux/gaw.php
44. Selic, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems. Elsevier, Amsterdam (2013)
45. SocLib Consortium: The SoCLib project: an integrated system-on-chip modelling and simulation platform. Technical report, CNRS (2003). www.soclib.fr
46. Taha, S., Radermacher, A., Gérard, S.: An entirely model-based framework for hardware design and simulation. In: Hinchey, M., et al. (eds.) BICC/DIPES -2010. IAICT, vol. 329, pp. 31–42. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15234-4_5
47. 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
48. Vidal, J., de Lamotte, F., Gogniat, G., Soulard, P., Diguet, J.P.: A co-design approach for embedded system modeling and code generation with UML and MARTE. In: DATE, pp. 226–231. IEEE (2009)
49. VSI Alliance: Virtual Component Interface Standard (OCB 2 2.0), August 2000
50. Zeigler, B.P., Kim, D.: Distributed supply chain simulation in a DEVS/CORBA execution environment. In: WSC 1999, 1999 Winter Simulation Conference Proceedings. Simulation-A Bridge to the Future (Cat. No. 99CH37038), vol. 2, pp. 1333–1340. IEEE (1999)
51. Zhu, J., Sander, I., Jantsch, A.: HetMoC: heterogeneous modelling in SystemC. In: 2010 Forum on Specification & Design Languages (FDL 2010), pp. 1–6. IET (2010)