Hardware / Software / Analog System Partitioning with SysML and SystemC-AMS

Daniela Genius, Ludovic Apvrille

Hardware / Software / Analog System Partitioning with SysML and SystemC-AMS

Daniela Genius, LIP6, Paris Sorbonne Université, Paris, France, *daniela.genius@lip6.fr*
Ludovic Apvrille, LTCI, Télécom Paris, Institut Polytechnique de Paris, Sophia Antipolis, France
ludovic.apvrille@telecom-paris.fr

Abstract

Model-driven approaches for designing software and hardware parts of embedded systems are generally limited to their digital parts. On the other hand, virtual prototyping and co-simulation have emerged as a promising research topic, but target the modeling levels when partitioning has already been performed. This paper presents a model-driven platform for the partitioning of analog/mixed-signal systems.

keywords: virtual prototyping, embedded systems, analog/mixed signal, design space exploration

1. Introduction and Context

Embedded systems are frequently built upon heterogeneous hardware e.g. processors, FPGAs, DSPs, hardware accelerators, digital and analog analog/mixed signal (AMS) and radio frequency (RF) circuits. In early design phases, rapid but not-so-precise exploration of the design space is required in order to efficiently use the target platform. For this purpose, heterogeneous embedded systems require a high-level representation that includes very abstract models of their AMS and RF components. In previous contributions, we have shown how AMS components can be handled at a lower abstraction level i.e. after the system has been partitioned [19]. Thus, from a precise deployment diagram, we can generate a virtual prototype featuring both digital (hardware, software) and analog aspects. Yet, before partitioning, several challenges have to be tackled, in particular how to properly represent AMS components.

After the related work in Section 2, the paper introduces the fundamental concepts in Section 3. Our contribution is described in Section 4; we then apply it to a case study in Section 5 before we conclude.

2. Related Work

Our work spans the domains of virtual prototyping and co-simulation of analog/digital embedded systems.

The Architecture Analysis & Design Language [14] allows the use of formal methods for safety-critical real-time systems in avionics and automotive, among other domains; an approach that generates a system implementation from models using Simulink has been presented in [7].

Capella [26] relies on Arcadia, a comprehensive model-based engineering method. Widespread in the domains of defense, space and transportation within Thalès company, it provides architecture diagrams allocating functions to components, allocation of behavioral onto implementation components which are typically hardware. Capella also provides sequence diagrams, state machines and advanced mechanisms to model bit-precise data structures; it also supports co-simulation.

MDGen [31] starts from Rhapsody, which can automatically generate software, but not hardware descriptions from SysML, which in Rhapsody is untimed and sequential.

UML/SysML based modeling techniques such as MARTE and Gaspard2 [34, 16] are extremely popular for capturing the behavior of embedded systems, but less widely used for heterogeneous system design [29]. Furthermore, with very few exceptions such as [32, 23], they do not support refinement until cycle/bit accurate level virtual nor provide OS support for full-system simulation. Co-simulation between different Models of Computation is usually out of scope, too.

Modelica [15] 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 sym-

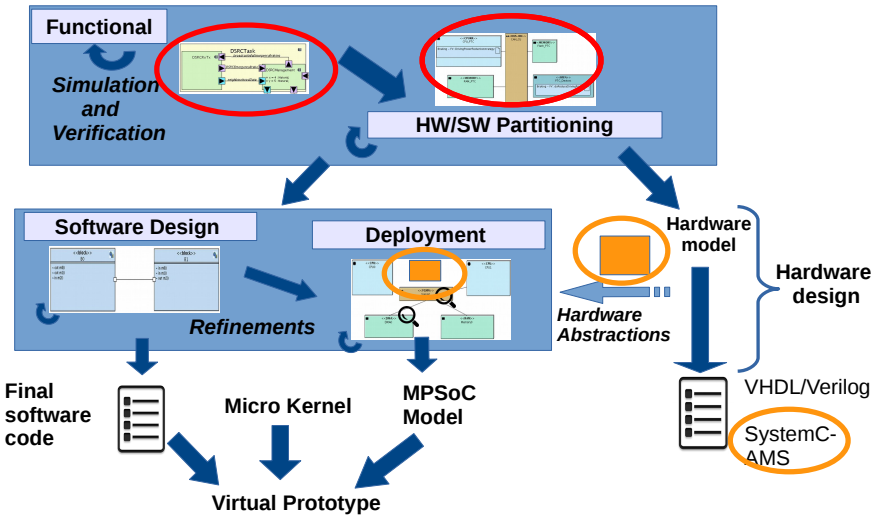

Figure 1. Hardware/Software partitioning and Code generation for MPSoC platforms

bolic way in order to determine an execution order between components of different MoCs.

Our approach links two simulation semantics. Linking simulation with different MoCs can be done by using e.g. the Functional Mockup Interface [10], which is linked to Modelica tools. Yet, in our case, we need to take into account both DE and AMS semantics because we want to compute a valid schedule before simulation, as explained in Section 5.2.

Ptolemy II [27], based upon a data-flow model, address digital/analog systems by defining several sub domains. Instantiation of elements controlling time synchronization between domains is however left to the designer.

Metropolis [4] uses high level models and facilitates the separation of concerns between computation and communication aspects. Heterogeneity can however only be represented using processes, mediums, quantities and constraints, hierarchical models are not allowed. Metro II [12] is based on high level and hierarchical models. So-called *Adapters* can be used for data synchronization between components belonging to different Models of Computation (MoCs), yet the model designer still has to implement time synchronization. As a common simulation kernel handles the entire process execution (digital and analog), MoCs are not well separated.

SystemC [21], a library of C++ classes, makes it possible to model digital hardware. SystemC-AMS extensions [6][33], about to become a standard, describes an extension of SystemC with AMS and RF features 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, commercialized by Coseda [13]. Another simulator is proposed in the H-Inception project[20]. However, no means for running larger software parts (OS, loader) are provided.

3. Basic Concepts

Let us briefly introduce the two fundamental concepts and associated tools which are the basis of the present work.

3.1. TTool

TTool [2] is a tool for model-based engineering of (digital) embedded systems at different abstraction levels grouped into two subsets: DIPLODOCUS (*functional*, *partitioning*) and AVATAR (*software design*, and *deployment*). To each abstraction level corresponds specific SysML views, see Figure 1 (adapted from [19]).

In the DIPLODOCUS methodology [3], software and hardware tasks to be partitioned are first captured within the functional abstraction level. Then, system-level mapping onto high level abstract hardware components is performed. After partitioning, software tasks can be detailed and then deployed on more concrete hardware models of the target system composed of elements of the destination platform (hardware components, operating system).

The AVATAR methodology [25] allows the user to design the software, perform functional simulation and formal verification, and finally test the software components in a virtual prototyping environment. A deploy-

ment diagram introduced in [17] is a SysML representation of hardware components, their interconnection, tasks and channels. The latter are based on the *SoCLib* [30] public domain library written in SystemC. Deployment diagrams themselves are not subject to formal verification. In deployment, partitioning decisions can be re-validated or invalidated [23].

In both partitioning and deployment, the computational 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.

Verification relies either on formal verification using timed automata semantics (using TTool model checker or UPPAAL, see [8], or simulations. Simulation at high abstraction level (i.e. DIPLODOCUS) is fast but imprecise. On the contrary, simulations performed from (AVATAR) deployment models are cycle/bit accurate but slow. Last but not least, executable C code can be generated from AVATAR software components.

While Analog/Mixed Signal components can be described along with software tasks in a quite detailed manner in order to generate the virtual prototype [19], as will be shown in the next section, they are not yet represented nor simulated at partitioning level until now.

In Figure 1, orange circles point out AMS extensions to hardware parts and deployment. On the upper part of the figure however, depicting the partitioning level, distinction is only made between functional blocks mapped to hardware or software. No particular attention is given to *analog* blocks which, at best, can be imperfectly modeled as hardware accelerators. The present paper addresses these aspects, which are encicled in red.

3.2. SystemC AMS

Besides Discrete Event (DE) models for digital blocks, SystemC AMS relies on the Timed Data Flow (TDF) Model of Computation, itself is based on the timeless Synchronous Data Flow (SDF) semantics [22].

Timed Data Flow A TDF module is described with an attribute representing the time step and a processing function. A *processing function* is a mathematical function depending on the module inputs and/or internal states. At each time step, a TDF module first reads a fixed number of samples from each of its input ports, then executes its processing function, and finally writes a fixed number of samples to each of its output ports. TDF modules can interact with the DE world (such as digital MPSoC platforms) using converter ports.

Figure 2 shows a TDF cluster in the representation defined in the SystemC AMS standard [6]. 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, DE ports as white squares and TDF signals as arrows. So-called **converter ports**, shown as black-and white squares, serve as interface between the TDF and DE MoC. The TDF modules have the following attributes:

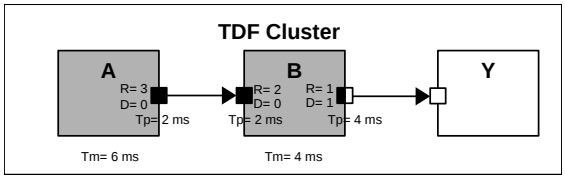

Figure 2. TDF Cluster

- Module Timestep (**Tm**) denotes the period during which the module will be activated. One module will only be activated if there are enough samples available at its input ports.
- Rate (**R**). Each module will read or write 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.
- Port Timestep (**Tp**) is the period during which each port of a module will be 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.

Due to their different Models of Computation, AMS components require to be simulated apart from the rest of the system. Yet, they regularly have to synchronize with the digital platform. The SystemC kernel thus **controls** the AMS kernel which runs continuously until interrupted. When a SystemC AMS simulation is being executed, the execution of the SystemC simulation kernel is blocked, while the SystemC AMS simulation kernel continues running. As a consequence, during this period the DE simulation time does not advance at all, while the TDF simulation time advances. A crucial problem is thus building a co-simulation environment which synchronizes DE and TDF without causing causality issues. A recent work [1] shows how to solve such issues for SystemC AMS code before simulation. Yet, TTool goes even further by detecting them **before the code generation** of a SystemC AMS system [19].

Electrical Linear Networks TTool also offers the possibility to generate code for Electrical Linear Networks (ELN) encapsulated within TDF clusters. ELN clusters can be captured with specific diagrams in which a limited set of components can be used. They are not yet tested with a software part running in the MP-SoC.

4. Contribution

As said before, analog components can already be captured in deployment diagrams (software design level, a.k.a. AVATAR) [19]. The present paper proposes an extension to the partitioning level (DIPLODOCUS), allowing to capture analog/mixed signal modules and to generate abstract simulation code in C/C++, just as it is already done for discrete components.

4.1. Modeling and verification approach

The HW/SW candidate architecture of the system is modeled in form of a graph made of execution, communication, and storage nodes. Execution nodes are for example CPUs and hardware accelerators. Our contributions adds a representation of analog/mixed signal modules, which are execution nodes too. Communication nodes include bridges and buses, storage nodes are memories.

TDF clusters are jointly modeled with discrete components using hierarchical SysML block diagrams. Blocks are connected through ports featuring channels for exchanging data (blocking and non-blocking semantics possible), events and requests for control information. In terms of behavior, our idea is to capture the behavior each TDF cluster with UML activity diagrams, since the latter are already used for describing the behavior of discrete components [3]. Behavioral diagrams can contain nondeterministic choices, finite and infinite loops, and general control operators. So-called *Delays* can also be used to express the physical duration of a computation, not to be confounded with TDF Delays.

Diagrams are first converted in C++ before being simulated. Our simulation engine is predictive; each processing elements goes on at its own simulation pace until a system event (a data transfer, a synchronization event, etc.) makes current transactions invalid. Then, the latter are cut back as much as necessary in the past, and the simulation continues from the cut transactions.

4.2. Modeling TDF clusters

In the following, we describe our abstract model TDF clusters on partitioning level.

- To capture the semantics of TDF, only channels are used in the activity diagrams, neither events nor requests. Each TDF port is associated to one data channel in activity diagram.
- Branches stemming from choices (that later become if statements in the *processing* function) can be directly translated into branch control structures in the activity diagram.
- The TDF *cluster timestep* corresponds to the *delay* in the activity diagram. There can only be one timestep per cluster. It is either estimated or derived from the scheduling an existing TDF model.
- Activity diagrams allow to specify a *number of data samples* written to/read from a channel: we thus model the TDF *rate*.
- The behavior of each cluster is captured within a *loop forever* in the activity diagram.

All features individual to a module within a cluster, such as port timesteps and delays, are not explicitly modeled, so they are left to the detailed TDF model on the software design level.

5. Case study

The following case study illustrates both the partitioning level (the main contribution of this paper), and the software design level (presented in [19]), as well as their interaction. Our simplified rover system, intended to assist rescuers in finding, victims buried in rubbles. has four main components: central control, motor control and two sensors: ultrasonic distance control and a temperature sensor.

5.1. Partitioning

Figure 3 shows the purely functional model of the rover. Green boxes represent functional blocks connected through ports via data channels (in blue) and synchronization events (in purple). The two analog sensors of the rover can now be captured as specific tasks. As for discrete tasks, their behavior is captured within *activity diagrams*. Figure 4 shows on its left the activity diagram of the distance sensor, in the center the activity diagram of the temperature sensor , and on the right the activity diagram of the motor control. Figure 5 shows the activity diagram of the main control.

The architecture and mapping diagram, shown in Figure 6, features a specific hardware model named *CAMS* (abbreviating SystemC-AMS). The Figure thus shows on its upper right the two AMS components, on which the *TemperatureSensor* and the *DistanceSensor*

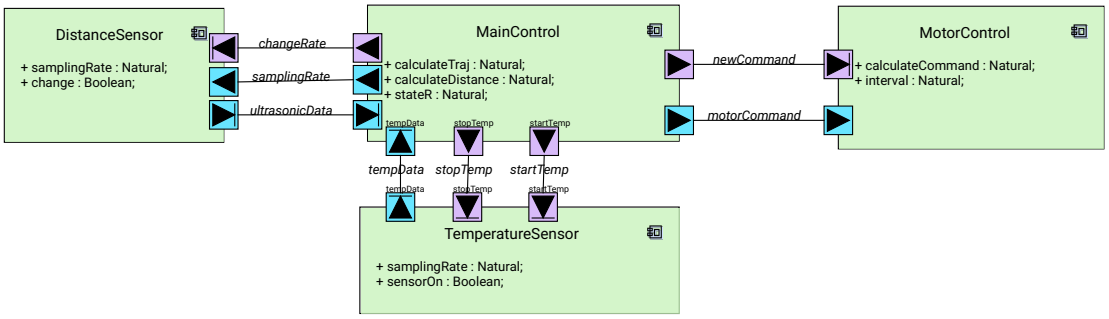

Figure 3. Functional view of the rover

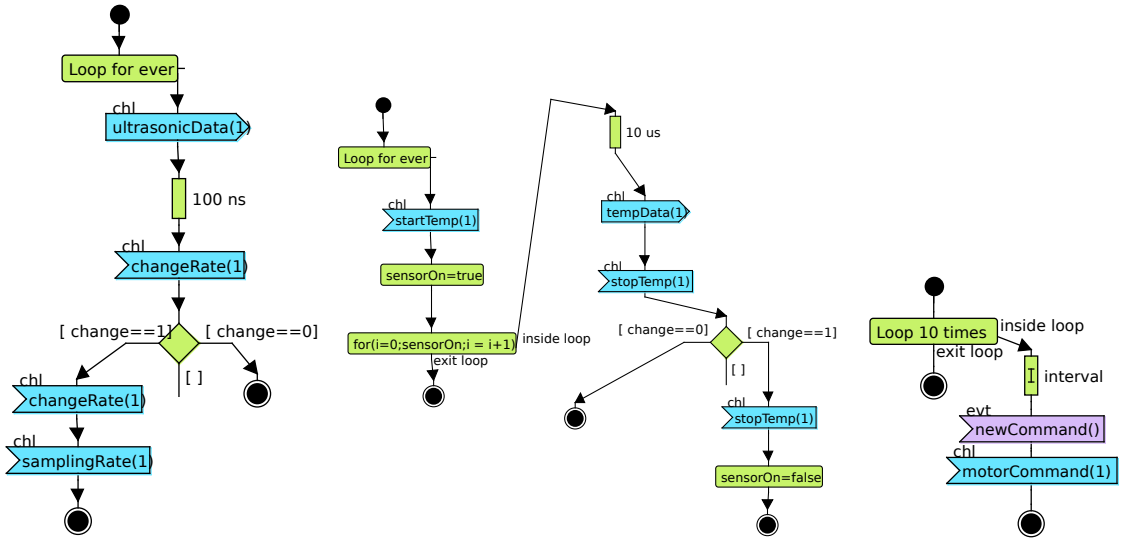

Figure 4. Activity diagrams: distance sensor (left) temperature sensor (center) motor control (right)

functionalities are mapped, as it would have been done with hardware accelerators.

5.2. Software design and deployment

Obviously, SysML block diagrams for capturing software components do not capture AMS components: they are therefore modeled in separate SystemC-AMS panels. Also, communication channels between software components and AMS components cannot be connected at this level of abstraction. Indeed, in [19] we showed that these communication are handled differently, as explained in the next paragraph.

The two software blocks communicate with each other through a signal *motorCommand* which is sent by the *MainControl* to the *MotorControl* and contains two parameters: the right and the left velocity. The behaviour of software blocks is given by state machines. The state machine of the *MainControl* block is shown on the left hand side of Figure 8.

Interaction and co-simulation The two software blocks interact with the two analog sensors through so-called *GPIO2VCI* components. The latter are SystemC AMS components with DE ports on one side, while they function as Virtual Component Interconnect [35] targets in the SoCLib MPSoC on the other. The part of the control running in the digital part activates and deactivates the sensors or initiates rate changes for the ultrasonic sensor, whereas the sensors write measurement data to the digital platform at regular intervals. A library named *libsyscams* has been introduced to define read and write primitives on the side of the MPSoC, namely the *read_gpio2vci* and *write_gpio2vci* functions [11]. By executing software functions, the CPU of the digital platform is thus able to write or read values from the analog components. The right hand side of Figure 8 shows the *MainControl* block modified to communicate

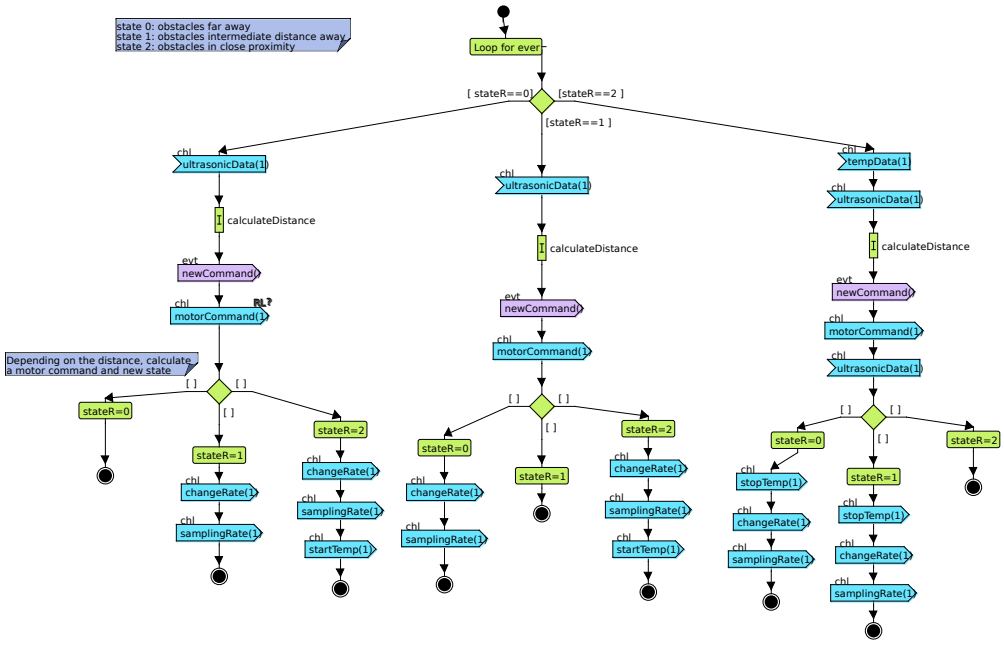

Figure 5. Activity diagram of the main control

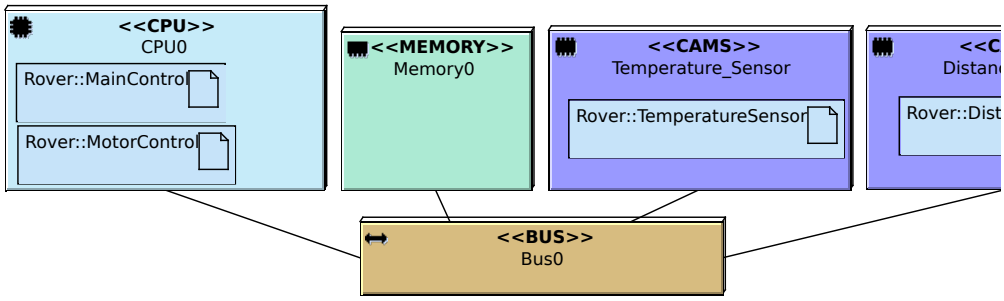

Figure 6. Partitioning level architecture and mapping diagram

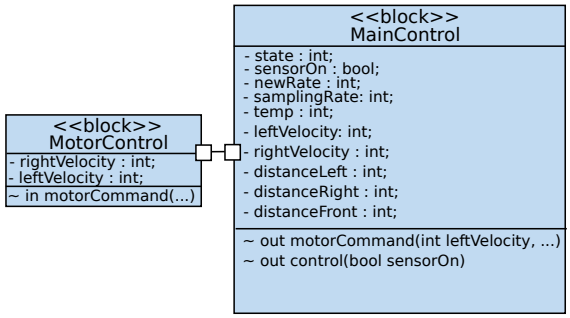

Figure 7. Software block diagram

with the analog sensors. C code inserted as so-called *entry code* into the *startController* state, turns the temperature sensor unit module on and off, initiates changes of the sampling rate and reads the distance sensor.

Analog components Analog components can be parameterized with SystemC AMS parameters (rate, delay, timestep). In contrast to the version without rate change presented in [11], Dynamic TDF [5], as implemented in SystemC-AMS PoC 2.1, allows to implement the change of the sampling rate necessary for the original rover model. The TDF model of the distance sensor was thus rewritten w.r.t. the model shown in [19].

Figure 11 shows TTool's graphical representation of a TDF cluster modeling the ultrasonic distance sensor. As we only have one GPIO2VCI interface, we code by integers the sensor in question (left, front, right), and the rate change. There is one additional DE module, controlling the access to the different ports of the TDF module. Figure 12 shows a graphical representation of the temperature sensor, modeled as a cluster containing a single TDF block. From these graphical

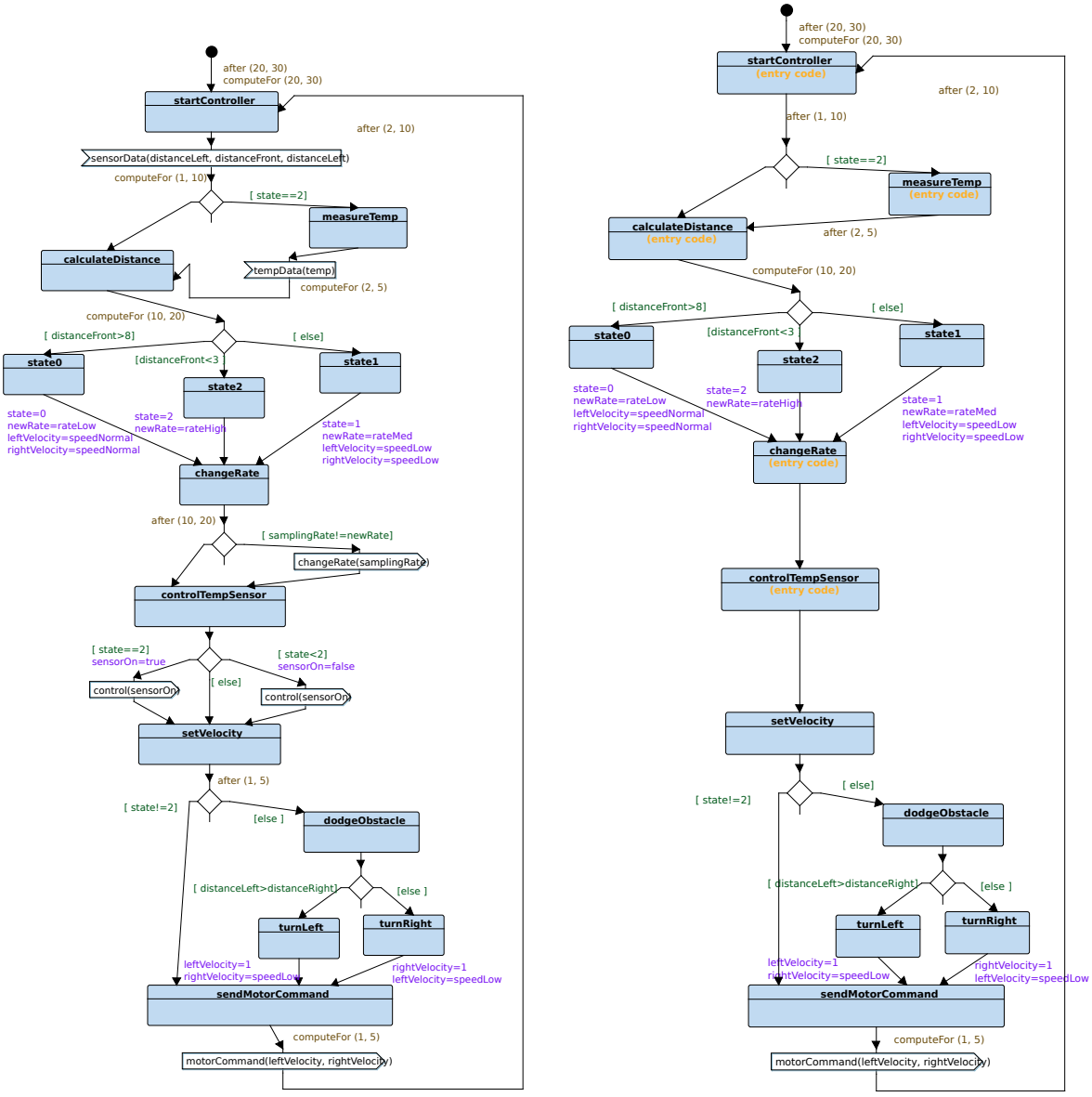

Figure 8. Main control state machines: original(left) and AMS (right) with entry code in the *startController*, *readDistanceSensor*, *setTempSensor* and *measureTemp* states

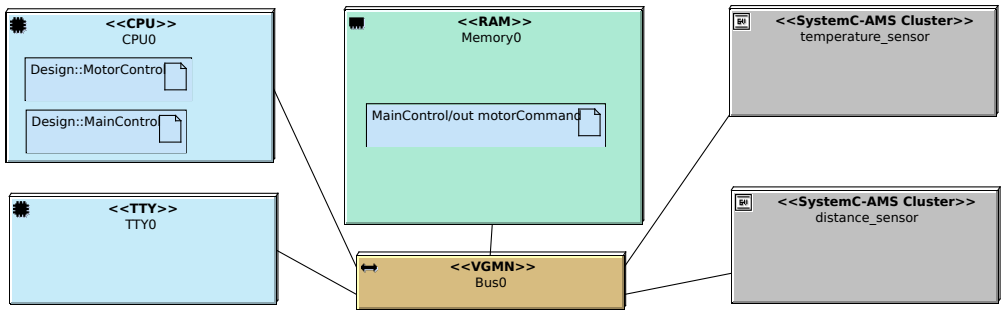

Figure 9. Extended deployment diagram

information, TTool first computes a coherent schedule, and then generates SystemC-AMS code, including the ports, delays and interfaces [11]. These steps are done in a so-called *validation* window (Figure 10). Once the cluster schedule is validated, one can then initiate simulation code generation (start button). In case TTool de-

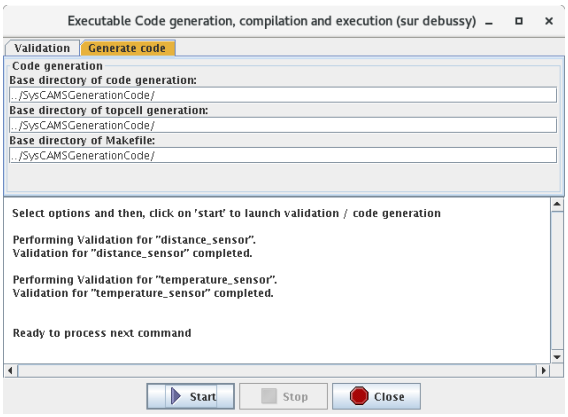

Figure 10. Validation/code generation window

tects that additional delays are necessary, the designer shall also modify the high level activity diagrams (i.e. in DIPLODOCUS) in order to insert them manually.

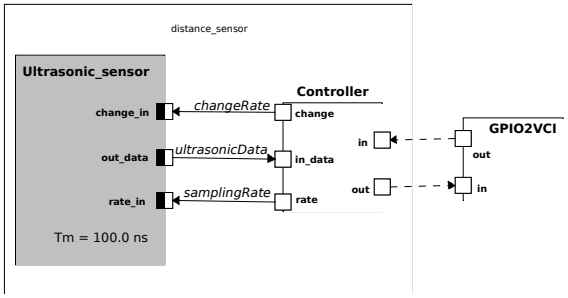

Figure 11. SystemC-AMS representation of the ultrasonic distance sensor in TTool

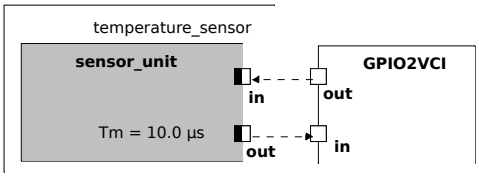

Figure 12. SystemC-AMS representation of the temperature sensor in TTool

Extended deployment diagram Figure 9 shows the extended deployment diagram containing the analog and the digital parts. The two software tasks (*MainControl* and *MotorControl*) are mapped onto the CPU, and the channel between the tasks — which does not appear explicitly at mapping level — is mapped onto the memory. TDF clusters appear as gray boxes along with digital hardware components; they are interconnected to the central (digital) interconnect — bus or network on chip — through the GPIO interface components.

The extended *deployment diagram* (Figure 9) gives an overview of the mapping of software tasks and channels. The former are mapped to CPUs, the latter to on-chip memory. For a better overview, the diagram contains sensors as gray boxes, each corresponding to a SystemC-AMS cluster connected via a GPIO. Clicking on the box opens the corresponding SystemC-AMS panel.

5.3. Investigating latencies

Interesting latencies to study are typically the reaction delay of the rover to an obstacle, i.e. the latency to adapt its speed and modify its sampling rate. For instance, we measure the latency between the reception of an ultrasonic signal and the reaction by motor control on the virtual prototype generated from the deployment view, important for braking in reaction to an obstacle.

In order to have a better estimate of the delay parameter, we use the validation mechanism of the SystemC AMS modules on the lower level, which calculates valid schedule and by doing so yields a cluster period (for the temperature sensor and for the distance sensor). We modified the delay parameters for the two sensor clusters accordingly. Table 1 compares to the software-only models from [19] and shows that the latter were unrealistic.

5.4. Trace generation

While it was possible to generate cycle accurate *vcd* traces of the digital signals in TTool before adding the AMS extensions, the integration of SystemC-AMS necessitates the tracing of the analog, thus continuous, signals, on the virtual prototype. This functionality, 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 virtual prototype and of the SoCLib topcell. Traces can then be displayed with a tool adapted to analog traces, like GAW - Gtk Analog Wave viewer [28]. As usual, traces of the SystemC digital part can displayed with e.g. gtkwave. In Figure 13, the incoming signals from the two sensors are traced with GAW. The upper curve shows the output from the distance sensor, the lower the

Table 1. Average latencies compared to those obtained in [18]

Signal	AMS sensor models	Software sensor models
send(tempData)-> receive(tempData)	16.2	4
send(ultrasonicData)-> receive(ultrasonicData)	34.1	34
send(motorCommand)->receive(motorCommand)	19.3	8
send(ultrasonicData)-> receive(motorCommand)	4.9	2
receive(ultrasonicData)-> send(changeRate)	13.4	23

output from the temperature sensor.

6. Discussion and Future Work

We show how we can take into account digital and analog aspects of an embedded system from the first modeling phases onwards: for that purpose, we have extended TTool's partitioning models with SystemC-AMS components. Thus, we use the already existing simulation methods at DIPLODOCUS level, relying on the generation of C++ code. For this, we describe the behavior of a TDF cluster more abstractly in terms of activity diagrams, and adapt the method with which other hardware coprocessors are simulated.

We use cluster timestep obtained on the software design and deployment level to obtain a more accurate estimation of the execution time for an operation. Yet, automated feedback to the new models is not yet automated. It would also be very interesting to try to establish a "translation" between TDF description and activity diagrams and vice versa.

Currently, the communication between digital and analog part is handled by only one GPIO2VCI channel in either sense per TDF cluster, which obliges to send control and samples one by one [11].

Our toolchain relies entirely on free software. Many others, also cycle-accurate, use commercial SysML editors or simulation tools [32, 24, 31]. TTool itself is already used in industry, for instance by Nokia (in the domain of telecommunication, e.g. 5G systems), with whom long standing collaborations exist.

The added features, allowing to model analog parts —in particular sensors— more realistically, open up new perspectives for application in the automotive and robotics domains. In this, TTool does not compete against commercial tools, but can complement them by its very fast high-level modeling facilities, while, if necessary, providing very detailed low-level simulation data. We are also currently working on larger industrial case studies in the scope of the AQUAS H2020 project. Yet, cycle-bit accurate simulations are slow, sometimes too slow for complex systems. TTool can already generate standalone SystemC-AMS code which can be useful when no MPSoC platform is required for running software, and will be extended to generate Transaction Level SoCLib prototypes.

References

[1] L. Andrade, T. Maehne, A. Vachoux, C. Ben Aoun, F. Pêcheux, and M.-M. Louërat. 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.

[2] L. Apvrille. *Webpage of TTool, https://ttool.telecom-paris.fr/*, 2003.

[3] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A uml-based environment for system design space exploration. In *2006 13th IEEE International Conference on Electronics, Circuits and Systems*, pages 1272–1275. IEEE, 2006.

[4] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. L. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[5] M. Barnasconi, K. Einwich, C. Grimm, T. Maehne, and A. Vachoux. Advancing the systemc analog/mixed-signal (ams) extensions. *Open SystemC Initiative*, 2011.

[6] M. Barnasconi, K. Einwich, C. Grimm, T. Maehne, and A. Vachoux. *SystemC AMS Extensions 2.0 Language Reference Manual*. Accellera systems initiative, January 2016.

[7] M. Ben Youssef, J.-F. Boland, G. Nicolescu, G. Bois, and J. Hugues. Bridging the high-level model to execution platform for design space exploration and implementation. In *ERTSS*, 2014.

[8] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*, pages 87–124. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[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] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, et al. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Techni-*

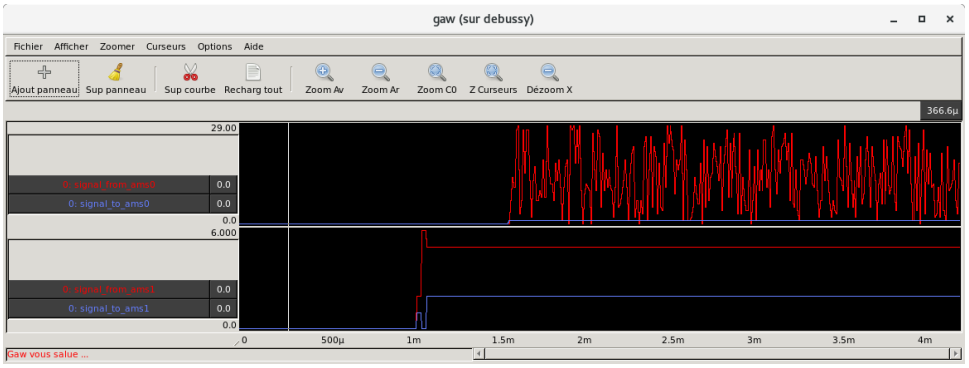

Figure 13. Analog trace generated from TTool's simulation

cal Univeristy; Dresden; Germany, number 063, pages 105–114. Linköping University Electronic Press, 2011.

[11] R. Cortés Porto. Integration of SystemC-AMS simulation platforms into TTool. Master's thesis, Technische Universität Kaiserslautern, 2018.

[12] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. A next-generation design framework for platform-based design. In *DVCon*, volume 152, 2007.

[13] K. Einwich. *SystemC AMS PoC2.1 Library, COSEDA, Dresden*, 2016.

[14] P. H. Feiler, B. A. Lewis, S. Vestal, and E. Colbert. An overview of the SAE architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In P. Dissaux, M. Filali-Amine, P. Michel, and F. Vernadat, editors, *IFIP-WADL*, volume 176 of *IFIP*, pages 3–15. Springer, 2004.

[15] P. Fritzson and V. Engelson. Modelica—a unified object-oriented language for system modeling and simulation. In *European Conference on Object-Oriented Programming*, pages 67–90. Springer, 1998.

[16] A. Gamatié, S. L. Beux, É. Piel, R. B. Atitallah, A. Etien, P. Marquet, and J.-L. Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embedded Comput. Syst*, 10(4):39, 2011.

[17] D. Genius and L. Apvrille. Virtual yet precise prototyping: An automotive case study. In *ERTSS'2016*, Toulouse, Jan. 2016.

[18] D. Genius, L. W. Li, and L. Apvrille. Multi-level Latency Evaluation with an MDE Approach. In *MODELSWARD*, Jan. 2018.

[19] Genius, Daniela, Cortés Porto, Rodrigo, Apvrille, Ludovic, and Pêcheux, François. A tool for high-level modeling of analog/mixed signal embedded systems. In *MODELSWARD*, 2019.

[20] H-Inception Consortium. *Heterogeneous Inception Project*, 2012-2015. https://www-soc.lip6.fr/trac/hinception.

[21] IEEE. *SystemC*. IEEE Standard 1666-2011, 2011.

[22] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[23] L. W. Li, D. Genius, and L. Apvrille. Formal and virtual multi-level design space exploration. In *MODELSWARD, Springer CCIS vol 880*, pages 47–71, 2018.

[24] W. Mueller, D. He, F. Mischkalla, A. Wegele, A. Larkham, P. Whiston, P. Peñil, E. Villar, N. Mitas, D. Kritharidis, et al. The SATURN approach to sysml-based hw/sw codesign. In *VLSI 2010 Annual Symposium*, pages 151–164, Lixouri, Greece, 2011. Springer.

[25] G. Pedroza, D. Knorreck, and L. Apvrille. AVATAR: A SysML environment for the formal verification of safety and security properties. In *The 11th IEEE Conference on Distributed Systems and New Technologies (NOTERE'2011)*, Paris, France, May 2011.

[26] Polarsys. ARCADIA/CAPELLA (webpage). In *https://www.polarsys.org/capella/arcadia.html*, 2008.

[27] Ptolemy.org, editor. *System Design, Modeling, and Simulation using Ptolemy II*. 2014.

[28] H. Quillevere. *Gtk Analog Wave viewer*, 2019.

[29] B. Selic and S. Gérard. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Elsevier, 2013.

[30] SocLib consortium. *The SoCLib project: An Integrated System-on-Chip Modelling and Simulation Platform, www.soclib.fr*, 2003.

[31] Sodius Corporation. Mdgen for SystemC. http://sodius.com/products-overview/systemc.

[32] S. Taha, A. Radermacher, and S. Gérard. An entirely model-based framework for hardware design and simulation. In *DIPES/BICC*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 31–42. Springer, 2010.

[33] A. Vachoux, C. Grimm, and K. Einwich. Analog and mixed signal modelling with SystemC-AMS. In *ISCAS (3)*, pages 914–917. IEEE, 2003.

[34] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet. A co-design approach for embedded system modeling and code generation with UML and MARTE. In *DATE*, pages 226–231. IEEE, 2009.

[35] VSI Alliance. *Virtual Component Interface Standard (OCB 2 2.0)*, Aug. 2000.