2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Heterogenous Virtual Prototyping for IoT Applications

Mark Burton, GreenSoCs, Bordeaux, France *(mark.burton@greensocs.com*)

Luc Michel, Antfield, Grenoble, France (*luc.michel@antfield.fr*)

Paul Ehrlich, Karsten Einwich, COSEDA Technologies, Dresden, Germany
(*paul.ehrlich|karsten.einwich@coseda-tech.com*)

***Abstract*—New technologies like IoT demand extended virtual prototyping technologies and a seamless integration of different worlds. This paper describes how different techniques can be combined to create virtual prototypes which represent the whole analog, digital hard/soft-ware system and the physical environment.**

Keywords—IoT, Virtual Prototyping SystemC, SystemC-AMS, Qemu, Continous Integration, Multi-domain modelling

I. INTRODUCTION

From the blinkered perspective of simulation technologies, to our dismay, the world seems to be increasingly heterogeneous. IoT (Internet of Things) devices combine state of the art computational platforms with analog, electrical and non-electrical sensor technologies. Furthermore such devices interact with each other to control analog physical quantities like temperature, humidity or lighting with increasingly complex control algorithms. So it becomes essential to understand the interaction between the different components, both digital and analogue, increasingly the complexity is in the interaction, rather than the individual devices. As the entity of interest moves inexorably from the device to the network of interconnected devices, hardware prototyping of IoT devices and networks will be very costly if not impossible. Additionally, of course, devices in an IoT network will be provided and developed by different vendors. The classical “design by (paper) Spec” approach will be stressed to breaking point, leading to time consuming and costly re-spins, unsatisfied customers or very conservative designed systems with high costs and non-competitive performance.

The solution is (of course) virtual prototyping. But not at the level of individual devices, rather at the level of the collection of devices, in a heterogeneous system, spanning both the digital, analogue and ‘real’ worlds. Virtual prototyping allows engineers to play with architectures and the devices without the availability of any hardware components and, crucially, without the need for specific environmental conditions! Furthermore virtual prototypes can be used to check the compatibility of different party’s devices with the remaining IoT network and they permit a much better introspection and debugging. So application and especially failure scenarios can be verified which cannot easily be checked in reality (or only at high costs). This gains importance for safety and security critical systems as numerous scenarios and security attacks can easily be verified. This combines well with modern verification technologies for virtual prototypes, which allow continuous integration techniques to be applied. So, after each change to the hardware or software components all verification scenarios can be automatically executed immediately identifying issues. This increases the design efficiency significantly and makes the design process much more predictable due the reduced risk of the integration phases.

Virtual prototypes are of course the solution, but, on the whole, today’s virtual prototypes tends to be separated into a pure digital hardware/software world or a mixed-signal world. Combining both has, hitherto, been somewhat tricky. SystemC has been adopted as the unifying technology for virtual prototypes, it aims to fulfill requirements such as high simulation performance, model exchangeability and integration into a continuous integration flow. With the recent introduction of SystemC-AMS, all the required base technologies for creating heterogeneous mixed-signal prototypes are now available. For digital system modelling the IEEE1666 standard has been used for more than a decade, and includes modelling technologies like discrete event and TLM (transaction level modelling) as well as libraries for e.g. verification and parameter handling. The SystemC AMS IEEE1666.1 standard adds on

top of SystemC feature for modelling analog continuous time behavior. So it adds timed dataflow, signal flow and conservative system modelling – all technologies to enable abstract modelling and thus the creation of extremely fast running models. However, although both parts of the SystemC standard live under the same roof, the ability to truly build heterogeneous simulations has not really been put to the test. In each domain, impressive developments have been reported, with state-of-the-art virtualization technologies provided by commercial vendors or the open source community (e.g. with the Qemu platform), and likewise impressive analogue simulation performance being demonstrated in mixed signal environments. The missing link is to show them working successfully together.

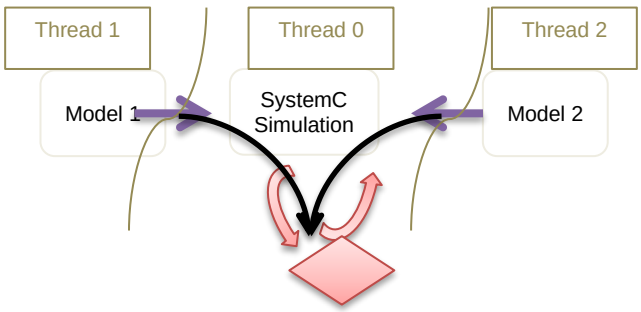
This paper will present exactly this combination of TLM virtualization, booting linux in near real-time, while including SystemC AMS components that interact with models of the real-world and mixed signal circuitry. Combining powerful virtual prototypes for the hardware and software design of IoT devices and their integration into an IoT application. Especial focus is given to the interfaces and integration of the different technologies. The integration of a Qemu based platform via TLM and the use of loosely time and quantum based techniques will be discussed. Different solutions for the modelling of the digital analog interface based on TLM and signal based interfaces are discussed. Technologies for modelling different physical domains based on SystemC AMS will be presented. To illustrate the presented techniques we will conclude with an example of a multi-physical system controlled by an ARM processor based host controller.

II. QEMU SYSTEMC INTEGRATION

Using Qemu as a CPU simulation kernel has a number of advantages. It not only covers a wide variety of architecture (including ARM, PPC, x86, Mips, etc), but it is also efficient, and can be flexible in terms of what is modeled. One of the advantages Qemu brings is the ability to run a number of CPU cores in separate threads. This yields extremely high performance simulations, but care must be taken when integrating Qemu with SystemC specifically synchronization must be managed.

A number of attempts have been made to integrate Qemu and Tlm/SystemC (e.g. see [9]). Several approaches have been made to synchronize the two. The issue is that different solutions seem to be favorable in different circumstances. The approach we take is somewhat more flexible and consists of some distinct parts.

The SystemC kernel provides a mechanism to receive events from an asynchronous thread. We make use of this mechanism to signal that Qemu is ready to communicate to the SystemC model. However, we must first deal with a problem within SystemC's itself. When the SystemC kernel runs out of events, it exits. In this case, it may be that an asynchronous event would have arrived later. In other words, we need a mechanism to prevent SystemC exiting if there are (potentially) pending asynchronous events.



We can arrange this in SystemC, but to allow different model sources, patches have been proposed to the SystemC kernel and will be in SystemC 2.3.1 which will provide a suitable central mechanism. A single central semaphore enables any number of models to be combined.

In our implementation, we can use either our own semaphore, or the central one provided by the kernel; performance is indistinguishable.

The second issue we deal with is the execution of SystemC models within the SystemC thread. This is critical for SystemC models, as they rely on light-weight processing, and that's highly susceptible to which thread executes.

Therefore, we must ensure that SystemC code is (in general) executed by the same SystemC thread. We provide a bridging mechanism – which uses the asynchronous event mechanism above.

The third part of the integration involves synchronization itself. Several approaches to this can be taken. We have investigated two. The first is simply no synchronization at all. In this case we allow SystemC and Qemu to run (and advance their notion of time) as quickly, and as much as they can, or want to. This approach is not always appropriate, and is clearly not appropriate for many mixed analogue models. However it can be appropriate even when (for instance) hardware is in the loop. It may be that Qemu is being used to drive the Hardware, and the 'local' Qemu time is not actually relevant to the hardware under test. The second approach is a windowing approach. This attempts to keep both the SystemC and Qemu sides as closely synchronized as possible, by adjusting time (or, if that proves impossible) slowing down one side or the other. This is a development of the 'windowing' approach proposed by [10]. However, what we have discovered is that in general, we can not make any guarantees about the synchronization with respect to the quantum. (A model may always call 'wait', and we have no control how far time will move. Any attempt to control this will trivially end in a deadlock of course.) None the less, using this approach, we have successfully modeled the mixed digital/analogue system presented in this paper.

III. DIGITAL ANALOG INTERFACE MODELLING

The SystemC AMS standard defines a synchronization between SystemC signals and the different SystemC AMS modelling domains. The SystemC AMS synchronization is based on a sampling approach that means at each AMS calculation time the current value of the discrete event signal is sampled or assigned respectively.

On top of this SystemC AMS standard interaction other schemes can be realized. As an example of such a synchronization we will describe the interfacing between a TLM based interactions with an analog sub-module. An example for such interaction is a peripheral like an adc (analog digital converter) or a dac (digital analog converter) which is connected to a bus system. Thereby the TLM transaction can be loosely timed and using timing annotation. The adc and dac as well as analog modules like filter are usually modelled in the timed dataflow (TDF) SystemC AMS domain.

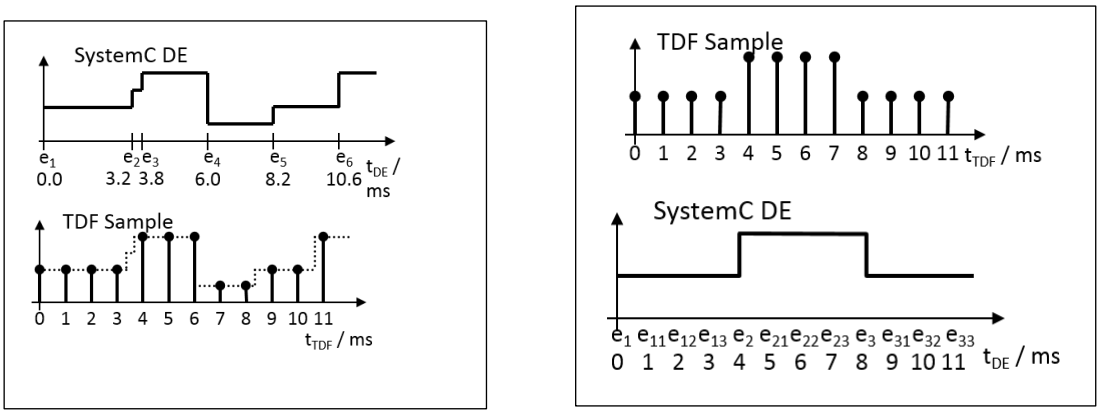


Figure 1: SystemC - SystemC-AMS interaction

* Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

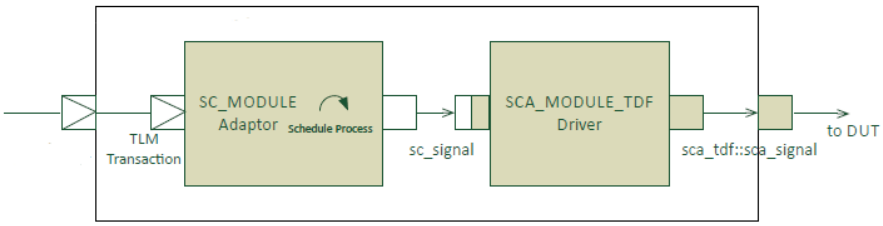


Figure 2 Principle TLM - analog interaction

Therefore the "loosely time" of the transaction must be synchronized to the "strict" time of an AMS simulation. A simple realization for this synchronization is shown in Figure 2. In this case a SystemC target module receives the transaction and writes the content to a SystemC signal at the corresponding time points. The SystemC sc_signal is connected to the analog TDF module via a converter port and thus uses the standard SystemC-SystemC AMS synchronization. This form of synchronization is well suited for the case that the time distance between transactions to an analog module is lesser than the analog time step. For the case the time step between transactions is in the same order and this communication may will become performance limiting, a more complicated mechanism can be used, where the loosely timed transactions are buffered and proceeded without an explicit synchronization by the TDF module.

IV. MODELLING PHYSICAL DOMAINS WITH SYSTEMC-AMS

The SystemC AMS standard defines an electrical linear network (ELN) modelling domain. ELN is a so called conservative modelling approach relying on the Kirchoff's law (the sum of all currents flowing into a node is zero, the sum of the voltages across elements in a loop is also zero). This principle can also applied for other physical domains. Generalized we speak about across values (corresponds to the voltage in the electrical domain) and through values (corresponds to the current). The defined ELN elements of the SystemC AMS standard allow to describe any linear equation system. So via so called analogy relation [8] any physical system, which can be described by linear DAE (differential algebraic equations) can be described with the available ELN elements.

A. Principle analogy relation

To illustrate the principle of analogy relation we will use the modelling of a mechanical system using electrical elements. Principally two kinds of analogies can be used, the force-voltage and the force-current analogy. The second is more practical for our application, due how we will see later, the wiring will not change. So we define that the force f becomes the through value and thus corresponds to the current i. additionally we define that the velocity vel is the across value and thus corresponds to the voltage v. Demonstrative this fits, due if we consider one node, than the sum of all forces must be zero and if we add the velocities in an arbitrary loop the sum must be also zero. This is equivalent to the electrical currents and voltages. To illustrate this we will use the spring-mass system of Figure 3.

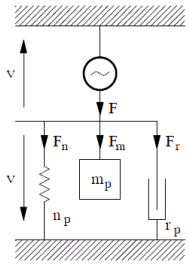


Figure 3 Mechanical mass-spring-damper system

For the forces F we can setup the following equation: $F_m + F_r + F_n = F$

Whereby F_m is the force resulting from the mass $m_p$and thus (acceleration of gravity not considered):

$$F_m = m * \dot{vel}$$

F_n is the force resulting from the spring: $F_n = k * s = k * \int vel * dt$

whereby s is the distance.

$F_r$is the force resulting from the damper and calculated by: $F_r = m * vel$

If we have a closer look to the equations, it can be easily recognized, that the equation of the mass corresponds to the equation of a capacitor: $i = C * \dot{v}$

the spring corresponds to an inductor: $i = \frac{1}{L} * \int v * dt$

and the damper corresponds to a resistor: $i = \frac{1}{R} * v$

and finally the force F trivially corresponds to a current source. The sum of the forces corresponds to the sum of the currents due Kirchoff's current law. Using this analogies, we can redraw the mechanical system of Figure 3 to the electrical system of Figure 4

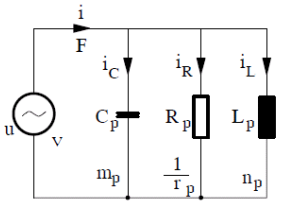


Figure 4 Electrical analogy of mass-spring-damper system

The nice thing of this analogy relation is, that each mechanical element corresponds to an electrical element. This enables us to create a library of mechanical elements based on a library of electrical elements.

Such analogy relations can be found for any other physical domain.

For example, in a thermal domain the across value corresponds to the temperature and the through value to the energy/heat flow. An isolation (like a wall) corresponds to a resistor and a heat storage like the mass of the wall or the floor corresponds to a capacitor. Similar analogy relation can be setup for domains like magnetic and hydraulic. The interaction between different domains like in a motor relates usually to an ideal transformer.
Based on this consideration any physical domain can be modelled using electrical elements.
Doing so, will lead to models which can be understood by experts only. The first attempt to solve this issue is to create model libraries for each physical domains, so that modules for a mass, friction or a heat storage are available.

B. Physical domain libraries based on SystemC-AMS

As described before, choosing the corresponding analogy relation, most physical components can be directly mapped to electrical components. Thus in the first attempted a physical domain library can be simply created by using the object oriented nature of SystemC-AMS – the mechanical elements can be simply derived from the corresponding electrical elements.

```
#include <systemc-ams>

class spring : public sca_eln::sca_l
{
public:
        spring(const char* nm,double k) : sca_eln::sca_l(nm,1.0/k)
        {}
};
```

Figure 5 Principal implementation of a spring

Doing so, the nodes will be still electrical nodes. This has disadvantages like, that nodes of different domains or using different analogy relations can be connected and while reading the model or tracing signals the different physical quantities cannot be distinguished.
To solve this issues, the same principle can be applied to the nodes (corresponding in SystemC to signals) and terminals (corresponding in SystemC to ports). Thus we can derive domain specific nodes and terminals from the standard SystemC-AMS electrical nodes and terminals.
Using C++ feature, this principle can be generalized in a way, that the nodes and terminal definition becomes a template with the domain information as a template parameter. This leads to a syntax, which fits very well to the SystemC sc_signal syntax, where the template argument is the datatype.

```
namespace sca_ln
{
class sca_node_base : public sca_eln::sca_node
{…};

template<class PDOMAIN>
class sca_node :  public sca_ln::sca_node_base,
                  public sca_ln::sca_node_if<PDOMAIN>
{…};
}
```

Figure 6 Principal implementation of a generalized physical domain node

```
sca_ln::sca_node<sca_translational> mech_node;

class spring : public sca_eln::sca_l
{
public:
  sc_ln::sca_terminal<sca_translational>  t1;
    ...
};
```

Figure 7 Example for usage of generalized physical domain nodes and terminals

Using this approach, libraries of different physical domains can be easily created. Thus libraries for mechanical, magnetic, fluidic and radiant domains where implemented.

V. Using Open Source Qemu And SystemC (AMS) to solve the problem of Industrial Continuous Integration (CI)

A. Whats special about CI

Today, continuous integration (CI) using complex hardware/software environments is expensive and resource constrained.
The hardware diversity leads to expense, both related to the devices themselves and to the manpower needed to maintain them. Equally debug for each environment can be different and painful to setup and use. The more board you have, the more issues you encounter that require manual and time consuming interventions.
Hence, this does not scale well. As more and more projects require a larger and larger number of components, this gets worse and worse.
A natural solution to this issue is to replace the hardware parts with virtual platforms.
Virtual platforms simulate a system at the functional level, that is, everything needed to run the software stack.
Virtual platforms are fast, provide a great opportunity for scalability, have a consistent interface, no need for specific debug environments and are portable. They can be used on an engineer's desk top.

B. How models fit

Virtual platforms can be used to model virtually any kind of systems, including both digital and analogue components. It brings great flexibility compared to the real hardware, by allowing better reconfigurability and observability of the system. For example, one can easily pause the simulation and observe the registers of a given peripheral, without being intrusive. But equally, (depending on the business model and costs), there is no limit to the number of virtual platforms that can be run. This is especially useful to test mass IoT applications. This approach scales naturally, by adding more compute nodes to simulate more devices. And their flexibility allows for virtual platforms to be easily integrated in a standard development flows. Furthermore, the fit naturally into a continuous integration environment as they can be run in a 'headless' batch mode, requiring no human intervention.

C. Industrial case

Successful deployment in an industrial environment shows a real advantage of having virtual platforms within a CI flow. Each commit in firmware repositories triggers the integration tests to be re-run, and gives a quick feedback if something goes wrong. Removed the hardware allows for total and reliable automation, while greatly reducing the integration costs.

D. Economic challenge

Our virtual platforms are based on Open Source software like QEMU, the well-known system emulator and the well-established standard SystemC to model the hardware. This means the the size of the CI test farm is not a financial consideration in terms of models.

VI. APPLICATION EXAMPLE

Typical IOT systems consist of numerous independent sensors as well as control units. A subset of a more complex model could be the thermal house model discussed below. It includes a QEMU based digital heat controller running a control firmware as well as a complex heterogeneous environment around it.

The QEMU platform includes an ARM-A7 processor and computes the control signal for the heater based on

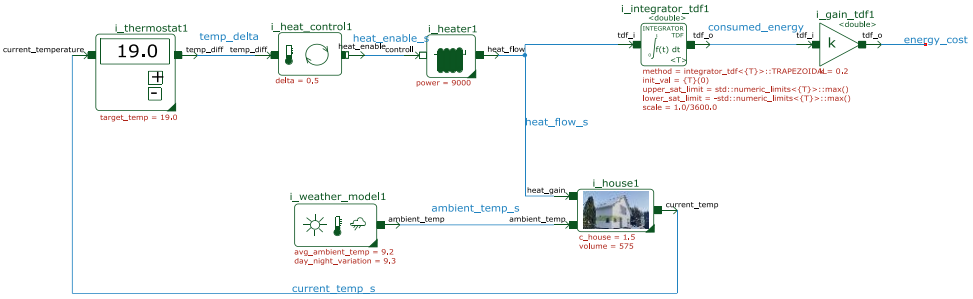


Figure 8 Thermal house model – toplevel view

the user input as well as the current indoor temperature of the house. The firmware reads in the temperature from an analog voltage via an ADC and writes the controls to digital outputs.

The surrounding mixed signal environment consists of a thermostat, a weather model, a heater and the house model itself. The thermostat converts the actual temperature into an analog voltage for the controller. The weather model provides the ambient temperature to the house by modeling a day and night cycle around the average ambient temperature. The heater provides a conditional heat power based on its digital controls.

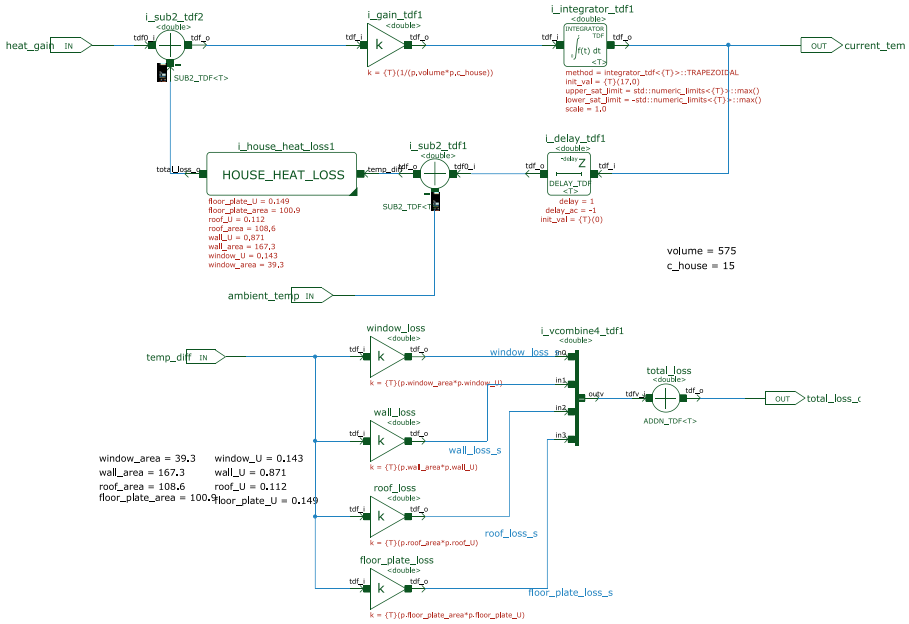


Figure 2 a) house b) heat loses

The house model itself takes the ambient temperature as well as the conditional heat power as input to calculate the current indoor temperature. The energy flow relates to the temperature (stored energy) via the volume and average heat capacitance of the house. To calculate the resulting energy flow the model considers the heat power

as well as a number of loses (wall, window, roof, floor plate ...), which each depend on their corresponding area and heat conductivity in relation to the supplied ambient temperature.

Additionally the model provides some calculation blocks to integrate the energy flow up to a consumed energy and its corresponding cost.

A. Results and Performance

The simulation run represents 3 day night which were equally split into 72000 steps. To computational afford is very fast and consumes around 81seconds. The results show in the waveform are the ambient and indoor temperature, the digital control of the heater as well as the consumed energy cost. It shows that the heating is triggered more frequently during the night and totally switched of during the day (ambient temperature peeks), where the indoor temperature increases due to the high ambient temperature.

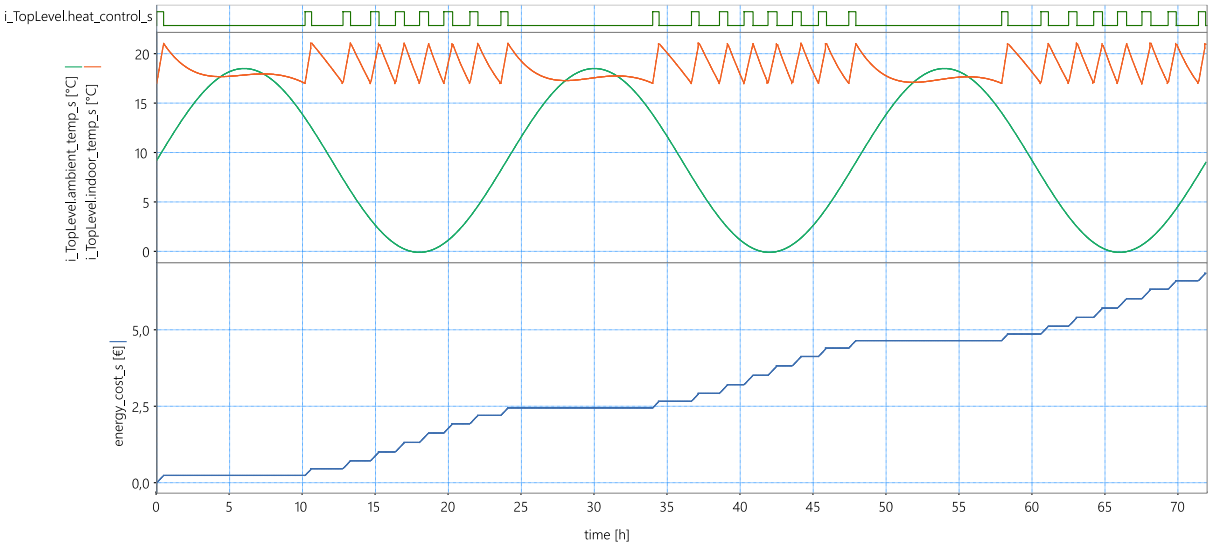


Figure 9 Simulation result

REFERENCES

[1] IEEE Computer Society, 1666-2005 IEEE Standard SystemC Language Reference Manual

[2] IEEE Computer Society, 1666.1-2011 IEEE Standard SystemC Analog/Mixed Signal Extensions Language Reference Manual

[3] Einwich, K.; Uhle, T. (2010): SystemC AMS - holistic analog, digital, hardware and software system-level modeling. In: *EDA Tech Forum Journal* 7 (1), S. 28–33.

[4] R. Lerch, G. M. Sessler and D. Wolf, Technische Akustik: Grundlagen und Anwendungen, Springer Berlin Heidelberg, 2009, pp. 281-289.

[5] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." USENIX Annual Technical Conference, FREENIX Track. 2005.

[6] Fowler, Martin, and Matthew Foemmel. "Continuous integration." Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf (2006): 122.

[7] Neul, R. et al.: A modeling approach to include mechanical microsystem components into system simulation. Proc. Design, Automation & Test Conf. (DATE'98), Paris, 1998, pp. 510-517.

[8] R. Lerch, G. M. Sessler and D. Wolf, Technische Akustik: Grundlagen und Anwendungen, Springer Berlin Heidelberg, 2009, pp. 281-289.

[9] Montón, Màrius, Jordi Carrabina, and Mark Burton. "Mixed simulation kernels for high performance virtual platforms." *Specification & Design Languages, 2009. FDL 2009. Forum on*. IEEE, 2009.

[10] G. Delbergue, M. Burton, B. Le Gal and C. Jego. Multi-threaded Virtual Platform Simulation: An open-source approach, using SystemC TLM-2.0, and QEMU. In Proceedings of the Forum on specification & Design Languages (FDL'15), Barcelona, Spain, September 14-16, 2015.