RISC-V AMS VP: An Open Source Evaluation Platform for Cyber-Physical Systems

Sallar Ahmadi-Pour
Institute of Computer Science,
University of Bremen
Bremen, Germany
sallar@uni-bremen.de

Vladimir Herdt
Institute of Computer Science,
University of Bremen
Cyber-Physical Systems, DFKI GmbH
Bremen, Germany
vherdt@uni-bremen.de

Rolf Drechsler
Institute of Computer Science,
University of Bremen
Cyber-Physical Systems, DFKI GmbH
Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Recently, *Virtual Prototypes* (VPs) implemented in SystemC TLM (*Transaction-Level Modeling*) have been introduced into the growing RISC-V ecosystem to facilitate early software development and testing. However, accurate environment modeling, which is crucial for Cyber-Physical Systems (CPS), has been mostly neglected to this point.**

Thus, in this paper, we propose the RISC-V AMS VP framework, that combines an existing open source RISC-V VP with the SystemC AMS (*Analog/Mixed Signal*) environment modeling style to obtain a RISC-V evaluation platform tailored for CPS. As a case study we created a temperature control system that integrates a sensor and heater component together with a control software. Moreover, we present results on an exemplary fault-injection evaluation that is enabled by bringing together software, hardware and environment models in our unified RISC-V AMS VP framework. Finally, we provide the RISC-V AMS VP framework together with the temperature control system as open source to stimulate further research and as foundation for educational purposes.

I. Introduction

RISC-V is a modern free and open source *Instruction Set Architecture* [1], [2] that gained an enormous popularity in academia and industry. A key feature of RISC-V is the modular and extensible design. Starting with a small mandatory base integer ISA, a set of optional standard instruction set extensions, such as compressed instructions or multiplication are defined. Moreover, to further boost performance and energy efficiency, custom instruction set extensions can be integrated. This enables to build highly efficient application specific solutions which are very suitable for resource constrained embedded devices such as employed in the *Internet of Things* (IoT) domain or other *Cyber-Physical Systems* (CPS).

An important part of RISC-V is the continuously growing ecosystem that includes a large set of tools, simulators and processor implementations - both open source and commercial - to support the development flow. Recently, Virtual Prototypes (VPs), such as the open source RISC-V VP [3], have been introduced into the RISC-V ecosystem to facilitate early software development and testing as well as other system level use-cases. A VP is essentially an executable abstract model of the entire hardware platform and predominantly created in SystemC TLM (*Transaction-Level Modeling*) [4]. VPs enable to capture interactions between the hardware and software accurately. However, in addition accurate environment models are required which are crucial for CPS. SystemC AMS (*Analog/Mixed Signal*) [5] is a modeling standard to provide such environment models, but has not been integrated with open VP solutions for RISC-V so far.

Therefore, in this paper we propose to combine the open source RISC-V VP [3] with SystemC AMS to obtain a RISC-V evaluation platform tailored for CPS. We call this combined framework **RISC-V AMS VP** (Section IV). As a case study we created a temperature control system that integrates a sensor and heater component together with a control software running on a RISC-V platform (Section V). It illustrates the modeling principles of RISC-V AMS VP, which can be applied to build other CPS. RISC-V AMS VP provides a unified view on the software, the hardware in form of the VP and AMS environment model. This allows the design and verification engineers to accurately interact with the system at all different levels, which in turn facilitates to perform powerful analyses. We demonstrate this use-case by presenting exemplary results on a fault-injection evaluation which considers different modeling layers (Section VI). We believe that RISC-V AMS VP is a suitable foundation for further research and education. Therefore, we provide the RISC-V AMS VP framework together with the temperature control system case study as open source (GitHub link will follow in the final version).

II. Related Work

Considering RISC-V, there exist a number of simulators such as the reference simulator SPIKE [6], RISCV-QEMU [7], gem5 [8], [9], RV8 [10] or Renode [11]. They differ in their implementation techniques and intended use-case which range from mainly pure CPU simulation (SPIKE, RV8) to full-system simulation (QEMU, gem5) and even support for multi-node networks of embedded systems (Renode). However, they are mainly designed to simulate as fast

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project VerSys under contract no. 01IW19001 and within the project Scale4Edge under contract no. 16ME0127.

978-1-6654-1825-6/21/$31.00 ©2021 IEEE

as possible and primarily focus on modeling interactions between hardware and software, but only offer limited modeling capabilities to support environment interactions. Moreover, they are not designed to integrate well with the standardized SystemC modeling style. Therefore, recently, SystemC-based solutions have emerged in the RISC-V ecosystem. One example is the open source RISC-V VP [3], that we use as foundation in this work, which is implemented in SystemC TLM and hence provides a foundation for advanced SystemC-based use-cases. Viable alternatives in this context are the DBT-RISE [12] framework and the ETISS [13] ISS which are also designed with a SystemC integration in mind. With regard to DBT-RISE, an example VP platform implemented in SystemC TLM is provided [14] which integrates a RISC-V ISS [15]. Another SystemC TLM simulator for RISC-V is RISC-V-TLM [16], which is currently under active development to improve the RISC-V feature support. However, while these SystemC-based solutions provide a strong foundation to enable accurate and extensible modeling of hardware and software interactions, accurate environment modeling has been mostly neglected so far. Commercial VP tools, e.g. Synopsys Virtualizer or Mentor Vista, might support RISC-V in combination with accurate environment models in a unified VP framework, but their implementation is proprietary. Finally, some approaches exist that have been designed to formalize the RISC-V ISA semantics, e.g. SAIL [17] and GRIFT [18], which also provide or can generate simulator backends. However, these simulation backends are tailored for software execution and thus are not suitable to model extensive and accurate environment interactions.

Looking beyond RISC-V, for the simulation of hardware and physical environments, there exist a lot of commercial and free solutions. Among the prominent commercial tools there exists Mathworks MATLAB/Simulink [19], Wolfram Mathematica [20] and Maplesoft Maple [21]. The tools all offer extensions for the modeling and simulation of various physical systems, control and hardware systems as well as tasks around simulation and analysis of systems and control. There exist open-source alternatives to tools mentioned like Octave [22], SciLab [23] and the Python library scipy [24]. They partially offer the same set of features while being freely available without a commercial license. Another open-source solution for the simulation of heterogeneous systems is the Ptolemy Project by UC Berkley [25]. Ptolemy is an open-source software framework for modeling and simulating complex actor-oriented heterogeneous systems offering a variety of computational models. Neither the commercial tools with their alternatives, nor Ptolemy offer the simulation of software execution with hardware/environment interaction.

The RISC-V AMS VP closes the gap between the modeling of heterogeneous systems containing hardware with physical environments and their software driven interaction. Moreover, is is a beneficial open source extension to the growing RISC-V ecosystem.

III. Background: SystemC TLM and AMS

This section provides relevant background information on the SystemC TLM (Section III-A) and AMS (Section III-B) modeling standards.

A. TLM

SystemC TLM-2.0 is an industry-proven modeling standard used for the design of VPs. TLM adds a set of interfaces and coding styles that allow for significant improvement of simulation speed when compared with the traditional SystemC style with signals, ports, modules and event-driven processes. With TLM this signal and port based communication is abstracted into transactions that are routed on a memory mapped bus system from an initiator to a target. Transactions consist of a command (e.g. read/write), a pointer to the transaction payload, a start address and the payload length. To further improve the simulation performance two commonly utilized techniques are used: *Direct Memory Interface* (DMI) and *Time Quantum* (TQ). DMI allows to bypass the memory mapped bus and access specific address ranges directly through a pointer. TQ allows processes to postpone the synchronization with the SystemC kernel, which can be very beneficial in the VP context to avoid synchronization in the *Instruction Set Simulators* (ISS) after every executed instruction.

B. AMS

The SystemC extension for AMS adds new features regarding the interaction of embedded systems with their physical environment. A direct use-case implies embedded systems containing components like radio frequency interfaces, power electronics, sensors and actors. The formalism for designing, simulating and verifying these components can also be used to extend the use-cases for modeling environments connected to these components. SystemC AMS adds three formalisms, called *Models of Computation* (MoC), to the SystemC modeling library. These three MoC are *Timed Data Flow* (TDF), *Linear Signal Flow* (LSF) and *Electic Linear Networks* (ELN).

In TDF, models are described in discrete-time modules which calculate their dynamics according to a fixed schedule. The modules are interconnected to, so called, TDF clusters. With the static schedules the simulation overhead for the discrete-event simulation kernel of SystemC is minimized. In LSF, models are described as continuous-time modules through a set of primitive modules like add, multiply, integration or delay. Through these primitives any linear *Differential Algebraic Equation* (DAE) system can be described. The models are described with a block diagram notion and are solved through a DAE solver. In ELN, models represent electrical networks designed from predefined primitives like sources (voltage or current), linear lumped elements (resistors, capacitors and inductors) as well a restricted set of linear primitives and switches for modeling electrical energy conserving behavior. As the voltage and current in these networks behave according to the Kirchhoff's voltage law and Kirchhoff's current law, the models can be represented through DAE and thus be simulated through a DAE solver like the LSF models.

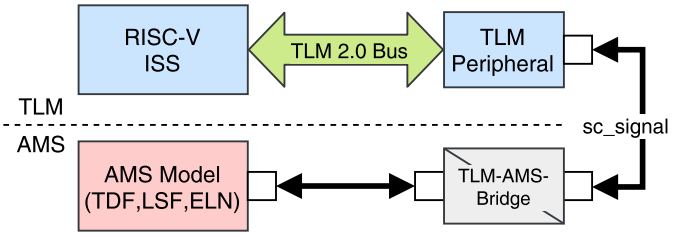

Fig. 1. Modeling approach with the RISC-V AMS VP

Further AMS support verification techniques for time-domain and frequency-domain analysis, which can be applied to all AMS MoC.

The AMS extension aims towards interoperability with TLM and therefore offers interfaces between the TDF and discrete-event domain of TLM. For more details on the specification and features of SystemC AMS refer to the official language reference [5] or user guide [26].

IV. RISC-V AMS VP Framework

The RISC-V AMS VP is an extension of the open source RISC-V VP [3] offering SystemC AMS capabilities to simulate highly heterogeneous embedded systems. The VP uses fast TLM based software/hardware simulation while AMS enables the domain of analog/mixed signal and physical environments to be part of the simulation. Such a combination is very important, because modern CPS have a strong coupling between the software/hardware interaction and the physical environment they are embedded in. Thus, the RISC-V AMS VP provides a solution to design, simulate and verify CPS in a unified approach, tailored for RISC-V. Following the VP-based design flow, RISC-V AMS VP serves as a reliable executable specification and reference model of the CPS, from the specification phase, through refinement steps and up to deployment.

Fig. 1 shows the design approach for AMS modeling with the RISC-V AMS VP. The interaction with the AMS components and environments occurs through a TLM peripheral. At first the TLM peripheral can directly manipulate the I/O ports that are available on the AMS model, thus providing a functionally correct but abstract implementation. For further refinement, more detailed communication protocols can be implemented by leveraging *Intellectual Property* (IP) blocks commonly used and available on *System on Chip* (SoC) designs (i.e. PWM, I2C, etc.). On the AMS layer the environment models can be designed in a similar fashion, following an iterative demand-driven refinement procedure. Starting with a lightweight and undetailed model that covers the interface towards the SoC, more details can be added to cover additional aspects which are of interest to the designed system. For example, consider a temperature sensor which can initially be modeled as a functional mock-up that directly delivers measured temperature values into TLM registers. A subsequent refinement can then include additional processing steps which mimic the real behavior of a sensor in a physical environment,

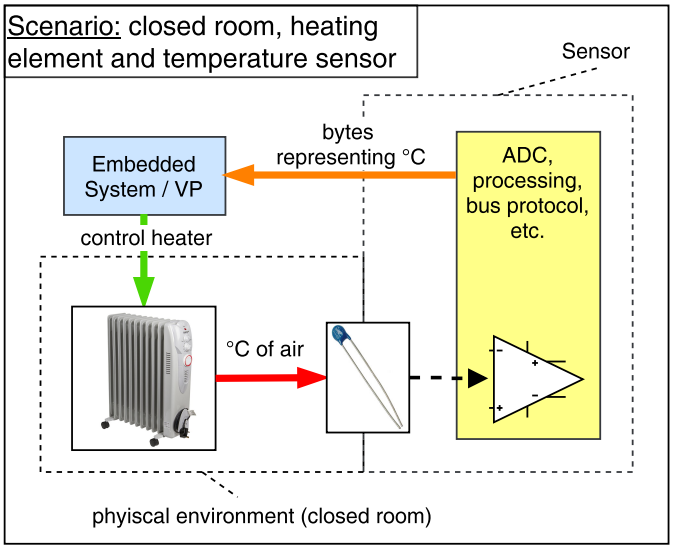

Fig. 2. System level view of our temperature control system

such as amplification or analog to digital conversion. To illustrate the modeling principles of RISC-V AMS VP, we present a temperature control system which we created using RISC-V AMS VP in the next section.

V. Temperature Control System Case Study

In this section we describe the temperature control system, which we created as a case study using our RISC-V AMS VP framework. We start with a general system level overview (Section V-A) and then present a refined architecture view that shows the hardware, software and environment model together (Section V-B). Next, we detail the temperature control algorithm (Section V-C) and finish with a concrete setup and application scenario for the temperature control system (Section V-D).

For differentiation we use the mathematical script (e.g $T(t)$ or T_{off}) for physical/mathematical parameters of the System and standard or verbatim font (e.g. `heat_o` or heat_o) for software/hardware parameters.

A. High-Level System View

Fig. 2 shows the system level view of our temperature control system case study. It consists of three main parts: 1) a heater that can be switched on or off, 2) a temperature sensor that provides the current temperature value, and 3) an embedded system that controls the heater based on periodic temperature sensor readings. We consider a closed room scenario. The heater and sensor operate in a physical environment. They control and pick-up the environment temperature, respectively. We represent the embedded system as well as models of the heater, sensor and environment temperature using the RISC-V AMS VP framework. The temperature control algorithm is implemented in software which is executed on the VP. Therefore, the software is periodically triggered via sensor interrupts. Inside the sensor there are multiple steps to process

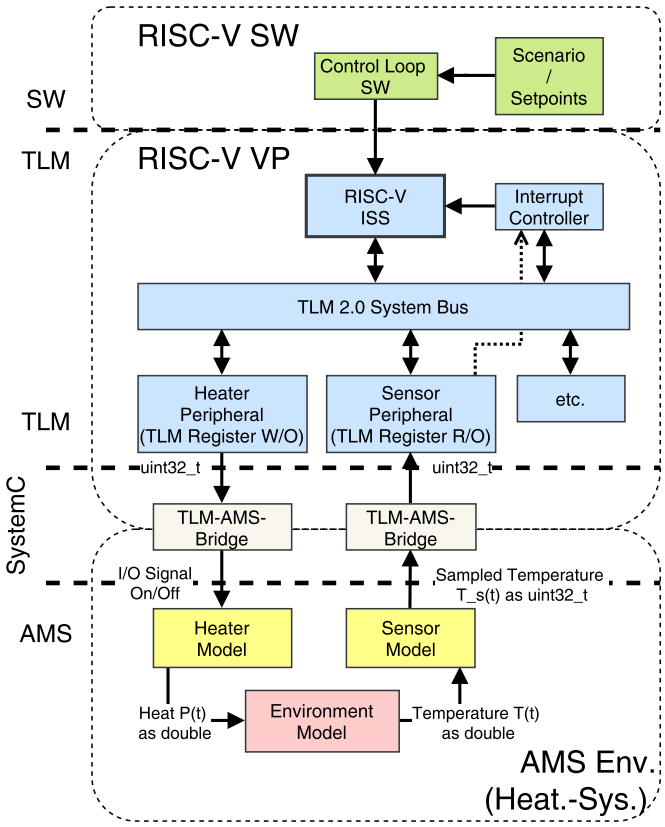

Fig. 3. Architecture view of the temperature control system as implemented using our RISC-V AMS VP framework

the raw temperature which is measured from the environment. First the heat changes the resistance of a sensor element (for example a PTC), the change in resistance then causes a change in voltage or current, which in turn is picked up by an amplification circuit and made available as readable bytes representing the temperature through an *Analog Digital Converter* (ADC). This processed temperature value is then picked up by the software as input for the control algorithm.

B. Refined Architecture View

Fig. 3 shows the architecture of the system and can be considered a refined view of the system level. It consists of three main parts: 1) the control software (top of Fig. 3), 2) the TLM-based VP that represents the embedded system and executes the control SW (middle of Fig. 3), and 3) the AMS-based sensor, heater and environment models (bottom of Fig. 3). To build the VP, we extended the base RISC-V VP configuration[1] by TLM peripherals representing the interaction with the heater and temperature sensor models. On the TLM level these peripherals provide registers for the state of the heater and sensor processed temperature using C/C++ types (`uint32_t`). These registers are accessed from the software side and they are connected to the AMS based

[1]The open source available RISC-V VP base configuration includes the ISS, interrupt controller and TLM bus system among others.

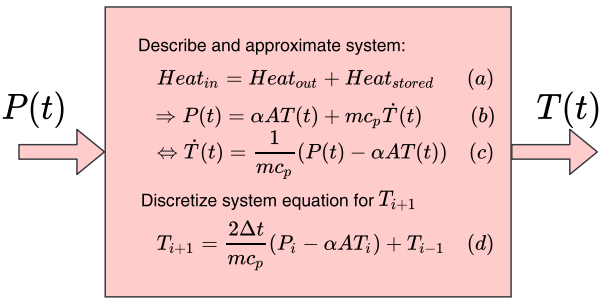

Fig. 4. Environment model with model derivation

environment simulation. The C/C++ types are converted and translated to their physical representations. If the heater is turned on, it emits a fixed amount of power as heat (5 W). The resulting temperature is fed into the sensor model in the AMS environment. For practical purposes we choose a temperature resolution of 1 °C as this is sufficient enough for temperature control since the modeled sensor (modeled after a Maxim Integrated DS18B20 Temperature Sensor[2])provides a accuracy of 0.5 °C.

Fig. 4 shows the description of the environment temperature model. The physical system is represented by the heat equation (equation (a) and (b) in Fig. 4). For the model it is assumed that the heat put into the system ($Heat_{in}$) either leaves the system through conduction ($Heat_{out}$) or is stored in the volume ($Heat_{stored}$). From this approach the *Ordinary Differential Equation* (ODE) of the model is derived. In order to describe the model with the TDF formalism of SystemC AMS, the ODE is rearranged and transformed. The following transformations is applied to the ODE to discretize it into a difference equation:

$$P(t) = P_i \tag{1}$$

$$T(t) = T_i \tag{2}$$

$$\frac{T(t)}{dt} = \frac{T_{i+1} - T_{i-1}}{2\Delta t} \tag{3}$$

This transformation discretizes the time into timsteps i (current timestep), $i-1$ (last timestep) and $i+1$ (next timestep) with Δt being the stepsize between the timesteps. Equation (d) of Fig. 4 is obtained after inserting the transformations and rearranging the terms for the temperature of the next time step T_{i+1} in dependence of the values of the previous timesteps (T_{i-1}, T_i).

C. Temperature Control Algorithm

Various control algorithms are available to build a temperature controller. For our case study we consider a hysteresis controller. This type of controller is very popular in various embedded application domains because of two key properties: 1) it allows for a lightweight design, and 2) it enables to reduce the wear on switching elements (such as a relay) and thus

[2]For more information the datasheet can be found on https://www.maximintegrated.com/en/products/sensors/DS18B20.html

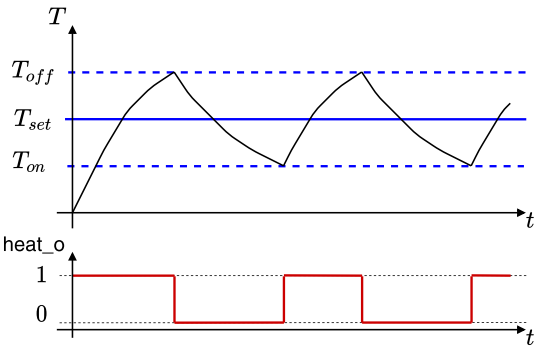

Fig. 5. Hysteresis-based control to reach and stay at a temperature setpoint

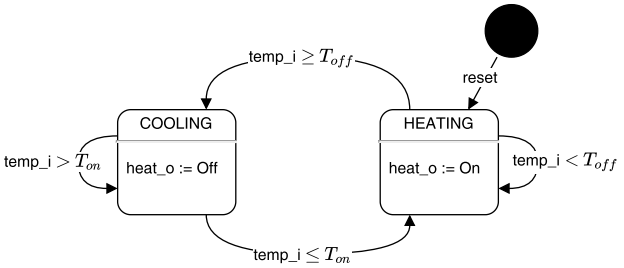

Fig. 6. FSM-based hysteresis control algorithm, which we implemented in software

prolongs the lifetime of the components. In hysteresis-based control the temperature setpoint (T_{set}) is reached by keeping the actual temperature value T around the setpoint. Therefore, an upper bound T_{off} is defined, at which the controller turns off the heat, and a lower bound T_{on}, at which the controller turns on the heat. Thus, the resulting temperature stays within a range around the desired temperature setpoint.

Fig. 5 shows a simplified graph of the desired hysteresis control over time. The top plot in Fig. 5 depicts the temperature T over time with the setpoint T_{set} (blue horizontal line) and the bounds T_{on}, T_{off} (dashed blue horizontal lines). The bottom plot in Fig. 5 shows the state of the heater (1 = on, 0 = off). The top plot shows that the temperature T is first increased over time, until it reaches the upper bound T_{off}. If the temperature is equal or bigger than T_{off} the controller turns off the heat and waits for the temperature to drop again. The temperature will drop until the lower bound T_{on} is reached. If the temperature is less or equal than T_{on} the controller will turn on the heat again and wait until it reaches the upper bound again. Thus, resulting in a sawtooth shaped curve for the temperature T over time.

To implement the hysteresis control in software, we designed a *Finite State Machine* (FSM) that takes the current temperature $T(t)$ as an input and outputs the state of the heater (on or off) controlling $P(t)$ to emit heat into the system.

Fig. 6 shows this FSM. The current temperature is called `temp_i`, the heat control output is called `heat_o`. The FSM has two states `COOLING` and `HEATING`. Initially the FSM will start in the `HEATING` state. In the state `HEATING` the output `heat_o` will be on (1), and in the state `COOLING` the output

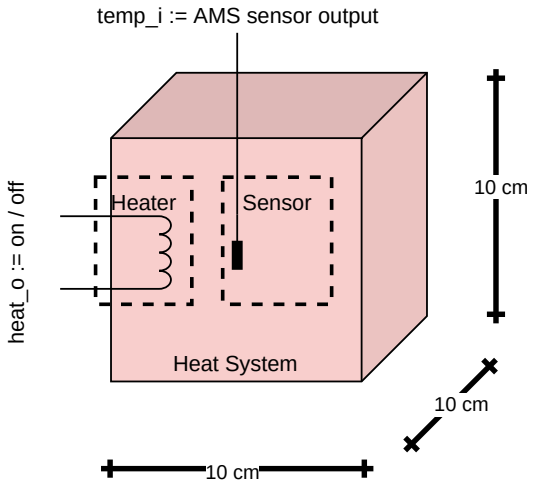

Fig. 7. Heat system modeled in case study for heat control

`heat_o` will be off (0). The FSM stays in the `HEATING` state as long as the temperature `temp_i` is smaller than T_{off}, if the temperature is greater or equal than T_{off} then it will transition into the `COOLING` state. Respectively, the FSM stays in the `COOLING` state as long as the temperature `temp_i` is bigger than T_{on}, and if the temperature is smaller or equal than T_{on} then it will transition into the `HEATING` state. Note that the guards for transitioning the states are mutually exclusive making the FSM deterministic.

The control software implements this FSM-based hysteresis control algorithm in an interrupt driven way. On each periodic temperature sensor interrupt, the software is triggered to process one transition of the FSM which can either change or keep the state of the FSM.

D. Evaluation Setup and Scenario

In this case study we exemplary model an enclosed and air-filled cube of 10 cm width, height and depth. For an illustration of the modeled setup refer to Fig. 7. We assume a heating element with 5 W of heating power that can be either turned on or off. For the cube we choose the following configuration parameters of the heat system:

$$\alpha = 5.6\ \mathrm{W/m^2K}$$
$$A = 0.06\ \mathrm{m^2}$$
$$m = 0.001269\ \mathrm{kg}$$
$$c_p = 1005\ \mathrm{J/kgK}$$

As an approximation, we assume the airmass inside the volume to be constant over time[3]. For the hysteresis controller we choose a hysteresis band of ± 1 °C. Thus, for a setpoint of $T_{set} = 20$ °C we get $T_{on} = 19$ °C and $T_{off} = 21$ °C. The AMS model is simulated with a stepsize $\Delta t = 1$ ms. Inside the TLM sensor module the temperature is sampled every 5 ms.

[3]While, in general a change in temperature has an effect on the airmass, for our case study the effect is mostly negligible.

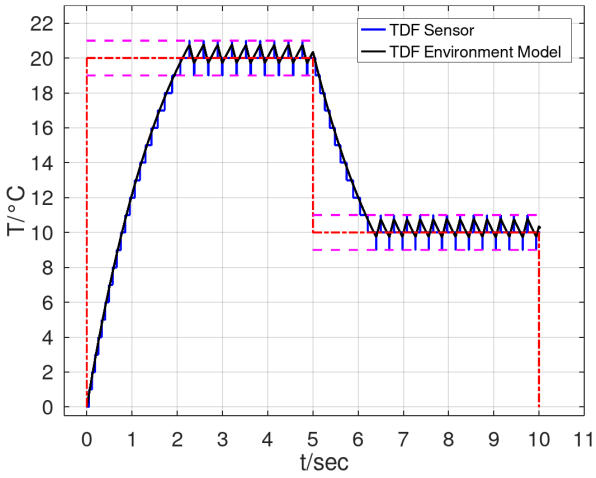

Fig. 8. Heat control for two setpoints (20 °C and 10 °C)

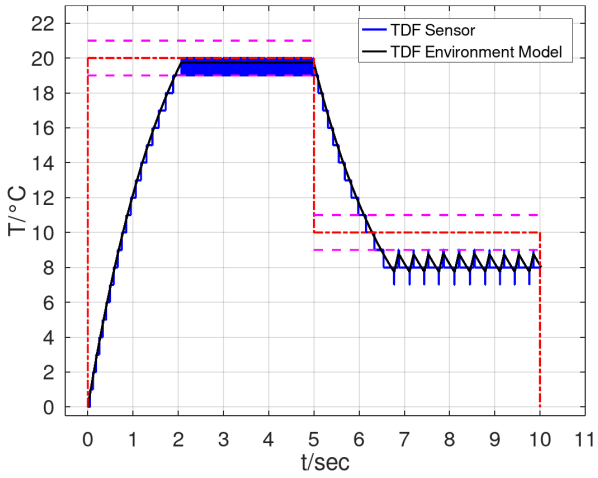

Fig. 9. Fault injection in TLM module

The RISC-V ISS is configured to execute the instructions at a cycle time of 500 ns.

For evaluation purposes we consider a scenario with two setpoints, the first at 20 °C and the second at 10 °C. Both setpoints are held for a defined number of time steps to represent a realistic temperature control scenario. Fig. 8 shows the temperature of the system traced from the environment directly (black line) and the trace of the sensor as read by the control loop software (blue line). Additionally the temperature setpoint is displayed as red dashed-dotted line, with the hysteresis band as the purple dashed lines. The hysteresis controller produces the characteristic sawtooth shape on the actual temperature. This control scenario serves as a baseline for the exemplary fault injection in the next Section VI.

Please note, that all these configuration and scenario parameters are configurable and thus can be easily adapted if necessary.

VI. Fault Injection Evaluation

Build on RISC-V AMS VP, our temperature control system (Section V) provides a unified view on the software, the hardware in form of the VP and AMS environment model. This allows an interaction with the CPS at different levels to obtain accurate analysis results. We demonstrate this use-case by presenting exemplary results on a fault-injection evaluation which considers different modeling layers, in particular: 1) the sensor TLM peripheral, 2) the ISS, and 3) the control loop software. This covers a broad range of potential error scenarios and as such provides the foundation for verification engineers to reason about the robustness of the CPS more comprehensively. We describe the observed effects of these three exemplary fault-injections, using the parameters and scenario from Section V-D as a baseline, in the following.

1) Inside the sensor TLM peripheral a stuck-at fault is inserted into the memory mapped register that holds the temperature value obtained from the AMS environment sensor model. The resulting controller behavior is shown in Fig. 9. For the first setpoint at 20 °C the control loop holds the actual temperature within the specified range, but switches the heater very rapidly. The second setpoint at 10 °C is not reached correctly and instead the control loop stabilizes at 8 °C.

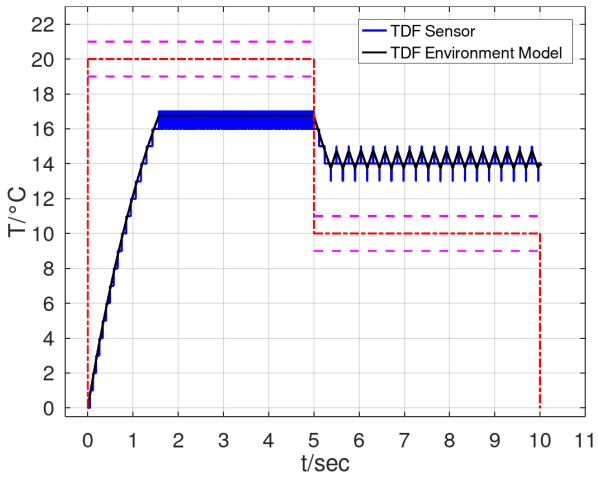

Fig. 10. Fault injection in ISS

2) In the ISS a stuck-at fault is injected into a register for a defined range of program counter values. The range of the program counter affects a section of the control loop software. The resulting controller behavior is shown in Fig. 10. Both setpoints are not reached correctly and at both stable temperatures the switching of the heating element occurs more frequently compared to the baseline.
3) Inside the software level the guards of the FSM for the state `HEATING` are swapped. The resulting controller behavior is shown in Fig. 11. For the first setpoint at 20 °C the temperature is held within the hysteresis band but with higher switching frequency. The second setpoint at 10 °C is not reached and after analyzing the software

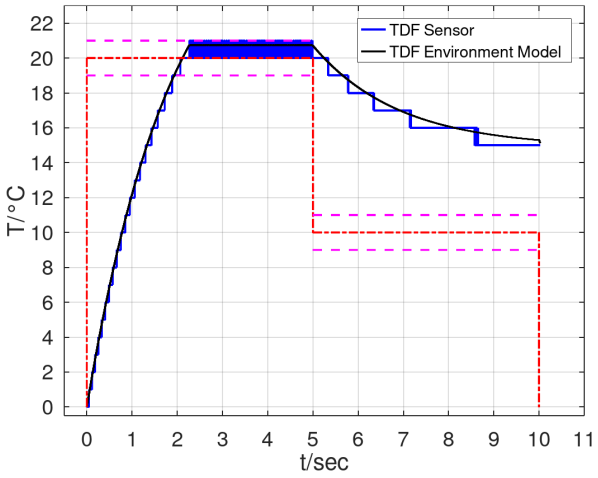

Fig. 11. Fault injection in control loop software

behavior and the logged wavetraces the resulting behavior of the actual temperature comes from the heater being switched at a constant frequency as soon as the second setpoint is set as the target temperature.

Faults injected at different modeling layers can have several effects and result in different errors at runtime. Errors like high switching frequencies will cause increased wear on the switching element (e.g. relay or triac). Other errors like wrong actual temperatures are even more critical as the controller does not fulfill its purpose correctly anymore. Therefore, it is important to have a unified framework available that allows to reason about the different modeling layers and accurately propagate information across them.

VII. Conclusion and Future Work

In this paper we proposed RISC-V AMS VP, an open source RISC-V evaluation platform for CPS. As a case study we created a temperature control system to illustrate the the modeling principles of RISC-V AMS VP. RISC-V AMS VP integrates the TLM and AMS modeling styles to provide a framework that allows a unified reasoning on the the software, the hardware in form of the VP and AMS environment model. Such a unified view enables to describe and reason accurately about system interactions at all these different levels, which is a strong foundation to perform powerful analyses. We demonstrated this use-case by presenting exemplary results on a fault-injection evaluation which allows to track the fault-injection effects accurately across the different modeling layers. To stimulate further research and education, we provide the RISC-V AMS VP framework together with the temperature control system case study as open source (GitHub link will follow in the final version).

To further boost our approach, for future work we plan to:

- Consider cross-level and hybrid modeling aspects that integrate components at the *Register-Transfer Level* (RTL) as well as real hardware components with the RISC-V AMS VP framework;
- Devise automated refinement procedures that aid in building accurate models at the VP level and also consider integration of models that estimate timing and energy consumption for design space exploration purposes;
- Implement and provide a more comprehensive library of common building blocks (sensors, actuators, models) specifically designed for CPS, which integrate with RISC-V AMS VP;
- Investigate integration of advanced verification techniques such as fuzzing and symbolic execution to generate test inputs in order to obtain comprehensive coverage results tailored for the RISC-V AMS VP platform.

References

[1] A. Waterman and K. Asanović, Eds., *The RISC-V Instruction Set Manual; Volume I: Unprivileged ISA*, 2019.

[2] ——, *The RISC-V Instruction Set Manual; Volume II: Privileged Architecture*, 2019.

[3] "RISC-V virtual prototype," https://github.com/agra-uni-bremen/riscv-vp.

[4] *IEEE Standard SystemC Language Reference Manual*, IEEE Std. 1666, 2011.

[5] *IEEE Standard for Standard SystemC(R) Analog/Mixed-Signal Extensions Language Reference Manual*, IEEE Std. 1666.1-2016, 2016.

[6] "Spike RISC-V ISA simulator," https://github.com/riscv/riscv-isa-sim.

[7] "RISCV-QEMU," https://github.com/riscv/riscv-qemu, accessed: 2018-05-13.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[9] "gem5," https://github.com/gem5/gem5.

[10] "RV8," https://rv8.io, accessed: 2018-05-13.

[11] "Renode," https://renode.io/.

[12] "Dbt-rise-riscv," https://github.com/Minres/DBT-RISE-RISCV, 2021.

[13] "ETISS (extendable translating instruction set simulator)," https://github.com/tum-ei-eda/etiss.

[14] "HIFIVE1-VP," https://git.minres.com/VP/HIFIVE1-VP, 2021.

[15] "DBT-RISE-RISCV," https://github.com/Minres/DBT-RISE-RISCV, 2021.

[16] "RISC-V-TLM," https://github.com/mariusmm/RISC-V-TLM.

[17] "Riscv sail model," https://github.com/rems-project/sail-riscv.

[18] "GRIFT - galois RISC-V ISA formal tools," https://github.com/GaloisInc/grift.

[19] MATLAB/Simulink, *version 9.8.0 (R2020a)*. Natick, Massachusetts: The MathWorks Inc., 2020.

[20] Wolfram Research, Inc., "Mathematica, Version 12.3," champaign, IL, 2021. [Online]. Available: https://www.wolfram.com/mathematica

[21] a. d. o. W. M. I. Maplesoft, "Maple 2021," waterloo, Ontario, 2021. [Online]. Available: https://www.maplesoft.com/products/Maple/

[22] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*, 2020. [Online]. Available: https://www.gnu.org/software/octave/doc/v5.2.0/

[23] ESI Group, "Scilab." [Online]. Available: https://www.scilab.org/

[24] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[25] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong, and S. Neuendorffer, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/488.html

[26] *IEEE Standard SystemC AMS User Manual*, https://accellera.org/images/downloads/standards/systemc/Accellera_SystemC_AMS_Users_Guide_January_2020.pdf, 2020, accessed: 2021-05-28.