Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

VIRTUAL HARDWARE IN THE LOOP: HYBRID SIMULATION OF DYNAMIC SYSTEMS WITH A VIRTUALIZATION PLATFORM

Jan Reitz
Alexander Gugenheimer
Jürgen Roßmann

Institute for Man-Machine Interaction
RWTH Aachen University
Ahornstraße 55
Aachen, 52074, GERMANY

ABSTRACT

This paper demonstrates the feasibility of co-simulation with virtual hardware as an alternative to Hardware in the Loop (HiL) simulation. Without real-time constraints, Virtual Hardware in the Loop (VHiL) simulation can be performed on general purpose hardware and software. Specific challenges to this kind of simulation, like data exchange and synchronization, are addressed. An exemplary implementation coupling the virtualization platform QEMU and the simulation framework VEROSIM is presented and used in a case study simulating a self balancing robot controlled by a virtual ATmega328p. The virtual hardware's properties are systematically varied. Results show that the effects of peripheral resolution and the processor's clock speed on the overall system behavior can be observed in a VHiL simulation. It is concluded that VHiL can be a viable alternative to HiL simulation, but the applicability of this approach depends on the availability and accuracy of emulators for specific target platforms.

1 INTRODUCTION

Technical systems increasingly rely on embedded systems to reduce costs or enable new features and connectivity. With these advantages comes an increasingly challenging design and development process. Collaboration between engineers from various specialties is required to integrate individual components into these complex system designs. Hardware in the Loop (HiL) simulation is an established tool to cope with this challenge and is widely used in industries that have been dealing with complex cyber-physical systems (CPS), such as aviation and automotive. It allows for safe and efficient testing of individual components and virtual integration tests, even when the complete system is not available. HiL simulation is a co-simulation where at least one component is a real device. To synchronize this co-simulation scenario, simulated time must follow wall time. This leads to real-time constraints on every simulation unit, which necessitates the use of special operating systems and simulation software. Furthermore, specialized hardware with enough computing power to complete simulations and perform data exchange within these timing constraints, as well as support for various I/O devices to connect to real devices are required.

The main proposition of this contribution is that by utilizing virtualized hardware with identical interfaces to the real device and the ability to run code compiled for the target platform, many use-cases of HiL simulation can be realized without these constraints. This idea is explored by implementing a hybrid co-simulation of a flexible simulation framework and a virtualization platform. We call this approach Virtual Hardware in the Loop (VHiL) simulation. The major advantage is that real-time constraints are removed, since the execution of virtualized hardware can be paused. This means that simulation software does not have to run on real-time operating systems and the constraints on the computational complexity

of simulations are removed as well. No special hardware is necessary to communicate with simulated I/O devices. In summary, this approach can be realized using general purpose hardware and software.

Another advantage is that virtualized hardware allows for easy exploration of the design space. It can be modified to add or remove peripherals, change the processor's clock speed or even the processor architecture. It also results in other benefits of simulation models, such as safety, repeatability and the ability to automate tests. This concept is illustrated in Figure 1. It depicts the co-simulation of an embedded system and its technical context. The microcontroller including its built-in peripherals is emulated in a virtualization platform, running code compiled for this target. Additional peripherals, on-board or external, are simulated in separate simulation units. In this example, one is responsible for peripherals with physical interactions, like actors and sensors, the other for data exchange, in this case via USB or Bluetooth.

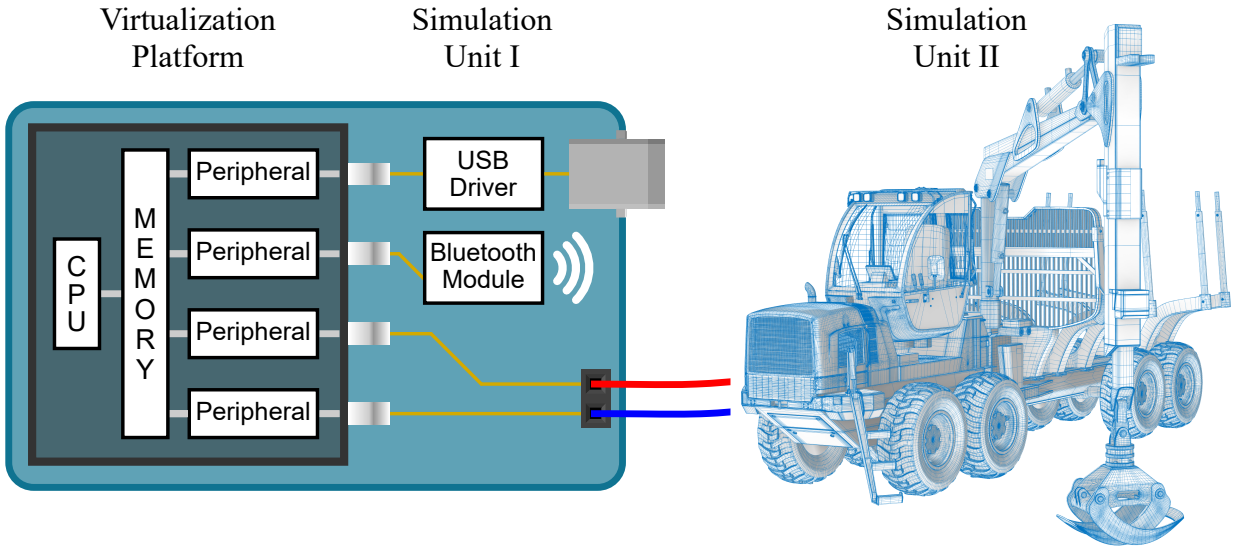

Figure 1: The microcontroller is emulated in the virtualization platform and its technical context is simulated in different simulation units.

1.1 RELATED WORK

The coupling of virtual hardware with other simulation tools is not a novel idea. Various contributions exist that have approached this problem from different angles.

Ptolemy II, presented by (Eker et al. 2003), is a modeling and simulation framework for actor-oriented models. Its goal is unambiguous modeling and simulation of heterogeneous systems, i.e. systems where individual components use different models of computation, such as discrete or continuous time. (Kim et al. 2019) simulate the thermal behavior of DRAM depending on its power usage. They run benchmark programs in the cycle accurate emulator gem5, presented by (Binkert et al. 2011), to estimate the DRAM power usage based on the number of memory accesses. This information is fed into a thermal model created and run in Ptolemy II. This application shows, that gem5 can be used to acquire detailed information about the emulated hardware, but requires long execution times to achieve this. The authors report, that the execution of 0.1 s simulated time ranged from 89 s to 320 s. Data exchange between both simulators is realized using shared files. The simulators are scheduled according to the Gauß-Seidel pattern, where the execution of simulation steps is alternated. This synchronization appears to be unnecessary for the presented application, since information only flows in one direction from gem5 to Ptolemy II. There is no closed loop, so the simulations could be run independently. The authors suggest other use cases for their integration, where the physical environment is simulated in Ptolemy II and an embedded system is emulated by gem5, specifically with regard to educational purposes.

(Verhoef et al. 2014) describe Crescendo, a tool for modeling and co-simulation of CPS. It integrates the Overture tool for discrete-event modeling using the Vienna Development Method and 20-sim, a commercial simulation framework for continuous time models. Based on Crescendo, (Zhang and Broenink 2012) propose a development approach and structuring mechanism for embedded control systems. They present a case study of a robotic cart, where the continuous time aspects are simulated with 20-sim and the controller functionality is modeled using Overture.

(Chaves et al. 2018) present the KhronoSim platform, a distributed architecture for HiL simulation of multiple interdependent CPS. They highlight the modularity and extensibility of their architecture and the possibility to freely substitute real and simulated devices. In this context, (Oliveira et al. 2018) propose the integration of QEMU instances in a KhronoSim scenario to create an emulation-in-the-loop setup. They describe a KhronoSim Extension that creates QEMU instances and controls them using the QEMU Machine Protocol. The execution speed of the emulator is throttled using micro sleeps to synchronize the virtual hardware with the remaining co-simulation. Unfortunately, further details like the data exchange between KhronoSim and QEMU is not specified and no results are reported. The combination of real and virtual hardware in a distributed simulation seems promising. Conceptually it remains a HiL simulation, since the KhronoSim platform imposes real-time constraints.

This contribution shares aspects with every mentioned publication. Like (Kim et al. 2019) and (Oliveira et al. 2018), an emulator is coupled with a general purpose simulation framework. While (Kim et al. 2019) focus on the thermal aspects of the embedded hardware, we focus on the interaction with the remaining system via the microcontroller's peripherals. Compared to (Oliveira et al. 2018), we explicitly treat data exchange and remove real-time constraints from the simulation. Our case study features a closed-loop control example similar to the one presented in (**?**). The approach differs, as our controller is implemented, compiled for and virtually run on the target platform instead of a discrete event model. This allows for the simulation of hardware details and their impact on the overall system behavior.

1.2 ORGANIZATION

In the following section, VHiL simulation is classified in the context of co-simulation. Specific challenges to co-simulation with virtual hardware, such as modeling of microcontroller peripherals, data exchange and synchronization are addressed. In an exemplary implementation the virtualization platform QEMU (Bellard 2005) is coupled with the simulation framework VEROSIM (Rossmann et al. 2013). This implementation is then used for a case-study, where the self balancing robot depicted in Figure 6 is modeled and a VHiL simulation with an ATmega328p is performed. The results section outlines the behavior of the overall system in a VHiL simulation where the controller is executed on virtual hardware. The discussion section includes an analysis of this approach, its applicability and limitations, as well as the conclusion and future research directions.

2 METHODS

Co-simulation refers to the simulation of a scenario consisting of multiple coupled simulation units, each consisting of a simulator, a model and an input trajectory (Gomes et al. 2017). Each simulation unit advances its simulation using individual solvers. An orchestrator synchronizes all simulation units, by controlling the progression of simulated time and the data exchange between simulation units (Blochwitz et al. 2012). It can be realized as an independent component or as part of a simulation unit. In the simplest case, the orchestrator distributes inputs and outputs of each simulation unit according to a scenario specification. It then sends a command to all simulation units to advance their simulated time by a step size $t_i + h = t_{i+1}$.

When coupling simulation units in a co-simulation scenario, the way each unit treats time has to be considered. Time discrete modeling is typically used for embedded systems (Fitzgerald and Pierce 2014), whereas their environment has many aspects that are more appropriately modeled using continuous

time, such as mechanics, electrodynamics or hydraulics. Co-simulation of simulation units with different treatments of time is called hybrid simulation. Next to being a hybrid simulation, VHiL simulation offers special challenges, which will be addressed in the following.

2.1 DATA EXCHANGE

Processors interact with their environment via peripherals. Microcontrollers usually come with a set of integrated peripherals, such as analog digital converters (ADC) or pulse width modulators (PWM). External peripherals can be connected to the microcontroller using general purpose I/O (GPIO) pins or serial interfaces, like universal asynchronous receiver transmitter (UART). As depicted in Figure 1, the virtual hardware system boundary is drawn around the microcontroller. Integrated peripherals are simulated within the virtualization platform and are used to communicate with external peripherals, such as sensors, actuators or networking devices, which are simulated in separate simulation units. This way, connections between simulation units semantically represent connections between components in the real system, which makes modeling intuitive.

Figure 2 depicts a class diagram illustrating the connection of a virtualization platform and a simulation unit. The microcontroller consisting of a CPU, memory, peripherals and an enclosure is simulated within the virtualization platform. These elements are configurable, as is indicated by their attributes. Components of the coupled simulation are connected to a proxy object, which provides a communication interface for data exchange with the virtual hardware. Exchanging data between simulation units and the virtualization

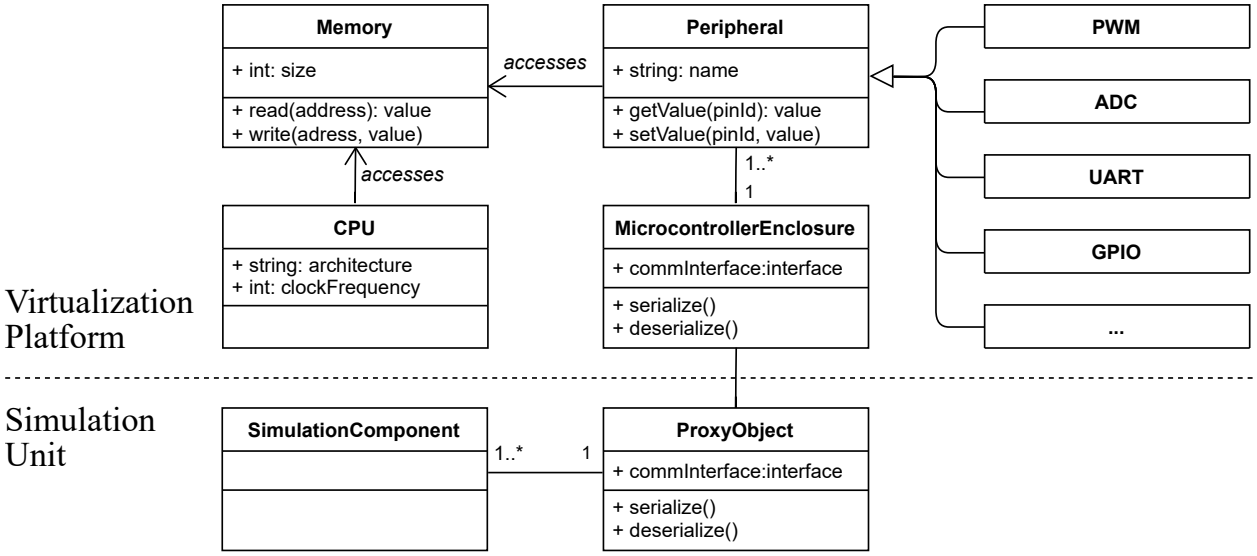

Figure 2: Class diagram of data exchange between virtualization platform and simulation units.

platform requires meta data identifying the connected pin and peripheral type of every signal. This is necessary, because microcontroller pins can be connected to multiple internal peripherals. The ATmega328p, for example, has six pins that can be connected to a PWM or be used as GPIO. This is an important distinction, as a signal carrying 5 V can be interpreted in different ways. If it is a PWM signal, it may be used to control a motor or to provide a clock. In one case, the relevant feature is the effective voltage, in the other, the current high value of a clock signal. If it is a GPIO signal, it may represent a logical value or power to turn on an LED.

(Schroll et al. 2004) introduce polymorphic signals to exchange data between models from different domains. They use semantic types, abstract signal interpretations, to correctly represent the semantics of the transmitted signals. Assigning each signal a peripheral type allows data exchange with meaningful representations, such as booleans for GPIOs, real numbers for ADCs, the duty cycles for PWMs and chars

for serial interfaces. It also enables the receiving model to make an appropriate conversion. This separation of concerns is desirable, because it decouples the virtual hardware from other simulation units and makes it agnostic to the signals usage. Based on this, the protocol depicted in Table 1, consisting of pin identifier, peripheral type, length and payload, is used for data exchange.

Table 1: Data exchange protocol and example messages using the ATmega328p pin mapping.

Field	Description	Example 1	Example 2	Example 3	Example 4
Pin	Microcontroller specific pin ID	PD1	PD3	PD4	PD5
Type	Peripheral type	UART	PWM	GPIO	ADC
Length	-	8 bit	32 bit	1 bit	32 bit
Data	-	'A'	0.645	true	3.647

2.2 ESTIMATING EXECUTION TIME

To synchronize simulation units, the orchestrator must be able to advance each simulation unit with a defined step size. Virtualization platforms are either instruction-accurate or cycle-accurate. An instruction-accurate simulator executes a program's instructions and updates the processor's registers accordingly. A cycle-accurate simulator goes further, as it provides information such as the passed time expressed in a number of clock cycles, power consumption and other metrics. Different CPU cycles, memory latencies and more can be considered and simulated. In general, such simulators are slower than instruction-accurate ones and less common.

When simulating with an instruction-accurate emulator, the amount of time passed can be estimated using the number of executed instructions. Since instructions require at least one cycle, a lower bound on the execution time is given by $t_{lower} = n/f$, where n is the number of instructions, and f the processor's clock speed. This estimation can be improved by storing the number of cycles required for a given instruction and advancing a cycle counter by that amount when executing the instruction. Counting cycles instead of instructions is a more accurate estimation, but not necessarily correct. Depending on the embedded system, caches and other CPU elements make execution times hard to predict (Wolf 2017).

2.3 EXEMPLARY IMPLEMENTATION

The described approach is implemented by coupling the virtualization platform QEMU to the simulation framework VEROSIM. QEMU is a fast, open-source processor emulator using dynamic translation. Multiple processor architectures are supported, such as ARM, SPARC, Coldfire or x64-86 and more can be added (Bellard 2005). It was chosen for its speed and wide user base. The case study requires the AVR architecture, which is not yet officially supported, but a development version is available. This development version has been extended to support the data exchange ideas presented in section 2.1. Our implementation currently supports GPIO, ADC, PWM and UART. It also provides an estimate of execution time as described in section 2.2.

VEROSIM has been chosen for its extensive modeling and simulation functionality, including a modeling GUI, rigid-body dynamics, various actor and sensor simulations, as well as rendering capabilities. All of this functionality is utilized in the case study presented in Section 2.4. VEROSIM includes a plugin interface, which was used to add a software component that serves three functions. First, it provides a proxy object with a communication interface, as depicted in Figure 2. Second, this proxy object is integrated into the modeling GUI so other simulation components can be connected to the virtual hardware. The integration is inspired by the electronics design automation tool Fritzing, presented in (Knörig et al. 2009). Figure 5 shows the intuitive modeling of connections between components in VEROSIM and the virtual hardware. This UI metaphor abstracts the co-simulation aspects from the user and makes it seem like he is

only using one simulation framework. Third, the plugin acts as orchestrator for this co-simulation setup. This is detailed in the next subsection.

2.3.1 COUPLING

Messages between both software components are exchanged via a TCP connection. It is used for data exchange and to send commands, such as *doStep(1 ms)*. Messages are exchanged asynchronously by writing to the TCP buffer. Neither component waits for a response, but reads the TCP buffer when it is ready. Figure 3 shows a sequence diagram of the co-simulation. It is assumed, that both components are already configured and initialized. After receiving the signal to start simulating, VEROSIM transmits the starting values to QEMU. It then sends the *doStep(1ms)* command and immediately proceeds to advance its own simulation with the same step size. This parallel execution is called Jacobi pattern. It is more efficient than the Gauß-Seidel pattern, i.e. advancing each simulation unit sequentially, but does not allow for interpolation using intermediate results of the other simulations. QEMU estimates the amount of instructions to meet the requested step size and executes them. During this code execution, the virtual microcontroller reads and writes to its peripherals. Whenever a pin value changes, QEMU sends a data exchange message using the protocol depicted in 1. After executing the requested amount of instructions, an acknowledgment message is sent. Upon completing its own computation, VEROSIM starts to synchronize, i.e. it writes the received data exchange messages to the pins of the respective proxy object. If multiple changes to the same pin occur during the same step, the older value is overwritten. In this example, VEROSIM takes longer to compute the current step, so the acknowledgement message is already available. If this was not the case, VEROSIM would wait for an acknowledgement message before proceeding. Similarly, QEMU is idle until the next command is received. This is conservative synchronization, as the causal order remains intact. The waiting could be avoided with optimistic synchronization, i.e. accepting violations of causal order. However, this technique requires all involved simulation units to support rollbacks. Next, current pin values are sent to QEMU and the next iteration starts.

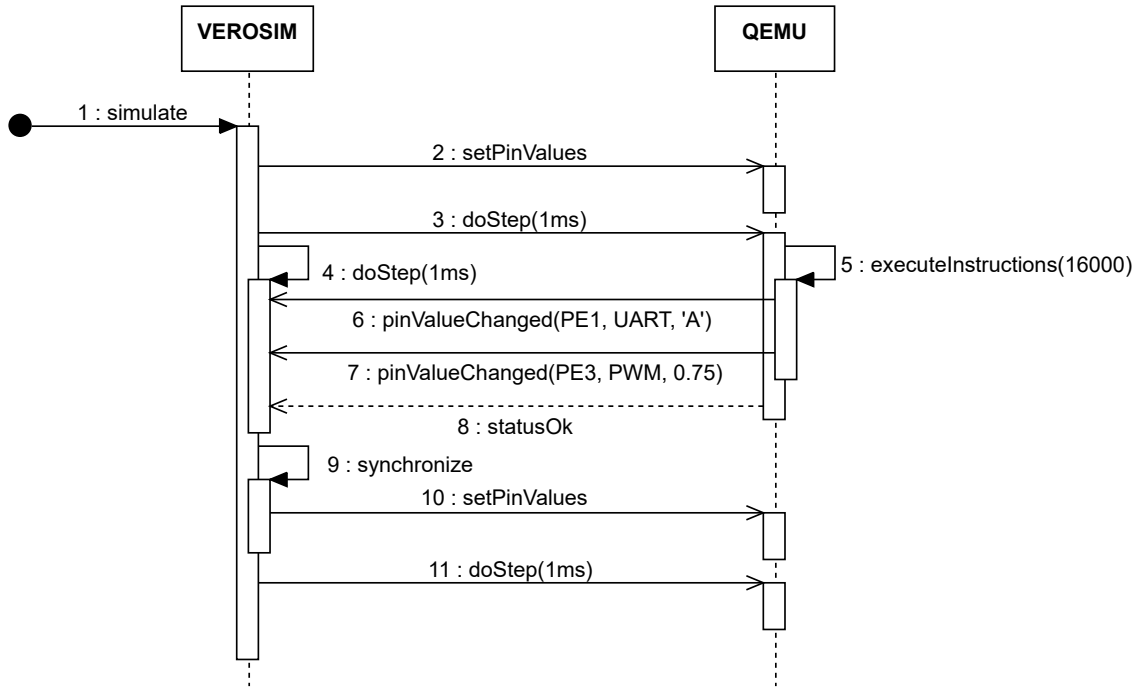

Figure 3: Message flow in a co-simulation of VEROSIM and QEMU.

2.4 CASE STUDY

The exemplary implementation discussed in Section 2.3 is applied to a self balancing wheeled robot. This section will outline the process of modeling and simulating this system using the VHiL approach. The robot's controller is implemented in C and executed on a virtual ATmega328p, the microcontroller of the popular Ardunio UNO. As reference, the controller functionality is also implemented as a VEROSIM plugin. The remaining robot components are modeled and simulated in VEROSIM. Figure 4 illustrates this setup. The balancing robot model consists of a dynamic model, two force based motors driving the

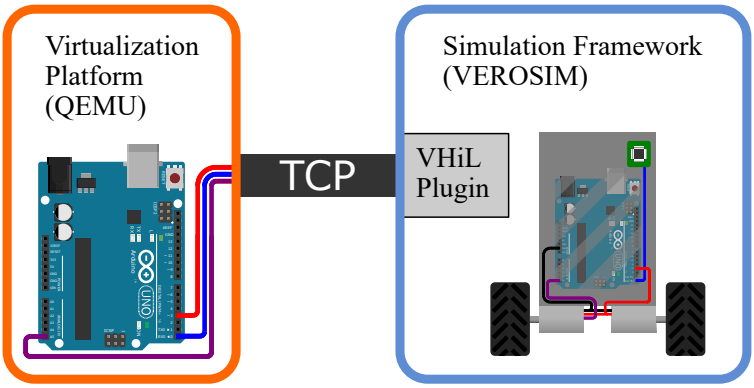

Figure 4: The embedded system is emulated in QEMU on the left. Its peripherals are connected to the embedding system in VEROSIM via a proxy object. The VHiL Plugin in VEROSIM synchronizes both simulations via a TCP connection.

wheels and a motion sensor. The sensor measures the robot's tilt angle and inferred position. The robot dynamics are modeled using three rigid bodies, one for each wheel and one for the chassis. Figure 6 depicts a rendering of this model in VEROSIM. The equations of motion for this model are derived using the second order Lagrange equation, as is illustrated by (Frankovský et al. 2017). The equations of motion are then transformed into a state-space representation. The state space vector $\vec{s}$ consisting of position, x, tilt angle, φ, and their time derivatives, $\dot{x}$ and $\dot{\varphi}$, is defined in Equation (1). The system dynamics are given in Equation (2). A Linear Quadratic Regulator is used as controller. Its control law is given in Equation (3).

$$\vec{s} = \begin{pmatrix} x \\ \dot{x} \\ \varphi \\ \dot{\varphi} \end{pmatrix} \tag{1}$$

$$\dot{\vec{s}} = \boldsymbol{A}\vec{s} + \boldsymbol{B}\vec{u} \tag{2}$$

$$\vec{u} = -\boldsymbol{K}(\vec{s} - \vec{s}_{Set}) \tag{3}$$

First, the controller's functionality is tested using the reference controller implemented as VEROSIM plugin. Once the model is behaving as expected, the controller is substituted by a proxy object representing the virtual hardware. Next, it is connected to the rest of the model. The motion sensors measured data, tilt angle and position, are connected to the ADC. A mapping of the measured tilt angle and position to the ADC's $[0\ ,\ 5]$ V range has to be considered. The motors are connected to the microcontroller's PWM output. Again, a mapping from the PWM signal to an effective torque range has to be set. Next, the controller code is ported from C++ in the context of VEROSIM to C in the context of the virtual hardware. Figure 5 depicts this setup in VEROSIM's modeling GUI. Different simulation components are connected to the proxy object, visualized as Arduino UNO. The semantics of each connections will be inferred from the peripheral type provided in the data exchange protocol.

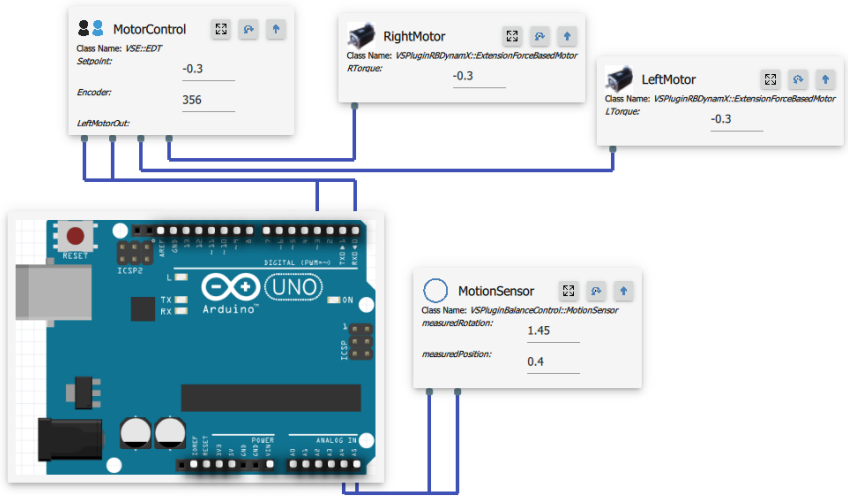

Figure 5: Modeling connections between simulation components and virtual hardware in VEROSIM.

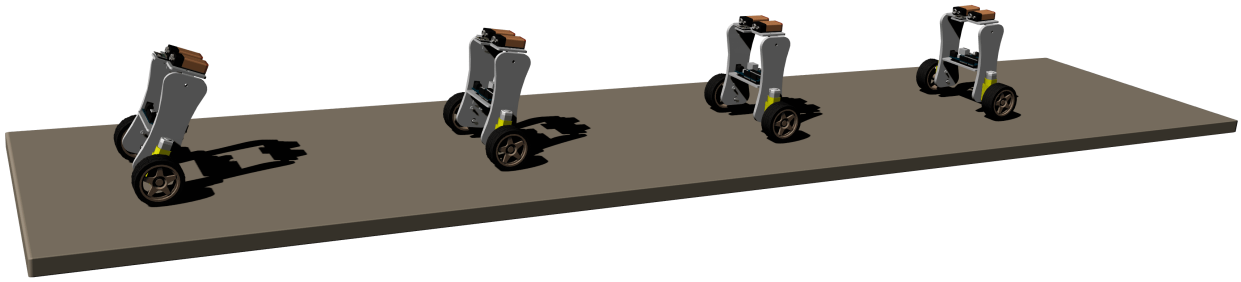

Figure 6: Rendering of the full system in different stages of a simulation in VEROSIM.

In this process, the engineer is forced to consider the limitations of embedded hardware. Limiting factors that are investigated in this case study are the resolution of peripherals and the processor's clock speed. With the VHiL approach, this design space can be explored to understand these limitations and their effect on the overall system behaviour. Two series of simulations have been performed, one with a focus on peripherals, varying their resolution and one with a focus on timing, varying the processors clock speed.

3 RESULTS

The results of two simulation series will be discussed, which explore the interaction between the embedding and the embedded system in a VHiL simulation. Both series simulate the same scenario. The robot starts in an upright pose and the robot controller is commanded to follow a straight line for 2 m and regain a stable upright pose. Figure 6 shows the robot during different stages of this scenario. In the following figures, the effective torque on the chassis resulting from both motors is plotted.

The first simulation run was performed to investigate the effect of finite peripheral resolution on the overall system behavior. The processor's clock speed is set to the original 16 MHz. The ADC has a resolution of 10bit which is mapped to the angle range $[-30\,,\,30]^\circ$ and the position range $[-5\,,\,5]$ m. The timer used to create the PWM signal has a 16 bit resolution and is mapped to the torque range $[-0.5\,,$

0.5] N m. Figure 7 illustrates the results of this simulation run. On the left, two plots representing the tilt angle of the VHiL simulation and of the reference simulation are shown. In the active starting phase, the signals are almost identical. The robot leans forward to accelerate and then leans back to regain an upright pose. Once traveling in an upright pose, very little torque is necessary to keep the robot balanced. During this phase with small signal amplitude the angle starts to oscillate in the VHiL simulation. This is due to the finite resolution of the angle measurement. The center plot shows the angle as simulated in VEROSIM along with the signal sampled by the ADC that is available to the embedded controller. Together with the resulting torque, plotted on the right, the root cause of the oscillations becomes apparent. The sampled signal suggests a large deviation from the intended trajectory. As the resulting exaggerated control signal causes the robot to pivot, the sampled signal flips to the opposing discrete value, restarting the process. Thus, the sampled signal oscillates between two discrete values at every step while the controller tries to counteract. The second simulation series was performed to investigate the clock speed's effect on the

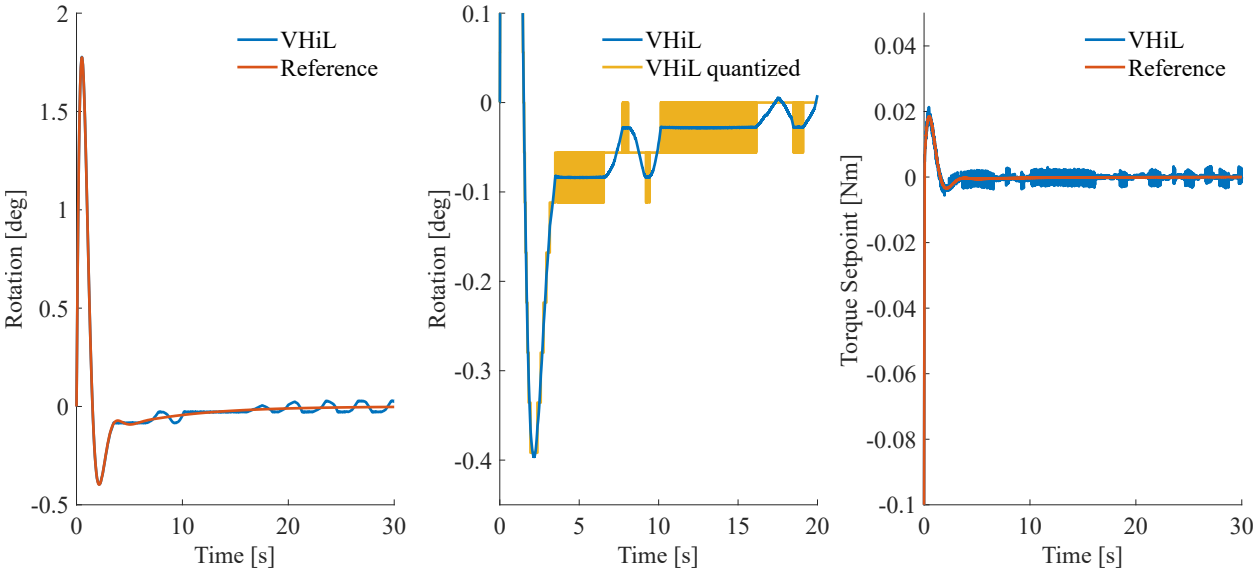

Figure 7: Simulation results highlighting the impact of peripheral sampling resolution.

overall behavior. To isolate the clock speed's effect, peripheral resolution is set to 64bit. Simulations with the clock speeds 16 000 kHz, 1600 kHz, 700 kHz, 600 kHz, 333 kHz and 332 kHz were performed. Slower processors effectively add a zero-order hold to the system, as the virtual hardware requires multiple simulation steps to compute a new torque set point. In the embedded controller, one iteration of the control loop takes roughly 2000 instructions, which means that at 600 kHz one iteration takes at least $3.\overline{3}$ ms or 3 simulation steps. The result can be observed in the torque setpoint as depicted in Figure 8. At lower clock speeds, the delayed control signal causes the system to oscillate and to eventually become unstable.

4 DISCUSSION

The results of both simulation series demonstrate, that effects stemming from interactions between embedded systems and their technical context, such as sampling and timing issues, can be observed in a VHiL simulation. Individually, these control specific results could have been obtained using other techniques, such as the z-transformation. However, VHiL is applicable to a wider range of applications, such as communication, image processing or human machine interfaces. Through the general co-simulation setup, arbitrary simulation units can be connected to realize other aspects of the individual application.

VHiL simulation is useful to bridge the gap between prototyping and actual system design. It requires integrated, more detailed models with consideration of hardware limitations, their interplay with the overall

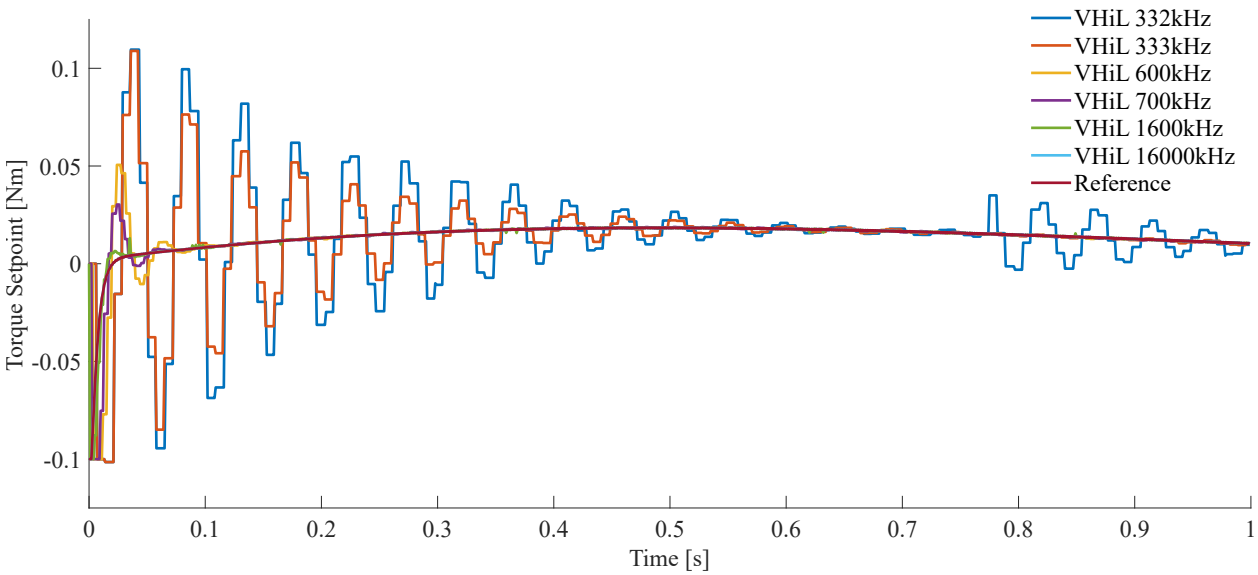

Figure 8: Influence of the virtual hardware's clock speed on the overall behavior via the torque set point.

system and code compiled for the target platform. This process forces engineers to confront these design decisions early on and yields usable code and circuit schematics that can guide the final implementation.

The presented implementation is a proof of concept. The feasibility of applying this approach to real world use cases depends largely on the availability of emulators for the target platform and their ability to virtualize hardware with sufficient accuracy for the individual use case. The presented implementation is based on the official QEMU release, which is efficient but has some known accuracy limitations, such as the difficulties in precisely estimating execution time mentioned in Section 2.2. Alternatives for specific use cases exist. (Patel et al. 2011) present MARSS, a cycle accurate virtualization of multicore x86 processors based on QEMU. (Dung et al. 2014) present another extension of QEMU that includes the simulation of caches. When performing a VHiL simulation for systems depending on low level hardware effects, it might be necessary to trade off lower performance for a more accurate virtualization platform, such as gem5.

4.1 CONCLUSION

We proposed, that many use cases of HiL simulation could be realized without real-time constraints using virtualized hardware. This approach has been classified as a hybrid co-simulation with a virtualization platform. Special challenges related to the co-simulation with virtual hardware, such as data exchange and synchronization, were analyzed and addressed. An exemplary implementation coupling the simulation framework VEROSIM with the virtualization platform QEMU was presented. This prototype was used to model and simulate a self balancing robot. Using VHiL simulation, the effects of peripheral resolution and the processor's clock speed on the robot's overall behavior were investigated. This case study demonstrated, that with VHiL simulation, hardware properties could be systematically varied and their effect on the overall behaviour could be analyzed on a general purpose computer.

4.2 FUTURE WORK

While there was a focus on providing a general, usable interface, there are still aspects that are specific to this implementation. In trying to establish QEMU as a component to be used in VHiL applications, it is intended to implement an FMI wrapper. The increasing spread of internet access makes networking an integral part of modern CPS. By including a Bluetooth chip, Figure 1 shows that networking peripherals fit well into the presented approach. Exploration of this area is subject to further research.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Regional Development Fund.

EUROPEAN UNION
Investing in our Future
European Regional
Development Fund

REFERENCES

Bellard, F. 2005. "QEMU, a Fast and Portable Dynamic Translator". In *Proceedings of the USENIX Annual Technical Conference*. April 13th-15th, 2005, Anaheim, California, 41-46.

Binkert, N., 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. 2011. "The Gem5 Simulator". *Association for Computing Machinery's Special Interest Group on Computer Architecture Computer Architecture News* 39(2):1–7.

Blochwitz, T., M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. 2012. "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In *Proceedings of the 9th International MODELICA Conference*, edited by M. Otter and D. Zimmer, 173–184. Linköping, Sweden: Linköping University Electronic Press.

Chaves, A., R. Maia, C. Belchio, R. Araujo, and G. Gouveia. 2018. "KhronoSim: A Platform for Complex Systems Simulation and Testing". In *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*, edited by A. A. N. Melendez, 131–138. Red Hook, New York: Institute of Electrical and Electronics Engineers.

Dung, T. V., I. Taniguchi, and H. Tomiyama. 2014. "Cache Simulation for Instruction Set Simulator QEMU". In *Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing*, edited by T. Dillon and M. Parashar, 441–446. Los Alamitos, California: Institute of Electrical and Electronics Engineers.

Eker, J., J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. 2003. "Taming Heterogeneity - The Ptolemy Approach". *Proceedings of the IEEE* 91(1):127–143.

Fitzgerald, J., and K. Pierce. 2014. "Co-Modelling and Co-Simulation in Embedded Systems Design". In *Collaborative Design for Embedded Systems: Co-Modelling and Co-Simulation*, edited by J. Fitzgerald, P. G. Larsen, and M. Verhoef, 15–25. Heidelberg: Springer.

Frankovský, P., L. Dominik, A. Gmiterko, I. Virgala, P. Kurylo, and O. Perminova. 2017. "Modelling of Two-Wheeled Self-Balancing Robot Driven by DC Gearmotors". *International Journal of Applied Mechanics and Engineering* 22(3):739–747.

Gomes, C., C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. 2017. "Co-simulation: State of the Art". *arXiv preprint arXiv:1702.00686*.

Kim, H., A. Wasicek, and E. A. Lee. 2019. "An Integrated Simulation Tool for Computer Architecture and Cyber-Physical Systems". In *Proceedings of the 7th International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, October*, edited by R. Chamberlain, W. Taha, and M. Törngren, Volume 11267, 83–93: Springer International Publishing.

Knörig, A., R. Wettach, and J. Cohen. 2009. "Fritzing - A Tool for Advancing Electronic Prototyping for Designers". In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, edited by N. Villar, S. Izadi, M. Fraser, S. Benford, D. Kern, and A. Sahami, 351–358. New York, New York, USA: Association for Computing Machinery Press.

Oliveira, P. R., M. Meireles, C. Maia, L. M. Pinho, G. Gouveia, and J. Esteves. 2018. "Emulation-in-the-Loop for Simulation and Testing of Real-Time Critical CPS". In *Proceedings of the IEEE Industrial Cyber-Physical Systems*, edited by A. W. Colombo, Y. Shi, J. L. M. Lastra, and S. Karnouskos, 258–263. Red Hook, New York: Institute of Electrical and Electronics Engineers Inc.

Patel, A., F. Afram, S. Chen, and K. Ghose. 2011. "MARSS: A Full System Simulator for Multicore x86 CPUs". In *Proceedings of the 48th Design Automation Conference*, edited by L. Stok, 1050–1055. New York, New York, USA: The Association for Computing Machinery.

Rossmann, J., M. Schluse, C. Schlette, and R. Waspe. 2013. "A New Approach to 3D Simulation Technology as Enabling Technology for eROBOTICS". In *1st International Simulation Tools Conference and Expo 2013*. September 18th-20th, Brussels, Belgium, 39-46.

Schroll, R., C. Grimm, and K. Waldschmidt. 2004. "HEAVEN: A Framework for the Refinement of Heterogeneous Systems". In *The Forum on Specification and Design Languages, September 13th-17th 2004, Lille, France*, edited by P. Boulet. Dordrecht, The Netherlands: Springer.

Verhoef, M., K. Pierce, C. Gamble, and J. Broenink. 2014. "Collaborative Development of Embedded Systems". In *Collaborative Design for Embedded Systems: Co-Modelling and Co-Simulation*, 3–14. Heidelberg: Springer.

Wolf, M. 2017. *Computers as Components: Principles of Embedded Computing System Design*. Amsterdam, Boston, Heidelberg, London, New York: Morgan Kaufmann.

Zhang, X., and J. F. Broenink. 2012. "A Structuring Mechanism for Embedded Control Systems Using Co-Modelling and Co-Simulation". In *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. July 28th-31st, 2012, Rome, Italy, 131-136.

AUTHOR BIOGRAPHIES

JAN REITZ is a researcher at the Institue for Man-Machine Interaction at RWTH Aachen University. He holds a B.Sc. and M.Sc. in Mechanical and Process Engineering from TU Darmstadt. His research interest is in modeling, simulation, and distributed systems. His email address is reitz@mmi.rwth-aachen.de.

ALEXANDER GUGENHEIMER is a student at RWTH Aachen University. He holds a B.Sc. in Electrical Engineering and is currently acquiring his M.Sc. in the same course of studies. His interest is in software engineering and embedded systems. His email address is alexander.gugenheimer@rwth-aachen.de.

JÜRGEN ROßMANN is a professor with the faculty of electrical engineering at RWTH Aachen University and head of the Institute for Man-Machine Interaction. After graduating from the Universities of Dortmund and Bochum, he worked as a research assistant, then as a group leader and finally as a department head at the Institute of Robotics Research at the University of Dortmund. His work in the fields of space and industrial robotics, automation technology and virtual reality has been awarded more than 15 different prizes. He is a visiting professor for robotics and computer graphics at the University of Southern California since 1998. Furthermore, he is a member of the board of Directors of the Dortmund Institute for Research and Transfer (RIF) as well as managing director of VEROSIM GmbH. His email address is rossmann@mmi.rwth-aachen.de.