Work in Progress: Extending Virtual Prototypes of Microprocessor Architectures with Accuracy Tracing

Johannes Kliemt[a] and Dietmar Fey[b]
[1]*Chair of Computer Science 3 (Computer Architecture), Friedrich-Alexander Universitaet Erlangen-Nuernberg, Martensstr. 3, Erlangen, Germany*

Keywords: vHIL, Virtual Prototypes.

Abstract: Virtual Prototypes of microprocessor architectures (VPs) extensively support the software development process with the ability to build virtual Hardware in the Loop (vHIL) test benches. A physical hardware is not necessary since the VP is also a functional simulation model, although with reduced accuracy. The actual deviation to the physical hardware in the time domain is mostly unspecified and dependent on the executed application software. This leads to issues when used in the development of software with real time requirements. The authors propose a new way of determining this inaccuracy via a trace unit integrated into the VP. Accuracy is now determined for each application software on the fly taking its individual paths through the model and not by a unrelated general set of accuracy benchmarks. A more reliable statement on the later temporal behavior on the physical hardware can therefore be given.

1 INTRODUCTION

Functional simulation models of different microprocessors architectures called Virtual Prototypes (VPs) are already actively used in science for many years now. They recently became also popular for industrial usage. Together with Synopsys, Infineon released the third generation of their Aurix microcontroller family as VP back in 2020 for their automotive Tire 1 and OEM customers (Simone Souza, 2020), two years before physical chip examples became available. VPs can be used to realize so called virtual Hardware in Loop test benches where in contrast to normal HIL simulations also the actual hardware sensing and controlling the plant model is simulated (Reitz et al., 2020). Such a vHIL was developed by the authors in cooperation with a widely known car manufacturer for evaluating purposes with the second generation Aurix microcontroller and an electric engine as plant model. During the project, the question on the accuracy of the used Aurix TC3xx VP and on the reliability of its simulation results came up. Finding an answer to this question was difficult, leading to a new idea for approaching and handling inaccuracy in VPs. This concept will be proposed in this position paper.

2 OVERVIEW

A exemplary accuracy measurement on the TC39x VP for motivation of the proposed idea will be presented in section 3, Afterwards, a short overview of the standard modeling technique in VPs for system buses is given in section 4 explaining a possible source of the encountered inaccuracy. Consequences for VP users and an idea for mitigation with automatic inaccuracy tracing is described in section 5. A summary and a outlook on future work in section 6 concludes the paper.

3 ACCURACY OF VPS

In general a VP is to 100% accurate, if its simulation produces the exact same result as its counterpart, the physical hardware. This includes the functional aspect e.g. an instruction of a compute core produces the same result as well as the non functional aspect, e.g. the result of the instruction is available at the same point in time on both platforms.

Determining the total accuracy of a VP is a non trivial and complicated task. Different parts of the VP

[a] https://orcid.org/0009-0002-6515-4724
[b] https://orcid.org/0000-0002-6077-4732

Kliemt, J. and Fey, D.
Work in Progress: Extending Virtual Prototypes of Microprocessor Architectures with Accuracy Tracing.
DOI: 10.5220/0012131800003546
In *Proceedings of the 13th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2023)*, pages 409-416
ISBN: 978-989-758-668-2; ISSN: 2184-2841

might be modeled with different accuracy and simulation speed goals in mind. These two goals are inversely correlated to each other. Usually the different components are modeled by different developer teams as well, increasing the chance for deviation in the modeled accuracy. So it can occur that the most frequently used computational units (on the TC3xx architecture the TriCores) and their direct memory peripheral devices are simulated with near hardware accuracy whereas other peripherals show significant inaccuracies.

The achievable accuracy depends therefore on the concrete software application running on the VP, with its individual paths through the system. During the work with the developed vHIL some serious time deviation at transactions over the Aurix TC39x's central SRI Bus in the VP had been observed in comparison with the physical hardware. For a more precise and qualified statement an accuracy benchmark of the SRI Bus with usage of the Aurix Direct Memory Access (DMA) module was deployed. This benchmark and the evaluation of its results will be described in the following section.

3.1 Accuracy Benchmark Setup

The benchmark consists of a series of DMA transfers between two Data Scratch-Pad RAMs (DSPRs) with an increasing number of individual moves per DMA transfer (1 to 64). A move is composed of reading one 64-Bit data word (SRI Bus data width) from the first DSPR and writing it to the second one. This test case is not detached from real world applications as its functional aspect can be found in use cases like fetching ADC conversion results or piplined data processing with multiple TriCores. Figure 1 depicts the control and data flow of the benchmark setup, which was implemented with a simple C program. The corresponding binary was then executed both on the VP and on the physical hardware.

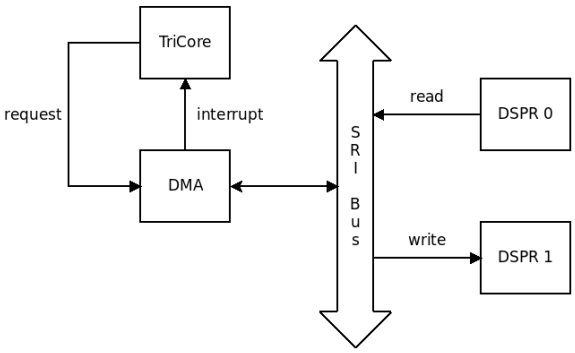

Figure 1: SRI Bus accuracy benchmark setup.

The DMA module was used as it has less hardware features possible affecting the measurement of the actual bus transfer times. One could also use one of the TriCores for transferring data between the two memories in this benchmark, but their internal architecture (Load-Store pipeline and Store Buffers) will hinder a reproducible and precise measurement.

Time was measured via the built-in performance counters in the TriCore orchestrating the DMA operation. This is the simplest but still precise time measuring method, requiring no additional external adaption effort neither to the VP nor to the physical hardware. A time stamp from the clock counter register was read immediately after initiating a DMA SW request and at the beginning of the Interrupt Service Routine (ISR) following the completed DMA transfer, measuring therefore the number of elapsed clock cycles during the DMA operation. The TriCore is idling via Nop instructions during this time. For narrowing down the measurement result as close as possible to the actual bus transactions, a second measurement was done, triggering the DMA ISR manually by SW. Subtracting this time from the first measurement results in the active time of DMA Move Engine as the only measurement overhead over the actual SRI Bus transfer times. A more accurate way of measuring the actual bus transfers on the physical hardware is not possible without special and costly (off-chip) trace devices.

The measurement for each DMA transfer was executed 100 times and rounded to the next integer to avoid any disturbance introduced by the instruction cache of the TriCore executing the measurement.

The Aurix TC3xx VP can be configured before simulation start regarding the simulated accuracy. To avoid any inaccuracy in the VP due to temporal decoupling between the TriCore (with its clock counter) and the DMA module, the quantum size was set to one clock cycle. The quantum size defines how long different components of the VP can run ahead of each other in simulation time before resynchronizing with the rest of the system. This optimization sacrifices accuracy for simulation speed and would distort the accuracy measurement. Setting it to one clock cycle for the TriCore results in an immediate reaction on the DMA interrupt as the physical hardware does.

There might still some inaccuracy in the actual implementation of the TriCore performance counters in the VP, delivering a wrong number of elapsed clock cycles in the measurement. To encounter that, the measurement setup from above was repeated using now a more complex and elaborate way for determining the timestamp values. The actual point in real time at the two time stamps had to be detected. For the VP,

this becomes a trivial task by attaching callback functions to the relevant events. They report the current point in simulated time when their events were triggered. These are the start of the actual write to the DMA modules control register initiating a SW triggered DMA transfer and the TriCore program counter register changing to the first address of the DMA ISR.

On the physical hardware one has to use the On-Chip Debug System (OCDS) module of the TC39x architecture with its (in relationship to the VP) limited tracing capabilities. For avoiding a high cost Lauterbach tracer hardware, the free of charge Infineon Multi-Core Debug Solution (MCDS) Trace Viewer software was used for capturing the relevant time information. The time of occurrence of the mentioned events from above are stored to a special trace memory inside the Aurix TC39x chip. Its size is limited, so the complete measurement run had to be divided into several steps.

For all measurements, the Synopsys TC39xB v2.4.0 VDK was used as VP and a Infineon KIT_A2G_TC397_5V_TRB BD-Step Starter Kit for its physical counter part. The clock of the TriCore executing the benchmark was configured on both platforms to 300 MHz as well as the clock for the SRI Bus. The DMA module runs internally with its maximum available clock frequency of 100 MHz.

3.2 Measurement Results

Figure 2 shows the total measured time in clock cycles for each DMA transfer on the VP (blue bars) and on the physical hardware (orange bars) for different number of moves per DMA transfer.

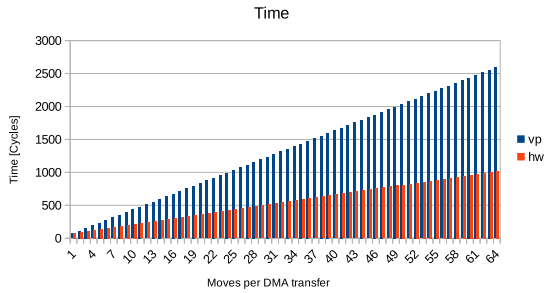

Figure 2: Measured time per DMA transfer on VP and physical hardware.

For a better view on the actual deviation, figure 3 plots the times per individual move for each DMA transfer. The numbers are the results of dividing the total numbers of clock cycles per DMA transfer from figure 2 by the number of actual DMA moves marked on the x-axis.

The asymptotic approach in both curves suggests

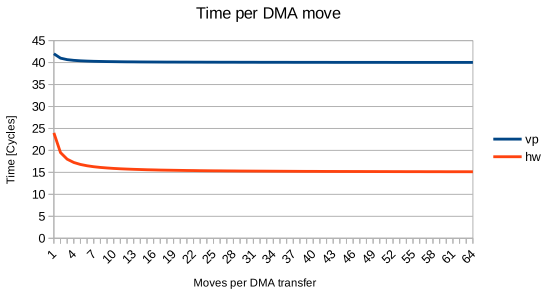

Figure 3: Measured time per DMA move on VP and physical Hardware.

that there is is still a overhead in the measurement due to the initial internal setup time of the DMA module for each new transfer. This overhead becomes more and more negligible with increasing number of DMA moves. Therefore the two curves lead to the actual DMA data transfer times over the SRI Bus. It can be seen, that each DMA move consisting of one read an one write transaction over the SRI Bus takes 15 clock cycles on the physical hardware and 40 on the VP resulting in an overestimation of the VP by 166%.

The actual internal DMA setup time can be determined from the measurement results as well. Table 1 shows exemplary the actual measured clock cycles depicted in figure 2 for DMA transfers from 1 to 5 moves per transfer.

Table 1: Measured number of clock cycles for DMA transfers with different number of moves.

# moves	1	2	3	4	5
vp	42	82	122	162	202
hw	24	39	54	69	84

Subtracting 2 respectively 9 clock cycles and dividing by the number of moves per DMA transfer leads exactly to the 40 and respectively 15 clock cycles the curves in figure 3 approach. The internal DMA setup time for each new transfer is therefore 9 clock cycles on the physical hardware. The VP models it with only 2 clock cycles underestimating the actual time by 77%.

3.3 Measurement Validation

Figure 4 plots the result of the same measurement setup from section 3.2 done utilizing now realtime time stamp values provided by the MCDS tracing module for the physical hardware and the elapsed simulated time for the VP. For better comparability, the measured times per move in each DMA transfer depicted on the y-axis have already been converted to clock cycles (through division by the clock period of $\frac{10}{3}$ ns).

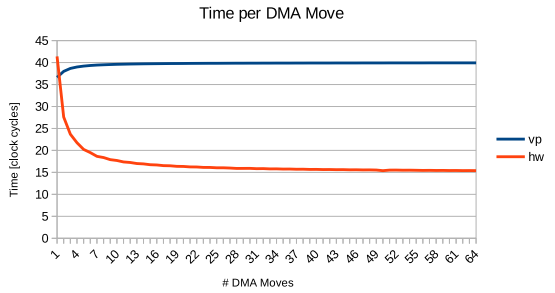

Figure 4: Measured time per DMA move on VP and physical hardware with MCDS tracing.

The time of one DMA move over the SRI Bus approaches 15 clock cycles in the physical hardware (orange curve) and 40 clock cycles in the VP (blue curve). These are the exact same number of clock cycles determined with the first measurement method. This less complicated time measurement method of counting the number of elapsed clock cycles on the TriCore and its results are therfore validated.

3.4 Actual Bus Transfer Times

The results from the last two sections do not reflect the individual data transfer times on the SRI Bus. They also include the additional operation time of the DMA model initiating the bus transfers and caching the read data from the source memory before it is written via a second SRI Bus transfer to the target memory. Due to the high insight offered by the VP, the actual starting and stopping times of the modeled bus transactions are available. Read access is modeled with 30 ns and the write access with 37 ns adding up to 67 ns SRI Bus time for one DMA move.

This times cannot be determined by measurement in the physical hardware. The measured time for one complete DMA data move extracted from the MCDS trace generated in section 3.3 varies between 40 ns and 55 ns. This is in the best case still 12 ns less than the actual bus time modeled in the VP. An insufficient timing model of the SRI Bus in the TC39x VP is therefore confirmed a second time. The exact source of this inaccuracy is indeterminable, since the Synopsys TC39xB v2.4.0 VDK is closed source.

Documentation of the VP shows that the Transaction Level Modeling (TLM 2.0) standard was used for modeling of all system buses. An overview over this modeling technique with its achievable accuracy will be given in the next section. Additionally, a possible explanation for the measured inaccuracy on the SRI Bus model of the TC39x VP will be provided.

4 TLM BUS MODELING

With the TLM bus modeling technique, all address, data and control signal lines of a bus are abstracted away through function calls between the models of the bus masters and slaves. Addresses and data are packed into a so called generic payload object and only a pointer to that object is transmitted as C++ function argument between bus participants. A second argument provides information to the current phase of the transaction. The TLM 2.0 language reference manual ((OSCI), 2009) describes two coding styles, namely ”Loosely Timed” (LT) and ”Approximately Timed” (AT) weakly related to different accuracy levels. AT is generally considered more accurate than LT, but the concrete accuracy difference is neither defined nor enforced. This also holds for the assignment of both approaches to a specific accuracy level in the interval from cycle accurate to untimed.

Cycle accurate (CA) means the VP is as accurate as its Register Transfer Level model. On this level of abstraction, the architecture is described in form of logic between register (hence Register Transfer Level) operating on system clock cycles as time base. Such a RTL model is also the starting point for chip manufacturer developing physical hardware. The aptness of TLM 2.0 for cycle accurate bus models achieving a speedup of at least one order of magnitude over RTL models was shown by (Günzel, 2011). To do so, the rules for the AT coding style had do be modified and extended. Only the relevant cycles corresponding to a phase transition in the modeled on chip bus protocol from the RTL model are then simulated, abstracting away from for a pure functional simulation unnecessary clock cycles. Those are waiting cycles for address or data lines becoming valid or for the master or slave becoming ready, which can not be skipped inside a RTL model. An example for a 2 beat write burst in a pseudo AXI protocol is depicted in figure 6 where the relevant clock cycles from the corresponding RTL simulation in figure 5 are not simulated. The (A)W_WALID signals indicate a valid write address or data when high, the (A)W_READY the time, when the slave is ready for a new address or new data. The address and data lanes of the bus are not shown here.

In clock cycle 5 neither the master nor the slave is ready for the transmission of the second burst beat, in cycle 6 and 7 the data to be transferred is not valid yet. The doted vertical lines in figure 6 mark the cycles with relevant phase transitions (control signals becoming high). They are simulated by executing a function call transmitting the relevant information from the master to slave on the bus (*fw()*) and vice

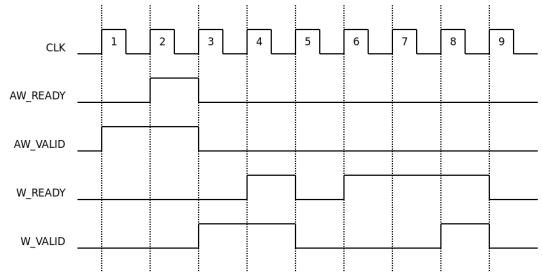

Figure 5: Burst write over pseudo AXI Bus - RTL model.

versa (*bw()*). At the transmission of the second data beat in clock cycle 8, the slave is already ready for the transmission (signal W_READY is high) so no backwards call is necessary.

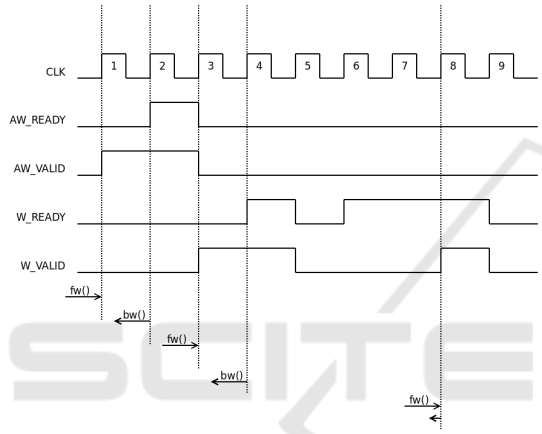

Figure 6: Burst write over pseudo AXI Bus - TLM CA model.

The TLM 2.0 AT coding style defines two basic phases, one for address and one for the data exchange and a corresponding set of rules. Although it is possible to define additional phases, a repetition of a phase is not supported. The alternative of defining numbered burst beat phases increases simulation overhead and is only possible for bus protocols with a predefined number of burst beats (Günzel, 2011). Therefore the multiple data transmissions inside a burst transfer have to be modeled with one data phase only approximating any delay between each data beat. Figure 7 locates this source of inaccuracy with a question mark.

In clock cycle 3 at the start of the data phase, the bus master has to anticipate how many clock cycles his data for the second write beat will not be valid in the future. Only the clock cycles marked with a vertical dotted line are simulated via forward (*fw()*) and backward (*bw()*) function calls analogous to the TLM cycle accurate case. Note that the backward function call at clock cycle 8 in figure 7 finishing the data phase can also occur a few clock cycles fewer or later due

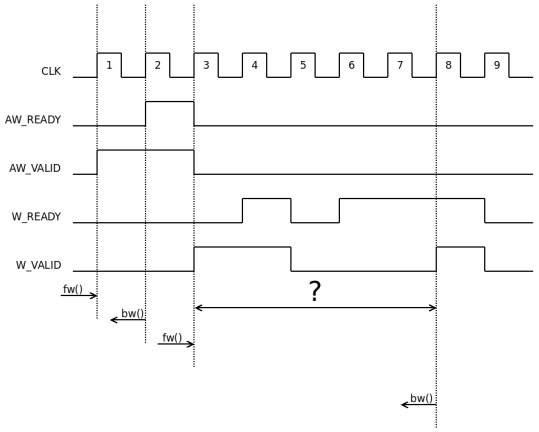

Figure 7: Burst write over pseudo AXI Bus - TLM AT model.

to the above described inaccuracy. If the data valid delay is already known at clock cycle 3, the TLM AT model can be indeed cycle accurate, too. The in section 3 measured inaccuracy on SRI Bus transactions inside Aurix TC39x VP comes very likely from badly modeled address and data phases with the TLM AT coding style.

With the TLM 2.0 LT coding style the abstraction increases one step, leaving only one phase for the whole bus transaction. This increases the possibilities for inaccuracy once again due to additional abstraction from the address phase. Figure 8 depicts the example transaction from above modeled in the LT style.

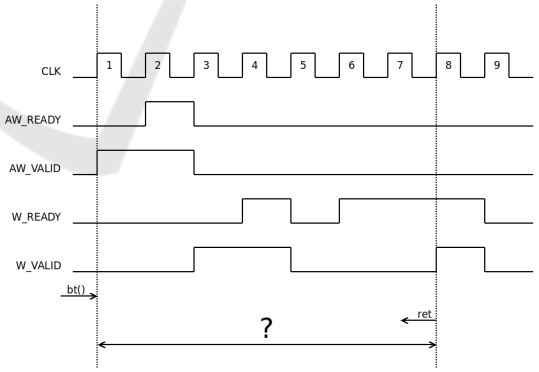

Figure 8: Burst write over pseudo AXI Bus - TLM LT model.

It consists of only one blocking transport (*bt()*) function call from the master to the slave starting at the first dotted vertical line and returning at the second one. Note that also here the return of the function call modeling the whole bus write transfer can occur fewer or later due to the time abstraction of the LT coding style.

5 HANDLING INACCURACY IN VPS

As described in (Synopsys, 2013) VPs are not a replacement for actual HIL testing but provide aid in the different stages of the V-cycle SW development model. With this in mind, one might argue, that a time inaccuracy of a few clock cycle as exemplary measured in section 3.2 does not make a VP functional incorrect and unusable for SW with real time requirements. However, for those SW applications timeliness of tasks is an additional functional property defining correctness. This rises the question, whether inaccuracy due to abstraction in the modeling makes VPs still usable here. In case of just the pure functional aspect of the individual task inside the software the answer is a definite yes. Realizing event chains with different hardware and software modules (e.g. ADC input sampling → DMA data transfer → Core calculating new PWM width → PWM module output) will definitely profit from the numerous advantages a VP offers. The actual challenge here is getting the overall timing of an event chain and the timing between its different components right. Typically there is a periodic deadline, where new output values have to be available for the whole real time system to work on properly.

In the best case, all modeled components inside a VP have a positive time inaccuracy compared to the physical hardware, giving tasks a greater slack on their deadlines on the latter one. However, if this positive time inaccuracy is to great, SW developers might encounter tasks not holding their deadlines only in the VP. This will result in unnecessary changes to the overall software design. In the exact opposite case the designed SW systems performs correct on the VP, but produces timing issues on the physical hardware. These issues cannot be detected and located insde the VP and require expensive testing and debugging on the physical hardware.

All these problem cases could be dealt with at least to a certain amount of confidence if the total inaccuracy inside the VP is known and always positive (task runtimes on VP not faster than on physical hardware). This is in most times not possible, since accuracy of VPs is usually determined by running a (high) number of benchmarks comparing there execution time between the VP and the corresponding physical hardware. An examples is the Embench Benchmark Suit [1] that was used by (Herdt et al., 2020) to evaluate the accuracy of their VP modeling the SiFive HiFive1 architecture. Each benchmark was deployed on the VP and then compared to the execution time on the open source RTL model of the corresponding SiFive HiFive1 board. The reported accuracy varies depending on the actual benchmark from -10.9% to 13.5% clock cycles (Herdt et al., 2020).

A second example is the GVSoC VP (Bruschi et al., 2021) which acts as a VP for the whole PULP processor platform [2]. The achieved average accuracy of 10% is acceptable for a VP designed for Design Space Exploration. However, accuracy was evaluated only with one complex testcase (MobileNet V1 convolutional neural network).

A general reliable statement on the VPs accuracy with this evaluation approach is not possible. As already mentioned in section 3, the accuracy highly depends on the paths that are individually followed through the whole system by an application software. For reliable approximations by the VP on the physical hardware, this accuracy comparison has to be repeated for each individual software application. Even worse, this comparison is necessary after each major change of the software in its development process.

To counter this problem the authors propose a modified approach for determining (in-) accuracy in VPs in the next section.

5.1 Proposed Solution

The key idea is to automatically determine the individually encountered inaccuracy and report it to the user of the VP. All time deviations in each modeled component have to be determined by the VP developer and than included in its source code for tracing. Costly accuracy benchmarks done by every VP user for its individual SW application can therefore be completely avoided.

For buses, this trace adds up the inaccuracy for each executed transaction over all involved bus components (e.g. arbiter) between master and slave. All sources of inaccuracy have to be located and quantified at least in well defined intervals. For the localization of possible inaccuracy in bus modeling via TLM, a short overview was already given in section 4. The challenging part is the quantification of the modeling error. There are two kinds of sources for these errors, namely static and a dynamic ones.

Static errors occur independently of any dynamic events on the bus peripherals. An example is a architectural constraint in the master that introduce a static number of wait cycles, before the data for a new burst beat can be outputted on the bus. A modeling error does not need any tracing here, since a refinement of

[1] https://www.embench.org/

[2] https://pulp-platform.org/

the TLM model is trivial and comes with no simulation overhead. The master computes the number of burst beats multiplied by the static delay time and notifies the slave with the forward call (e.g see clock cycle 3 in figure 7). The slave can account for that delay by delaying his backward call accordingly. If a refinement is not possible (e.g. due to a closed source VP) tracing the resulting static inaccuracy for a bus transfer is a suitable alternative. However, if at the begin of the data phase the data delay is not known yet for each burst beat, inaccuracy will occur in the TLM transaction. To reflect that dynamically caused inaccuracy in the proposed trace methodology, it logs the minimum and maximum possible delays for that transaction. These delays have to be (analytically) determined once for each master and each slave attached to the bus. Doing that for a specific VP as proof of concept will be up to future work.

For better explanation of the proposed accuracy tracing methodology, a minimal hypothetical example hardware is depicted in figure 9. It consists of a compute core (Core_0), that is connected to two memories (Memory_0 and Memory_1) via a common bus. Let the access time for Memory_0 be constant 3 clock cycles for a read operation and 5 clock cycles for a write operation. Both access types have been somehow modeled with a 2 clock cycle TLM transaction in the corresponding VP, see dotted arrow in figure 9. The same was done with the TLM modeled accesses to Memory_1, this time with 5 and respectively 6 clock cycles. Its physical counterpart has a dynamic access time for read operations between 3 and 7 clock cycles and for write operations between 5 and 9 clock cycles.

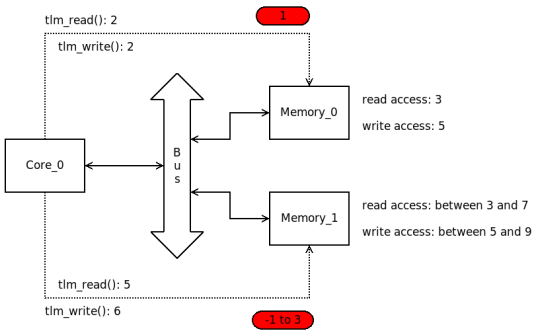

Figure 9: Minimal hypothetical example hardware for accuracy tracing of TLM modeled buses.

Consider now a load operation from Memory_0 by Core_0, that is followed by a store operation to Memory_1. This will take 3 + 5 to 9 clock cycles on the physical hardware. The TLM transactions inside the VP approximate it by 2 and 6 clock cycles. Deploying a time measurement in the VP results in a time of 8 clock cycles for the whole load store sequence. However, this measurement result contains a hidden inaccuracy of 1 to 4 clock cycles. To get hold of this, the accuracy trace can be consulted. It logged for the TLM read transaction a deviation of 1 clock cycle and for the write transaction a minimal deviation of -1 clock cycle and a maximal deviation of 3 (see red ovals in figure 9). With this additional information, the actual execution time of the load store sequence on the physical hardware can now be determined in at least a certain time interval by measurements in the VP. A software developer, that uses latter one for time performance evaluation, has therefore now a more reliable decision base for making changes to his code.

6 SUMMARY & FUTURE WORK

This position paper proposed the so far not yet implemented idea of VPs, tracing the their individual inaccuracy as an additional simulation output. The idea was motivated via an exemplary accuracy measurement done on a industrial grade VP of the Infineon TC39x microcontroller family. A short overview of the bus modeling technique with TLM 2.0 and its achievable accuracy was given, narrowing down the most likely reason for the detected deviations. The last section of this paper described shortly the consequences for VP users and a way for mitigation by introducing the idea of VPs reporting the individually encountered overall inaccuracy to its user.

Future work will be the actual implementation of the proposed idea, expanding this concept from bus modeling also to the other important components inside a VP. Examples are central compute units with their piplined instruction execution and caches, requiring additional modeling techniques with their specific sources of inaccuracy.

REFERENCES

Bruschi, N., Haugou, G., Tagliavini, G., Conti, F., Benini, L., and Rossi, D. (2021). Gvsoc: A highly configurable, fast and accurate full-platform simulator for risc-v based iot processors. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 409–416.

Günzel, R. (2011). *Taktgenaue Bus-Simulation mit der Transaction-Level-Modellierung*. PhD thesis.

Herdt, V., Große, D., and Drechsler, R. (2020). Fast and accurate performance evaluation for risc-v using virtual prototypes. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 618–621.

(OSCI), O. S. I. (2009). Osci tlm-2.0 language reference manual. https://www.accellera.org/images/downloads/standards/systemc/TLM_2_0_LRM.pdf. accessed 04.04.2023.

Reitz, J., Gugenheimer, A., and Roßmann, J. (2020). Virtual hardware in the loop: Hybrid simulation of dynamic systems with a virtualization platform. In *2020 Winter Simulation Conference (WSC)*, pages 1027–1038.

Simone Souza, Synopsys, I. (2020). Synopsys expands portfolio of automotive vdks with support for infineon's aurix tc4xx 32-bit microcontroller family. https://news.synopsys.com/2020-10-27-Synopsys-Expands-Portfolio-of-Automotive-VDKs-with-Support-for-Infineons-AURIX-TC4xx-32-bit-Microcontroller-Family. accessed 04.04.2023.

Synopsys (2013). Virtual hardware "in-the-loop": Earlier testing for automotive applications. https://www.synopsys.com/cgi-bin/proto/pdfdla/docsdl/virtual_hardware_wp.pdf?file=virtual_hardware_wp.pdf. accessed 30.03.2023.