Increased Accuracy through Noise Injection in Abstract RTOS Simulation

Henning Zabel, Wolfgang Mueller
Universität Paderborn, C-LAB
Fürstenallee 11, D-33102 Paderborn, Germany

***Abstract*—Today, mobile and embedded real-time systems have to cope with the migration and allocation of multiple software tasks running on top of a real-time operating system (RTOS) residing on one or multiple system processors. Abstract RTOS simulations and timing analysis applies for fast and early estimation to configure it towards the individual needs of the application and environment. In this context, a high accuracy of the simulation compared to an instruction set simulation (ISS) is of key importance. In this paper, we investigate the accuracy of abstract RTOS simulation and compare it to ISS and the behavior of the physical system. We show that we can reach an increased accuracy of the simulation when we inject noise into the time model. Our results indicate that it is sufficient to inject uniformly distributed random time values to the RTOS real-time clock.**

I. Introduction

Today, RTOS timing analysis is mainly performed by Worst Case Execution Time (WCET) analysis, Worst Case Response Time (WCRT) analysis [1], and by Instruction Set Simulation (ISS) [2]. Additionally, logic analyzers, tracing hardware, and specialized trace boxes come into application.

Traditionally, ISS is mainly used for functional and performance analysis based on a specific target processor and operating system. ISS mainly performs the emulation of assembly code at the machine code level covering the execution of the complete application plus operating system by accurately representing and simulating values of single variables and all registers. As such, ISS requires the complete software executable including the real RTOS to be available in binary form. Instructions set simulators are highly accurate and already reach considerable speeds. However, their simulation times are typically still insufficient for early estimation and rapid prototyping. However, fast simulation is of utmost importance in early estimation in order to scale different system parameters.

On the other hand, simulations based on abstract RTOS models give promising simulation speeds. Schirner et al. [3], for instance, report significant 1000x speed-ups. However, they come with a reduced accuracy of approx. 7-8 % in average compared to ISS. The resulting simulation errors are mainly due to the abstraction since not all execution paths are considered. Additionally, instructions are data–dependent thus they may vary in their execution time so that they can hardly be represented by fixed mean or worst case execution times. Inaccurately modeled critical sections may have further impact to the starting time and duration of individual tasks and, finally, we can identify small variations in the frequency of the crystal clock. In some cases, the combination of these effects may give completely different scheduling sequences, which finally result in more or less significant errors in the abstract simulation.

This article investigates the accuracy of abstract RTOS simulations for timing analysis compared to measurements taken by ISS and by a logic analyzer and their coverage by the abstract RTOS simulation. To increase accuracy, we investigate noise injection to abstract RTOS models through three different timing parameters: (i) execution time of basic blocks, (ii) communication delays, and (iii) the frequency of the RTOS real-time clock, which is typically implemented by an RTOS tick interrupt service routine (ISR). For all parameters we apply uniformly distributed random values within a specific timing interval. The main goal of our investigations is the compensation of possible time shifts in the scheduling sequence by the introduction of noise.

We present our studies on the impact of noise injection to different combinations of those parameters and finally find very clear indications for improved accuracy and the increased coverage of the measurements by noise injection to the RTOS real-time clock. We can also show how to meaningfully limit the noise to a small interval, which in turn decreases simulation time by smaller stimuli sets.

The remainder of this article is organized as follows. The next chapter discusses related works. Thereafter, we introduce basic principles of abstract RTOS simulation and develop a metrics for the coverage of measurements by abstract RTOS simulation before we investigate (i) the accuracy of a simulation w.r.t. the physical system and (ii) the coverage by the simulation. Finally, the article closes with a conclusion.

II. Related Work

Timing informations for schedulability analysis are usually derived from a Worst Case Execution Timing Analysis (WCET). Currently available tools for static analysis like aiT (AbsInt) and SymTA/S (SymtaVision) support static WCET and response time analysis, which are based on the estimated upper bound over all possible execution times of a task. An alternative approach is to model task execution times as a stochastic distribution. Manolache [4] introduces an analysis based on stochastic execution time models to retrieve the expected deadline miss ratio for a given task set.

Instruction Set Simulation (ISS) simulation usually comes with a slow execution so that no detailed evaluations and analysis like the evaluation of different scheduling strategies can be efficiently performed. Therefore, several research groups are working on C-based abstract canonical RTOS models which give simulation speed increase 500x-1000x compared to ISS [5], [6].

Desmet et al. [7] provide one of the early approaches to RTOS generation with SystemC. Later, several groups like [5], Yoo et al. [8], and [9] published approaches for fast SystemC- and SpecC-based RTOS simulation. They all support slightly different RTOS-relevant properties like POSIX compatibility and interrupts. Most of them are based on the simulation of static time annotation of basic blocks of tasks, where timing information is retrieved and backannotated before the simulation and introduce improvements compared to ISS.

Posadas et al. [10] have published several articles on RTOS simulation. They introduce concepts of their SystemC RTOS library PERFidiX, which covers approximately 70% of the POSIX standard. Segments of software threads are annotated by time estimations, which are estimated at run-time by overloading C/C++ operators and depends on the simulated target platform. They report improvements in simulation speed of more than 142x compared to ISS.

Huss and Klaus [11] apply the Gumbel distribution to compute execution times of tasks in their SystemC RTOS model. They evaluated different scheduling strategies for the case study of a mobile robot. This is one of the very few approaches, which tries to combine SystemC RTOS simulation with statistical methods. Unfortunately, no detailed numbers on the accuracy of their approach is available.

There are very few approaches which compare abstract RTOS simulation to execution times on physical hardware. Here, Hwang [12] investigates a cycle approximate retargetable performance estimation to generate timed SystemC models at transaction level. Time information is derived from the instruction set of the Low Level Virtual Machine (LLVM) and from the processing unit model (PUM) via a CDFG (Control-Dataflow Graph). Comparison to physical hardware showed an average error of 8% in the number of estimated cycles.

Our approach combines abstract RTOS simulation with statistical methods. To increase accuracy, we have implemented a noise injection for different time parameters. We show that a uniform distribution of noise to a specific parameter can lead to significant improvements in the accuracy and increase the coverage of measurements by our abstract RTOS simulation. For wider conclusions, we compare accuracy and coverage of our results to ISS and to physical hardware.

III. Abstract RTOS Modeling and Simulation

Modeling of embedded and SoC systems typically starts with an architectural model, which is further refined into hardware and software. In hardware-dependent software refinement, there are several approaches to develop high-level models. [3], for instance, divides abstraction levels into Application Level and Task Level before arriving at Firmware, Transaction Level Models (TLM) and cycle- and pin-accurate Bus Functional Models (BFM). At task level abstract Operating System (OS) models are typically introduced, which cover software tasks, interrupts, and interprocess communication (IPC). RTOS abstraction is applied for fast simulation in order to analyze different system parameters like scheduling strategy.

In more details, task level models are composed of the following entities:

- **I/O Access.** This covers the direct read/write to registers and other memory resources.
- **Driver and Communication Stacks.** Drivers and communication stacks typically implement routines for fast I/O access. This can be by the means of interrupts. This access can be through an Hardware Abstraction Layer (HAL) for register, access, and functional shielding.
- **Operating System.** An operating system is introduced when different hardware resources have to be shared by multiple software tasks. The operating system, for instance, provides different synchronization and communication schemes as well as a task scheduler.

Task level abstraction typically replaces the I/0 access and the HAL by a functional layer in order to analyze different aspects at that level. This functional interface has to be carefully defined so that no significant modifications have to be applied on the existing software. In the case that software functions are removed by such an abstraction, they have to be adequately replaced by its constant execution time, i.e., mean or worst case execution time.

Abstract RTOS simulation at the task level is based on partitioning of the application into hardware components and software tasks, including Interrupt Service Routines (ISR). Tasks and ISRs are further divided into a sequence of time-annotated software segments. Segments are usually defined at the basic block level, though it is possible to partition into more coarse- or fine-grain segments (see Figure 1).

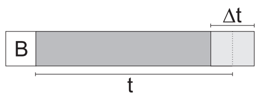

Fig. 1. Time annotated software segment

In abstract RTOS simulation by SystemC [9], this is typically modeled by the execution of the basic block (B) or its abstraction followed by a wait statement for modeling its fixed time delay (t). The time is retrieved from measurements or from ISS which typically introduce a small time error (Δt). As SystemC, like other C-based system description languages, is based on a non-preemptive simulation kernel this modeling style gives simulation inaccuracies in start times of tasks and ISRs when preemptive behavior, like an interrupt, is simulated. To overcome this drawback several extensions and libraries like [10] and [9] were introduced. However, due to the individual application a certain inaccuracy remains when abstract RTOS simulation is compared to ISS or to the physical system execution.

Figure 2 shows a simplified view of testbeds comparing the I/O behavior of a combined task/transaction level with an ISS and a physical system, respectively. The task level model runs the applications and the device drivers on an abstract RTOS (aRTOS) and on a TLM I/O model, which both run

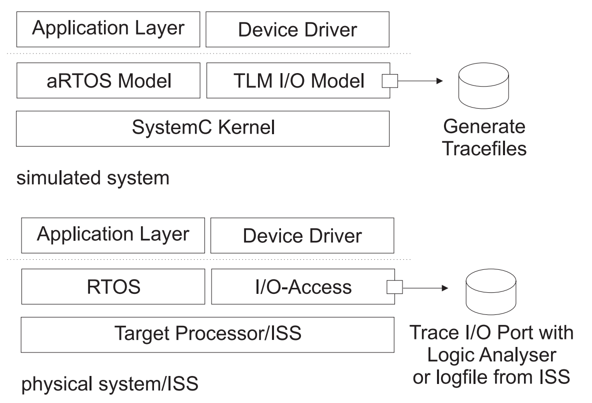

Fig. 2. Simulated vs. Physical System

on a SystemC kernel. We can easily generate trace files from the TLM model in order to run a further analysis. The ISS and physical system is sketched by application layer, device driver, RTOS, and the I/O access, which reside on the ISS or processor. The I/O access of the system can be traced by a logic analyzer for further measurements or a logfile can be generated. The execution times for back annotation to more abstract levels can be estimated by disassembling the binary for the target [13].

To avoid too many re-spins in the design cycle, main considerations concern the accuracy of the analysis results at task level with respect to the behavior of the ISS or the final physical system, which is subject of our remaining investigations.

Despite of accurate simulation models at task level, most executation times of segments cannot be exactly determined. This is due to specific individual execution paths and the data-dependent variances of instructions, which both have to be determined after optimization and compilation into assembler code. Small variations in the crystal clock and inaccurate models of critical sections have additional impact on their execution. In total, all these effects give more or less significant deviations in the start times and durations of tasks. In many cases, the total error may give a different task schedule compared to the execution by ISS or on the physical hardware.

To adequately cover all these effects, we apply a statistical method and inject a uniformly distributed random value to the different time values, which are given by the estimated execution times. This, of course, implies that an additional number of stimuli has to be applied so that we also have to conclude with findings how to limit the width of the noise intervals.

In our abstract RTOS simulation, we evaluate timing variances to the execution times of

1) basic blocks,
2) communication, and
3) the real-time clock

Abstract RTOS simulation is typically based of the simulation of basic blocks of each task annotated by their mean or worst case execution time. Time annotations for communication can be due to bus transmissions, buffer delays etc. Additionally, physical variations of the hardware clock or clocks may significantly determine variations in the system behavior, which has to be accurately covered during simulation. The notion of real-time clock refers to the RTOS tick interrupt that triggers all RTOS activities. The real-time clock depends on the clock crystal that resides on the board, often denoted as XTAL.

IV. Simulation Coverage

To achieve high accuracy at task level, the abstract RTOS simulation has to cover the final physical system behavior as much as possible. More precisely, this means that as many scheduling sequences of the simulation have to be as close as possible to the sequences of later measurement of the ISS or the physical system. In the ideal case, the set of scheduling sequences equals the set of sequences in the measurement in number and values, or just differ by a ϵ, which has to be kept as small as possible. To more formally cover these problem, we introduce some definitions.

Definition IV.1. *Let $J = \{J_1, J_2, .., J_n\}$ be the set of Tasks and ISRs, and $JS = \{RUNNING, READY, WAITING\}$ the set of possible states of an element of J. $\#J = n$ gives the number of elements of J. Then $se = (s, i, t)$ defines a scheduling event se at time point $t \in \mathbb{R}$, where J_i switches to state $s \in JS$.*

Note that an ISR is limited to $RUNNING$ and $WAITING$, where the latter denotes the state when waiting on an interrupt.

Definition IV.2. *We further define SE as a finite sequence of scheduling events $\langle se_1, se_2, .., se_n\rangle$, where for each pair $se_j = (s_j, i_j, t_j)$ and $se_k = (s_k, i_k, t_k)$ of that sequence $t_j \leq t_k$ is true if $j < k$.*

In the remainder, we have to compare different sequences SE^k where $k \in \{1..m\}$ and m is the number of elements of the set of all possible scheduling sequences $M(SE)$, i.e., $\#M(SE) = m$.

A scheduling sequence is related to the execution sequence of segments within a specific period. We denote the corresponding execution path by σ. Note, that the set of all possible paths is finite since the number of segments is finite.

Definition IV.3. *We define two scheduling sequences SE^k and SE^l with path σ and $k, l \in \mathbb{N}$ as ϵ-similar and write $SE^k \overset{\epsilon}{\sim} SE^l$, if:*

1) *$\#SE^k = \#SE^l = n$ and*
2) *$\forall p \in \{1..n\}$: $s^k_p = s^l_p$ and $i^k_p = i^l_p$ and*
3) *$\forall q \in \{2..n\} : |t^k_q - t^l_q - t_{offset}| \leq \epsilon$ with $t_{offset} = t^k_1 - t^l_1$.*

In other words, two scheduling sequences are ϵ-similar, if they refer to the same path σ, they have the same length, their sequences of states and tasks/ISRs are equal, and their time values only differ by ϵ with respect to a given initial offset t_{offset}. As a consequence, in the case of $\epsilon = 0$, two sequences then just have a t_{offset} time shift.

In our work, we consider scheduling sequences from simulation (SIM) and measurement (MEA). Both sets are subsets of all possible sequences $SIM, MEA \subseteq M(SE)$.

We are especially interested in the number of elements of $MEA^{\epsilon}(SIM) = \{se \in MEA | \exists se_a \in SIM : se \overset{\epsilon}{\sim} se_a\}$, which defines the set of measured sequences that are ϵ-similar to sequences in a given simulation. Thus, $\#MEA^{\epsilon}(SIM)/\#MEA$ defines the ratio of a set of measured sequences that are ϵ-similar to a set of simulated sequences divided by all measured sequences. In the remainder of this article we abbreviate this by $\rho_{SIM}^{MEA}(\epsilon)$. In the ideal case, $\rho_{SIM}^{MEA}(\epsilon)$ is close to 1, which means that the simulation covers all measurement sequences within an ϵ.

In the next section, we take traces from the transaction level simulation SIM and compare them to traces of the measurements MEA (see Figure 2). Traces refer to scheduling sequences SE^k in the above definitions.

V. Accuracy and Coverage Analysis

For early accurate estimations by abstract RTOS simulation, we are interested in two results

1) how to improve simulation accuracy by keeping the simulation speed
2) how to improve the simulation model to get the best coverage of the measurements

For this, we apply noise injection at specific points in the model and evaluate the relevance of different parameters to the accuracy and the coverage of the simulation. Our evaluations are based on two representative case studies with different properties and on a set of configurations of their parameters[1]. Case study 1 is a distributed light controller with heat sensors. The system is composed of 5 processors and implements communication over serial channels. Each processor executes 9 tasks and 5 ISRs. The RTOS handles sporadic events with static priorities and without preemption. Case study 2 implements a single processor CPU benchmark executing three periodic tasks, which are scheduled with Rate Monotonic with preemption. Two tasks have an almost constant execution time where the execution time of the third one varies significantly. In both cases, the execution of tasks are triggered by the real-time clock, which is implemented by an ISR, i.e., the RTOS tick interrupt. Though measurements of case study 1 are taken by a logic analyzer and measurements of case study 2 are taken by ISS they give the same direction.

The basic setup of the testbed for both case studies is outlined by Figure 2. Here, the scheduling events $se = (s, i, t)$ are the context switches of the RTOS, which are written to an I/O port, where only changes of the task state $s \in JS$ and task id i for J_i are recorded. The time point $t \in \mathbb{R}$ is either given by the current simulation time or the sampling time of the logic analyzer. Scheduling events within specific time intervals are combined to scheduling sequences SQ. These time intervals corresponds to the hyperperiods of the task set. The start and end of such an interval is determined by a stimuli generator.

We applied 3 different timing parameters for noise injection to both case studies: (i) communication time (v_{comm}), (ii) time annotation of basic blocks (v_{block}), (iii) and the real-time clock (v_{clock}), i.e., the time point for the execution of the RTOS tick interrupt. Applying noise injection to those parameters means, that we randomly changed their time values or delays in each simulation cycle within a fixed time interval. For this we identify the time interval by their upper bound and call them p_{comm}, p_{block}, p_{clock} due to the corresponding parameters. Upper bounds are given in terms of execution cycles.

[1]In both cases the hardware was the AT90CAN128 8Bit RISC Processor

In order to reach uniform distributions of the injected noise there is a direct correlation of high values of v_{comm} to the number of required stimuli or, in other words, the smaller the interval the less stimuli are required and the shorter the simulation time. That also means, that investigations on wider intervals are of theoretical interest only.

In a first step, we address our first main issue and evaluate the influence of our parameters to the accuracy of the simulation, i.e., to Δt between the execution time in the simulation and execution time in the measurement. To evaluate the influences between timing parameters p_{clock}, p_{block} and p_{comm}, we investigated different configurations, which can be found in the following table. An x denotes an applied noise injection on the respective parameter.

configuration	1	2	3	4	5	6	7
communication time	x	x	x	-	x	-	-
basic blocks annotation	x	x	-	x	-	x	-
real-time clock	-	x	x	x	-	-	x

All our results clearly indicated that no significant effects can be identified with $p_{block} \neq 0$ so that could set $p_{block} = 0$. This means that configurations 1, 2, 4, and 6 could be skipped leaving configurations 3, 5, and 7. We can also easily see that configurations 5 and 7 are all special cases of configuration 3 and can be evaluated by different settings of p_{clock} and p_{comm} in configuration 3, which was therfore taken for our main studies.

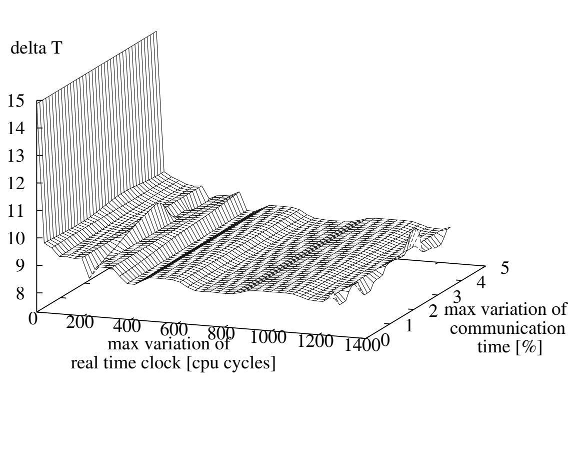

Fig. 3. Simulation Acuracy: p_{clock} vs. p_{comm}

Task	Mea-sured	Simulation w/o noise			Simulation w/ 200 cycles noise		
	t[μs]	t[μs]	Cyc.	Δt [%]	t[μs]	Cyc.	Δt [%]
1	160.4	168.8	134	5.2	163.5	50	1.9
2	10.4	11.5	18	10.6	11.6	21	12.4
3	38.3	41.1	45	7.3	32.4	-94	-15
4	3.2	3.2	0	0	3.3	2	3.7
5	13.7	23.0	149	67.8	15.1	22.4	10.7
				18.18			8.74

Task	Mea-sured	Simulation w/o noise			Simulation w/ 200 cycles noise		
	t[μs]	t[μs]	Cyc.	Δt [%]	t[μs]	Cyc.	Δt [%]
1	246.6	246.5	-2	0	246.5	-2	0
2	172.9	180.5	122	4.4	180.5	122	4.4
3	1551.5	1591.6	642	2.6	1580.8	469	1.9
RTC	10.7	10.9	4	1.9	10.9	4	1.9
				2.2			2.1

TABLE I
AVERAGE EXECUTION TIME OF TASKS W/ AND W/O NOISE INJECTION AND MEASURMENTS (CASE STUDY 1 AND 2 UNDER CONFIGURATION 7)

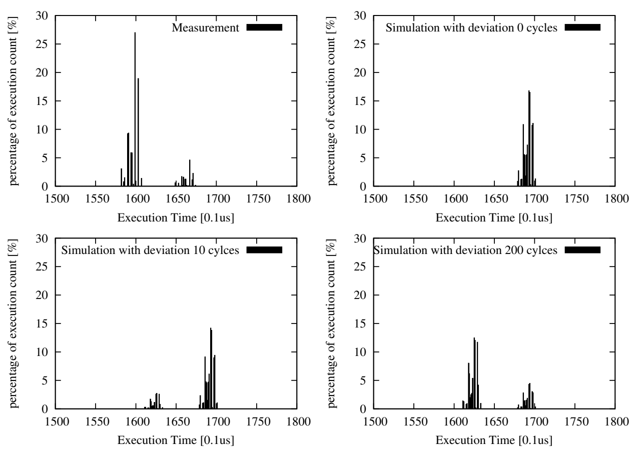

Fig. 4. Message handling task of case study 1: Measured and simulated with noise injection 0, 10, and 200 CPU cylces

A. Simulation Accuracy

Figure 3 is one example which compares different variation intervals of p_{clock} and p_{comm} on the horizontal plane to different values in accuracy in the vertical for case study 1 with configuration 3. Like in this example, all our studies showed that variations of p_{comm} have no visible impact on the accuracy in this configuration. In all cases, the influence of p_{comm} to the accuracy was 5x less than the impact of p_{clock}. As a conclusion, p_{comm} and p_{block} can be set to 0 without a major loss of accuracy. Therefore, we focused our investigations to variations of the real-time clock $v_{clock} \leq p_{clock}$. In this context, our studies with p_{clock} have shown, that beyond an upper bound of 500 clock cycles no further relevant improvement can be made. As an example, Figure 3 with numbers from case study 1 indicates, that a small increase in the interval upper bound p_{clock} gives a significant increase of simulation accuracy at all values of p_{comm}. In more details, it clearly indicates that noise intervals with $100\ clock\ cycles \leq p_{clock} \leq 500\ clock\ cycles$ gives a very high accuracy and still keeps a low number of stimuli for noise injection. Case study 2 gave a similar effect though the initial error was not that significant.

In conclusion for the simulation accuracy on both case studies under all configurations, we can summarize that a uniformly distributed noise injection to the real-time clock, i.e., to the RTOS tick interrupt, results in the highest simulation accuracy in the interval $[100, 500]$. Injections to the basic block are neglectable and even variations of the communication time have no significant impact and thus can be skipped. Therefore, their time values can be fixed to the given mean or worst case execution times so that we can focus on p_{clock}, i.e., configuration 7 in the remainder of this article.

Table I gives more details of both case studies for configuration 7. It compares measured execution times of individual tasks and their simulated execution times with and without noise injection ($p_{clock} = 0$ and $p_{clock} = 100$) including their relationship to simulation cycles. The first case study shows a significant improvement from 18.18% to 8.74%. The second case study shows an increased accuracy by 0.1%.

Our further evaluations addressed response times and message handling. Here, first studies indicated an maximum reduction for the accuracy of response times to approx. 20% in average. Figure 4 shows a set of histograms with the execution times for the message handling task of case study 1. The upper left gives the real numbers of the physical hardware. The other figures give more details on abstract RTOS simulation. In more details, the upper right with $p_{clock} = 0$ (w/o noise injection) lack multiple peaks at ca. $159\mu s$ which are the consequences of missing scheduling sequences in the abstract simulation. In that figure just some peaks at $169\mu s$ are recorded. Their displacement from 165 to 169 with respect to the measurement is due to a small overestimation in the annotated times. The lower two figures with $p_{clock} = 10$ and $p_{clock} = 200$ clearly indicate the advantage of noise injection within certain limits, where for $p_{clock} = 200$ (lower right) the peaks closely match the hardware execution.

B. Coverage of Measurement by Simulation

Let us now consider the coverage of the measurement by the simulation for configuration 7 with noise injection.

Recall that we have defined the ratio $\rho^{MEA}_{SIM}(\epsilon) = \#MEA^{\epsilon}(SIM)/\#MEA$ for our simulation coverage. In our studies, we evaluated $\rho^{MEA}_{SIM}(\epsilon)$ for the different intervals and investigate them for increasing ϵ. Figure 5 shows the coverage of the simulation of case study 1 with different intervals p_{block} for configuration 1 on the left. It shows $\rho^{MEA}_{SIM}(\epsilon)$ in % on the y-axis with an increasing ϵ given in $0.1\mu s$ on the x-axis. The four overlapping waveforms for the different noise injection intervals to the basic blocks $p_{block} = \{0, 10, 50, 200\}$ show a very low coverage of the measurement in all four cases.

Like in the previous accuracy studies, our evaluations also indicate here that variations of p_{block} and p_{comm} both give not a sufficient coverage for both case studies. We found that configuration 7, which only injects noise to the real-time clock, is far ahead of the other options for both case studies (see 5 center and right). With $\epsilon = 200$, the coverage could be increased from 2% up to 50% for case study 1. This basically corresponds to the time for one context switch. For case study 2, the waveforms show a coverage of over 95% for $\epsilon = 200$.

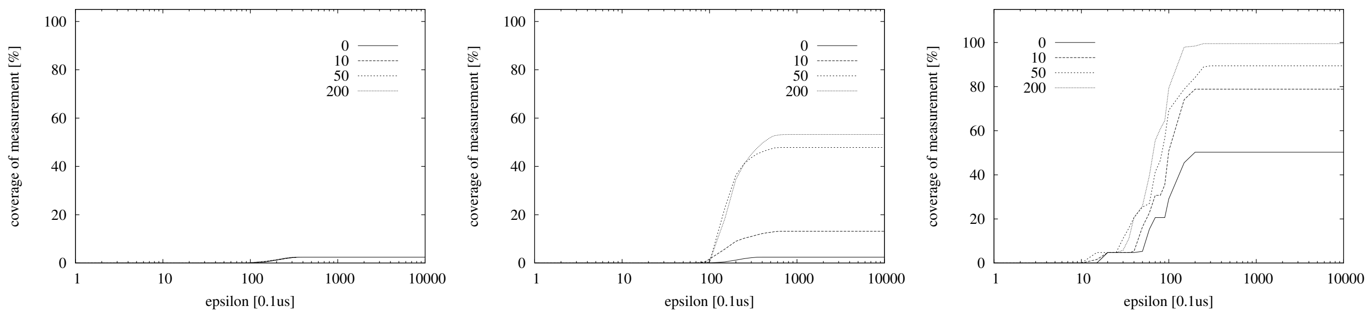

Fig. 5. Coverage by simulation: (left) case study 1/configuration 1, (center) case study 1/configuration 7 (right) case study 2/configuration 7

The waveforms for case study 1 and 2 in the center and on right additionally indicate a significant lower coverage of just 50% and $< 5\%$ when no noise injection is applied, i.e., when $p_{clock} = 0$ (solid line).

VI. Conclusion

This article investigated noise injection to increase the accuracy of abstract RTOS simulations and its coverage of measurements of real execution times. For this, we investigated variation of different parameters for time annotations of basic blocks, communication times, and the frequency of the RTOS real-time clock. We studied the impact of noise injection to different combinations of those parameters and finally found that noise injection to the RTOS real-time clock gives a significantly improved accuracy and increased abstract RTOS simulation coverage. We also could show that the clock cycle interval $[100, 500]$ is already sufficient for the highest accuracy and for a high coverage by the abstract RTOS simulation. This is a promising result since this also limits the number of traces to match the stimuli and thus does not come with significantly higher runtimes in simulation when introducing noise injection. Our studies are based on a considerably high number of traces. We applied approx. 50,000 traces for the abstract RTOS simulation and 5,000 CPU traces for ISS in the context of case study 1. For case study 2, we applied 45,000 for RTOS simulation and 1,000 for the physical system, respectively.

Our studies clearly indicate that noise injection to the real-time clock increases the coverage of the simulation and decreases the deviation from the mean execution times of task at the same time. Our internal investigations have shown that this effect is due the number of simulated interruptions and the simulation error mainly comes from inaccurate estimations of execution times. Due to the nature of those estimations they cannot be really avoided. We found that noise injection to the real-time clock introduces variations of such interruptions that also results in an increase of the simulation coverage. Additionally, noise injection also resolves the effect of variances in start times of tasks, which are due to small variances in the crystal clock. Though the latter effect seems to be less important, our experience with abstract RTOS simulation is that small variations may easily have observable impact in several cases, i.e., different task schedules. That effect may become more important when we have to deal with distributed systems based on multiple CPUs and chipsets.

VII. Acknowledgments

The work described herein is partly funded by the DFG through the Sonderforschungsbereich 614, the German Ministry for Education and Research (BMBF) through the ITEA2 project TIMMO (01IS07002), and by the EU through COCONUT (FP7-ICT-3217069).

References

[1] R. Ernst, S. Schliecker, A. Hamann, and R. Racu, “Formal methods for system level performance analysis and optimization,” in *DVCon’08: Design and Verification Conference and Exhibitation*, San Jose, CA, 2008.

[2] J. Wagner and R. Leupers, “A fast simulator and debugger for a network processor,” in *Proceedings of Embeddes Systems/Embedded Intelligence*, 2002.

[3] G. Schirner, A. Gerstlauer, and R. Dömer, “Abstract, multifaceted modeling of embedded processors for system level design,” in *ASP-DAC ’07: Proceedings of the 2007 conference on Asia South Pacific design automation*, 2007.

[4] S. Manolache, “Schedulability analysis of real-time systems with stochastic task execution times. licentiate thesis no,” Tech. Rep., 2002.

[5] A. Gerstlauer, H. Yu, and D. Gajski, “Rtos modeling for system level design,” in *Proceedings of Design, Automation and Test in Europe, March 2003.*, 2003. [Online]. Available: citeseer.ist.psu.edu/608553.html

[6] H. Posadas, J. Ádamez, P. Sánchez, E. Villar, and F. Blasco, “Posix modeling in systemc,” in *ASP-DAC ’06: Proceedings of the 2006 conference on Asia South Pacific design automation*. New York, NY, USA: ACM Press, 2006, pp. 485–490.

[7] D. Desmet, D. Verkest, and H. DeMan, “Operating system based software generation for systems-on-chip,” in *DAC’00: Design Automation Conference*, 2000.

[8] S.Yoo, G.Nicolescu, L. Gauthier, and A.Jerraya, “Automatic generation of fast timed simulation models for operating systems in soc design,” in *DATE’02: Proceedings of Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2002.

[9] H. Zabel, W. Mueller, and A. Gerstlauer, “Accurate rtos modelling and analysis with systemc,” in *W. Ecker, W. Mueller, R. Doemer (eds.) "Hardware Dependent Software - Principles and Practice"*. Springer Verlag, Dordrecht, Januar 2009.

[10] H. Posadas, J. A. Adamez, E. Villar, F. Blasco, and F. Escuder, “RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model,” *Design Automation for Embedded Systems*, vol. 10, no. 4, pp. 209–227, December 2005.

[11] S. A. Huss and S. Klaus, “Assessment of real-time operating systems characteristics in embedded systems design b systemc models of rtos services,” in *DVCon 07: Design and Verification Conference and Exhibitation*, San Jose, CA, 2007.

[12] Y. Hwang, S. Abdi, and D. Gajski, “Cycle-approximate retargetable performance estimation at the transaction level,” in *DATE ’08: Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 3–8.

[13] H. Zabel and W. Müller, “An efficient time annotation technique in abstract rtos simulations for multiprocessor task migration,” in *Distributed Embedded Systems: Design, Middleware and Resources*, vol. 27. Springer Boston, 2008, pp. 181–190, DIPES2008.