White Box Performance Analysis Considering Static Non-Preemptive Software Scheduling

Alexander Viehl†, Michael Pressler†, Oliver Bringmann†
†FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
{viehl,pressler,bringman}@fzi.de

Wolfgang Rosenstiel†‡
‡Universität Tübingen
Sand 13
72076 Tübingen, Germany
rosenstiel@informatik.uni-tuebingen.de

***Abstract*—In this paper, a novel approach for integrating static non-preemptive software scheduling in formal bottom-up performance evaluation of embedded system models is described. The presented analysis methodology uses a functional SystemC implementation of communicating processes as input. Necessary model extensions towards capturing of static non-preemptive scheduling are introduced and the integration of the software scheduling in the formal analysis process is explained. The applicability of the approach in an automated design flow is presented using a SystemC model of a JPEG encoder.**

I. Introduction

As the complexity of embedded systems further increases, new design and analysis methodologies need to be evolved for closing the steadily growing gap between demanded time-to-market and development cycles.

Platform-based design provides a suitable methodology for embedded hardware/software system development. It provides an early and formalized design specification and hence a platform specification that can be used at the beginning of the software design process. As result, software and hardware development can run in parallel and the overall design development time is reduced. But requirement verification is still complex and can lead to unacceptable long simulation run-times for achieving coverage targets. So new analysis methods need to be evolved that are based on system abstraction and that incorporate model-based analysis tailored to specific verification goals like for example the determination of non-functional requirements.

Especially but not limited to the area of real-time critical embedded systems, performance requirements pose crucial issues to the designers. The system behavior under different shared resources access policies is often insufficiently predictable using simulation and no guarantees on the worst-case behavior can be given.

Software scheduling is one aspect that has a major impact on global performance as well as on local software execution times due to the impact of context switching on micro architectural features as caches. The predictability of preemptive scheduling on the temporal system behavior depends on the predictability on local run-times. Interrupts and task preemption can extend the software execution times. As consequence, a worst-case analysis tackling these issues may lead to very pessimistic analysis results, for example in the case of short time slices or if a single process execution is often interrupted due to long execution times of arithmetic operations.

This work was partially supported by the BMBF project VISION under grant 01M3078B and the DFG project "Communication Analysis for Network-on-Chip" under grant BR 2321/1-1

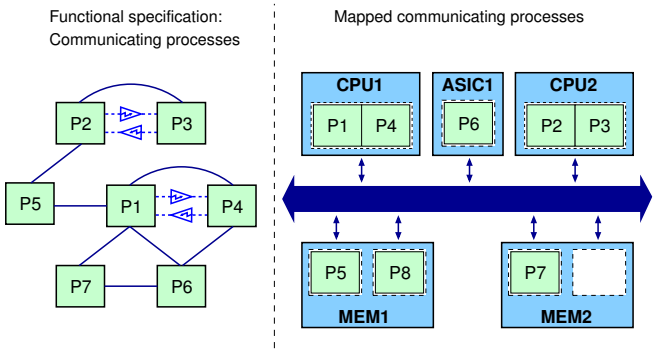


Fig. 1. Mapping of communicating processes

In comparison, the impact of non-preemptive software scheduling on the execution time of the single processes is predictable due to that no preemption penalties have to be considered. In the case of cooperative scheduling, the detection of static cooperative process yield points can be incorporated in worst case/best case execution time (*WCET/BCET*) analysis of code fragments. As result, local software process timing is predictable. Further, no additional process scheduler is necessary to control the order of process execution. But even the impact of such an additional scheduler process that controls the static order of process activation does not lead to an unpredictable temporal behavior if the scheduling is realized without preemption.

Hence, an issue is that the designer has to know the application and points in the control-flow where suspending the single processes and activating other processes in a certain order on the processing resources is beneficial and the schedule does not lead to system deadlocks. Relating these issues, the proposed methodology helps the designer to evaluate different non-preemptive static schedules and to detect deadlocks due to the software process schedule.

The presented work is based on an automated design flow (Section III) that maps the functional implementation of communicating SystemC processes to platform modules in platform-based design as depicted in **Fig. 1**. The design flow was

978-3-9810801-5-5/DATE09 © 2009 EDAA

extended to detect explicit interrupt calls that activate/suspend software processes. This paper describes the incorporation of extensions (Section III-A) towards integration of two classes of non-preemptive software scheduling, namely static order non-preemptive scheduling (Section III-C) and cooperative non-preemptive software scheduling (Section III-B). On account of that the integration is based on accurate model transformation (Section III-D) and not on pessimistic estimation, further scheduling policies (as e.g. TDMA scheduling) can be integrated on top, leading to the ability of hierarchical scheduling. Section IV presents a case study on the application of the extended methodology to a JPEG encoder SystemC model.

II. Related Work

Many approaches in fundamental and current literature tackle the issue of global performance analysis under shared resource utilization. These can be distinguished between simulation-based approaches and analytic approaches. Simulation-based methodologies lack a sufficient corner case coverage in a tolerable amount of time. Although methodologies as TLM [1] allow the consideration of different parts of the system at different levels of abstraction for speeding up simulation, TLM-based approaches (like e.g. [2], [3], [4]) can not generally break these limitations.

Analytic approaches have to be distinguished whether the underlying model is derived from functional system implementations [5], [6] (white box approach) or whether the analysis model is explicitly modeled [7], [8] or derived from formal specifications like UML models [9], [10], synchronous dataflow graphs [11] (black box approaches). Black box approaches abstract from the internal behavior of components and consider task graphs as representation of the internal communication and inter task communication. Based on this model, efficient analysis towards performance and schedulability can be performed (e.g. TDMA scheduling and optimization [12], [13]). The missing link from a direct derivation of the analysis model and the lack of representing aspects like synchronization by communication and the representation of complex interaction schemes and task internal control flow descriptions like loops and branches limit the practical applicability.

Other approaches like timed automata [14] or Petri nets [15] require an explicit description of the system in the analysis model and are very fine grained. The problem sizes to be analyzed are not suitable for complex systems. The application of these methods is more likely the area of single component verification at late stages of the design flow.

White box approaches derive the analysis model directly from a functional implementation of a system. The approach presented in [16] extracts an abstract analysis component model from functional implementations. Issues are the incorporation of complex component interaction schemes as well as the consideration of different synchronization primitives by communication.

In [17] an approach for extracting a formal system representation model called *communication dependency graph* from a functional implementation and information on component platform mapping is presented. The analysis methodology is restricted to include concurrent resource utilization by guaranteeing conflict free resource access using a method for conflict analysis. The approach presented in this paper is based on this approach and extends it towards static non-preemptive software scheduling on multi processor system-on-chip (*MPSoC*) platforms.

A. Analysis model

To represent quantitative temporal communication and computation properties of a design at system level, a model called *communication dependency graph (CDG)* is used.

A *communication dependency graph (CDG)* denotes a consolidated representation of a system consisting of communicating processes. In each process, only communication endpoints and the temporal and causal behavior between them are considered. The control-flow is represented by edges connecting communication endpoints. An edge exists if at least one path in the control-flow graph connects the communication endpoints without passing any other communication endpoint. The communication endpoints are characterized concerning the synchronization behavior, and whether they represent sending or receiving events.

Definition 1 (CDG). A *CDG* is basically denoted by $CDG := < V_{CDG}, E_{CDG}, E_{COM}, \tau_{CDG}, l_{CDG} >$, where

- V_{CDG} is a set of nodes representing communication endpoints.
- $E_{CDG} \subseteq V_{CDG} \times V_{CDG}$ is a set of directed edges describing the precedence dependencies between nodes.
- $E_{COM} \subseteq V_{send} \times V_{rec}$, with $V_{send} = v \in V_{CDG} : \tau_{CDG}(v) \in \{send_{async}\}$ and $V_{rec} = v \in V_{CDG} : \tau_{CDG}(v) \in \{receive_{async}, receive_{sync}\}$ is a set of directed edges describing the communication instances
- The function $\tau_{CDG}(v)$: $V_{CDG} \rightarrow \{send_{async}, receive_{async}, receive_{sync}, init\}$ denotes the type of each node.
- The edge weights are represented by the function $l_{CDG} : E_{CDG} \rightarrow \mathbb{N}_0 \times \mathbb{N}_0$ with minimum and maximum execution time $l_{CDG}(v_1, v_2) = (cs_{min}, cs_{max})$ between the nodes $v_1, v_2 \in V_{CDG}$.

III. Design and analysis flow

Due to the limited space in this paper, the general design flow depicted in **Fig. 2** is explained briefly. The starting point is the so-called system design which consists of communicating processes in SystemC/C++, an abstract platform description, the model of the temporal system environment, information on functional mapping and parameterization as well as requirements on the envisaged system. The first major part of the design flow consists of abstracting communicating software processes functionality with the objective of determining local temporal properties and an abstract representation of the global communication structure as represented by the CDG model. The software abstraction methodology and tool flow was presented in [17], [18], [19]. It was extended to determine static interrupt calls in assembler files. Continuing with the results from software abstraction, the global timing analysis (i.e. *communication analysis*) for incorporating the impact of blocking communication primitives on the global timing of the system can be applied if no cooperative scheduling will be integrated in the system. After communication analysis, performance analysis and requirement evaluation based on the determined global system timing can take place. The calculated properties can then be used for an exploration of the system design. The new aspects presented in this paper are necessary

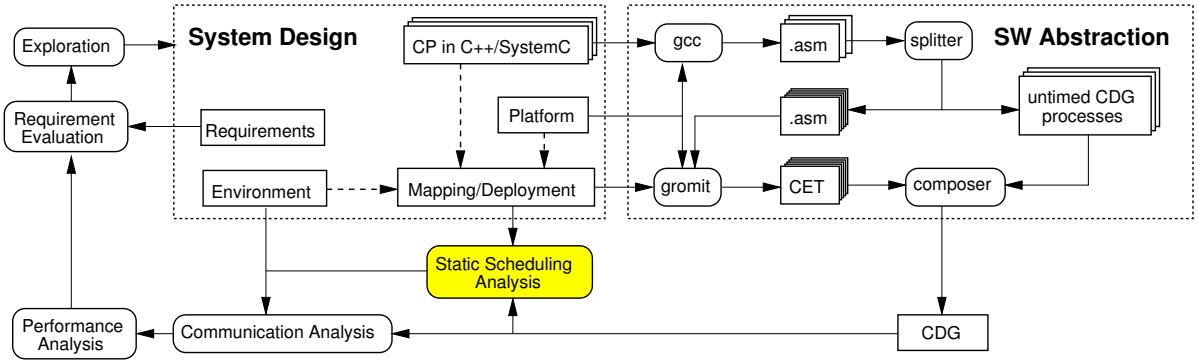


Fig. 2. Analysis flow

to be integrated in the design flow if cooperative scheduling is specified within the system design. The statically determined CDG with interrupt extensions is sufficient for the worst case incorporation of cooperative process scheduling. For this issue, a method for static CDG transformation with the objective of generating CDG models without interrupts is applied, which is presented in Section III-B.

A. Analysis model extension

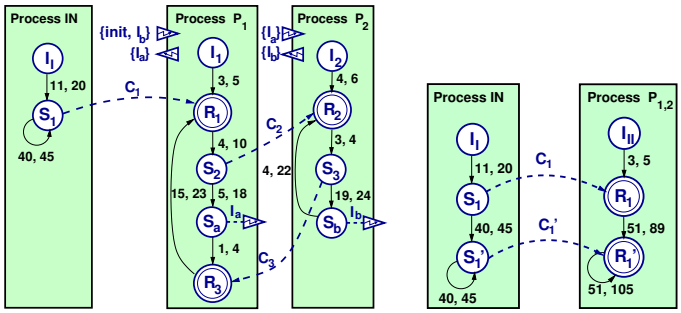
The CDG model presented in Section II-A represents plain communicating processes. In this section, we extend the basic CDG model for expressing what we call hierarchic behavior.

Definition 2 (CDG interrupt extensions)**.** The CDG definition is extended by:

- P_{CDG} is a set of communicating processes
- The mapping function $\varrho_{CDG} : V_{CDG} \rightarrow P_{CDG}$ that describes the affiliation of each node $v \in V_{CDG}$ to a process $\nu \in P_{CDG}$
- I_{CDG} is a set of interrupts
- $E_{INT} \subseteq V_{send} \times I_{CDG}$, with $V_{send} = v \in V_{CDG} : \tau_{CDG}(v) = \{send_{async}\}$ is a set of interrupt emissions
- The function α_{CDG} : $P_{CDG} \rightarrow (I_{CDG} \cup init)^{(n)}$ denotes the set of $n \in \mathbb{N}_0$ interrupts that activate a process.
- The function $\delta_{CDG} : P_{CDG} \rightarrow {I_{CDG}}^{(m)}$ denotes the set of $m \in \mathbb{N}_0$ interrupts that deactivate a process.
- The minimum and maximum interrupt service latency is_{min}, is_{max} of an interrupt $i \in E_{INT}$ is determined by the function $l_{INT} : I_{CDG} \rightarrow \mathbb{N}_0 \times \mathbb{N}_0$ with $l_{INT}(i) = (is_{min}, is_{max})$

Fig. 3a shows a simple system consisting of three communicating processes P_1, P_2, and *IN*. Blocking communication nodes are represented by two concentric circles whereas non-blocking nodes are represented by only one circle. Communication instances are shown as dotted lines, whereas control-flow edges are represented by drawn through lines. The processes P_1 and P_2 are mapped to one single computation resource whereas process IN provides input data. Both processes are scheduled cooperatively using the interrupts I_a and I_b.

Elements related to interrupts are graphically represented by triangular shapes. Interrupt emission of a process at a communication endpoint is characterized by an emission triangle that is connected via dotted lines with the emitting communication node. The interrupt-based activation (deactivation) of a process is represented by a triangle pointing to (away from) the process



(a) Extended CDG with cooperative scheduling (b) Equivalent CDG

Fig. 3. Cooperative process scheduling

in the left top part of the bounding process box. The association of a graphical model element with an interrupt I is denoted by annotation of I to the related triangle. In the example, process P_1 is activated initially (*INIT*) and by interrupt I_b. It is deactivated by interrupt I_a. P_1 emits I_a after receiving communication instance C_1 and sending communication instance C_2. Due to the deactivation sensitivity of P_1 to I_a, the process deactivates itself by emitting I_a. Due to the activation sensitivity of P_2 to I_a, process P_2 is activated as result. The process receives the buffered communication data of communication instance C_2 and sends C_3. Subsequently, it deactivates itself and activates P_1 by emitting I_b. P_1 receives the buffered communication instance C_3 and receives communication instance C_1 from process IN. The behavior of the processes recurs.

B. Cooperative scheduling

The extensions to the CDG allow the description of arbitrary scheduling policies, therein also preemptive policies. However, a general consideration of these extensions required intricate analysis and hence limits analyzable system complexities. Because of that, restrictions are formulated that describe the subset of cooperative scheduling policies and allow an efficient analysis by static model transformation.

Definition 3 (Cooperatively scheduled processes)**.** A set $\Gamma^{(k)}$ of k processes $\{\nu_1, ..., \nu_k\}$ is cooperatively scheduled if:
$i \in \{1 \leq i \leq k\}, j \in 1 \leq j \leq k : j \neq i$ and

- $\exists_1 i : init \in \alpha_{CDG}(\nu_i)$ This condition states that exactly one process exists that is initially activated.
- $\forall i \ \exists_1 \sigma \ \in I_{CDG} : \sigma \in \alpha_{CDG}(\nu_j)$ This conditions states that exactly one activating interrupt for each process exists.

- $\forall i \forall \sigma \in \delta_{CDG}(\nu_i) \exists_1 j : \alpha_{CDG}(\nu_j)$ This condition states that each deactivating interrupt of a process activates another process
- $\forall i \ \forall \sigma \in \delta_{CDG}(\nu_i) : (\sigma \times v) \in E_{INT} \wedge \varrho_{CDG}(v) = \nu_i$ This condition restricts the scheduling to non-preemptive behavior.

The model does not need to be restricted concerning process sets containing disjoint subsets that activate each other without an initial activation or external activation by other processes. Because of that they are never activated, the transformation algorithm presented in Section III-B automatically removes them from the graph.

C. Static non-preemptive scheduling

The restricted CDG extensions can be used to model static non-preemptive software scheduling. The basic idea is the representation of a scheduler as additional process that explicitly activates the single processes and deactivates itself using the interrupt extensions. On cooperative process deactivation, the scheduler is called and selects the next process to be activated. **Fig. 4** shows an example of a CDG containing a static scheduler process.

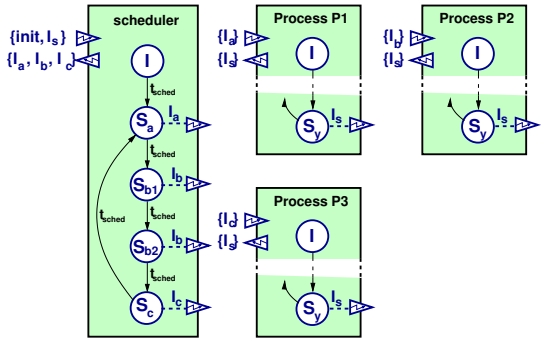


Fig. 4. Static order process scheduling

The scheduler process can be directly adopted from the deployment model. For this issue, information on the static order of processes is contained in this model as vector $\Lambda : P^{(m)}_{CDG} \rightarrow P^{(n)}_{CDG}$. According to the example in **Fig. 4**, $\Lambda(P_1, P_2, P_3) = (P_1, P_2, P_2, P_3)$. During analysis, the interrupt service latency l_{INT} and further scheduling overhead t_{sched} are integrated with the objective of exploration of different schedules. The benefit from the functional process modeling perspective is that the implementation of the processes does not need to be modified due to the entire process call order is specified by the deployment model and enforced by the scheduler process. However, each switch between two functional processes inherits an additional amount of $(l_{INT}+t_{sched})$ in the resulting CET that potentially leads to an overall performance decrease. The application of static schedules can lead to further benefits, if the temporal environment is unknown during functional implementation and later, when the (e.g. bursty) environment characteristics is known, the scheduling can be modified for optimizing properties as buffer sizes or end-to-end latencies without affecting the functional implementation.

D. Incorporation of model extensions

The restricted model extensions are handled by static model transformation. The basic idea is the construction of one new process that represents the behavior of the statically scheduled processes. This is possible due to that the process call order is static and dynamic behavior as process preemption is permitted by the restrictions of Definition 3. The resulting process represents the behavior of the single processes without the inclusion of interrupts or process activation/deactivation by interrupts.

1) Model transformation: An algorithm was developed that transforms a set of cooperatively scheduled processes to one resulting process under consideration of the interaction with other processes in the CDG. This algorithm is basically depicted in **Algorithm 1**. The general proceeding is that the process activation order is traversed, beginning at the initially activated process. The cooperative processes are subsumed to a single resulting process that represents the temporal behavior of the cooperative processes. It is integrated within the interprocess communication with the remaining processes of the system. Temporal properties of communication instances within the cooperative group, control-flow edges, interrupt latencies and scheduling overhead are accumulated and represented by control-flow edges of the resulting process.

The processes are traversed as long as no communication with a process outside the group of cooperatively scheduled processes is found. If an external communication (communication endpoint) is found, it is inserted in the resulting process as the end of the collapsed edge. The previously accumulated execution times are attributed to this edge. The call order and the internal control-flow of the processes are traversed until the structure of the resulting process recurs. Then a fix point is determined and the transformation ends. The transformed CDG of the cooperatively scheduled communicating processes depicted in Fig. 3a is shown in **Fig. 3b**.

An additional benefit apart from the general inclusion of cooperative scheduling is that this inclusion by transformation does no lead to additional pessimism during analysis. The developed algorithm further contains the handling of complex control-flow statements as cascaded loops. Due to the limited space, these extensions are not contained in the depicted algorithm. However, they were implemented and the case study which contains such complex patterns was processed with the implementation.

IV. Case study: JPEG encoder

For presenting the applicability of the presented approach, a SystemC model of a JPEG encoder was used. For allowing the representation of cooperative software scheduling in the functional source code, `sc_yield(sc_process_handle target)` was defined as extension to SystemC. The objective for defining these extensions is the localization of cooperative *yield* calls in the source code for CDG generation. Further, a user space library was developed that allows a semantically equivalent simulation of extended SystemC models and a validation of the analysis results. According to the CDG model extensions introduced in Section III-A, the JPEG encoder CDG shown in **Fig. 5** was automatically derived. The SystemC model of the JPEG encoder consists of six processes with four processes performing the functionalities of DCT(d), quantization(q), zigzag transformation(z), and run length encoding(r). These four processes are realized as software. The annotated BCET/WCET values were analytically calculated for a PowerPC 750 with 100 MHz and 8 kB L1

Algorithm 1 Cooperative schedule inclusion

Require: Communication Dependency Graph
Ensure: Flattening of hierarchical behavior

$p_{cur} = p \in P_{CDG} : \alpha(p) \cap init \neq \varnothing$ //current process
$v_{cur} = p_{cur}.initNode()$ //current node
$e_{cur} = new()$ //current edge
$p_{ret}.add(v_{cur})$ //return process
$s_{cur} = (p_{coop1}.initNode(), ..., p_{coopn}.initNode())$ //current system state
$S_s = []$ //set of all system states
while $(S_s \cap s_{cur} == \varnothing) \wedge (\exists i \in I_{CDG} : (v_a, i) \in E_{INT}$ **do**
  $S_s.put(s_{cur})$ //add current state to set of states
  $v_{next} = p_{cur}.nextNode()$ //next node of current process
  $e_{next} = (v_{cur}, v_{next}) \in E_{CDG}$ //visit next edge
  $l_{CDG}(e_{cur}) = l_{CDG}(e_{cur}) \oplus l_{CDG}(e_{next})$ //add execution time
  if $\exists i_{next} \in I_{CDG} : (v_{next}, i_{next}) \in E_{INT}$ //interrupt emission in next node **then**
    $p_{next} = p \in P_{coop} : \alpha(p) \cap i_{next} \neq \varnothing$
    $l_{CDG}(e_{cur}) = l_{CDG}(e_{cur}) \oplus l_{INT}(i_{next})$
  else
    if $\exists v_{comm} \in V_{CDG} : (v_{next}, v_{com}) \in E_{COM} \wedge \varrho(v_{com}) \notin P_{coop}$ **then**
      //external communication instance
      $e_{cur} = (v_{cur}, v_{next})$
      $v_{cur} = v_{next}$
      $p_{ret}.add(v_{cur})$
      $E_{CDG}.add(e_{cur})$
      $e_{cur} = new()$ //reset current edge
    else
      //intra process communication
    end if
  end if
  $S_{cur}.updateState(p_{cur})$
end while
$v_{ret} findReturnNode()$
$e_{cur} = (v_{cur}, v_{ret})$
$E_{CDG}.add(e_{cur})$

cache. The depicted CDG implies a static order schedule of all four processes with $\Lambda(d, q, z, r) = (d, q, z, r)$ that was derived from the deployment specification. Micro architectural

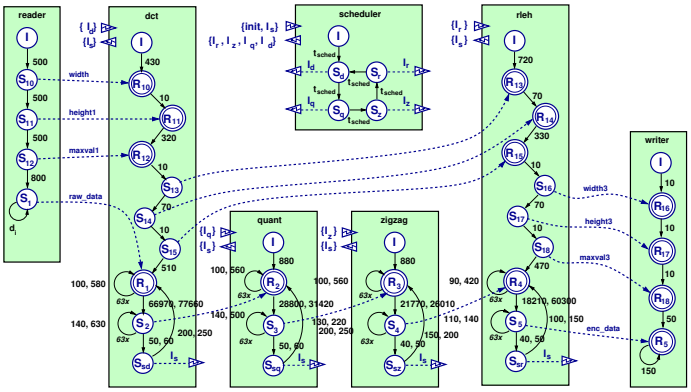


Fig. 5. Static order scheduled JPEG encoder CDG

platform component properties of the software parts (like e.g. clock frequencies, caches and pipeline characteristics) as well as the application of scheduling can be explored using the proposed methodology. In this section, we compare models with and without cooperative scheduling under different input data rates d_i. For evaluation, the utilization of underlying hardware components is used for comparison.

Due to the focus on computational aspects in this paper, we consider idealized temporal communication properties. This means, that communication latency and transmission duration are considered to be 0. As described in Section III, temporal communication properties are included in the existing analysis flow and can be seamlessly integrated with the proposed scheduling extensions in architectural exploration.

In the following, different configurations towards parameter exploration were examined. As basis for execution time analysis, four microcontroller configurations of a PPC 750 were used: μC_1 and μC_2 with 100 MHz and 8 kB L1 cache, μC_3 with 250 MHz and 16 kB L1, and μC_4 with 400 MHz and 32 kB L1 cache. The table in **Fig. 6** characterizes the used configurations concerning process mapping with cooperative scheduling to microcontrollers. Given are the inter arrival time

CONF #	d	q	z	r	d_i^{-1}	l_{INT}
1c	μC_1^*	μC_1	μC_1	μC_1	50 kHz	20 μs
2c	μC_1^*	μC_2^*	μC_2	μC_1	50 kHz	20 μs
3c	μC_1^*	μC_1	μC_2^*	μC_2	50 kHz	20 μs
4c	μC_1^*	μC_1	μC_1	μC_1	80 kHz	15 μs
5c	μC_3^*	μC_3	μC_3	μC_3	80 kHz	15 μs
6c	μC_4^*	μC_1^*	μC_1	μC_4	80 kHz	15 μs

Fig. 6. Cooperative scheduling parametrization

of input packets d_i and the interrupt service latency l_{INT} of all interrupts I_{CDG}. An asterisk denotes that the process is initially started at the platform component. Of cause, these parameters can be specified using intervals, complex input stream patterns and different values for each interrupt. The first cooperative configuration schedules all processes cooperatively on a single resource. In the second cooperative configuration, two processes at a time are mapped on one of the two identical microcontrollers. In Configuration 3c, the mapping is changed. In difference to Configuration 1c, Configuration 4c has a higher input data rate and lower interrupt latency. The fifth configuration executes all processes on a faster resource with the same parameters as Configuration 4c. Configurations 6c maps the processes with higher computation demand to resources with higher computation performance. In Configuration 6c, the processes *dct* and *rleh* are mapped to the same resource.

The configurations depicted in **Fig. 7** characterize a static scheduling of the four processes using a scheduler process in μC_1. The first two static configurations map the processes with a variation of the scheduling overhead. Configuration 3s realizes a static schedule in which each process is called twice with the same parameters as in Configuration 2s.

CONF #	$\Lambda(d, q, z, r)$	d_i^{-1}	l_{INT}	t_{sched}
1s	d, q, z, r	50 kHz	20 μs	2 μs
2s	d, q, z, r	50 kHz	20 μs	5 μs
3s	d, d, q, q, z, z, r, r	50 kHz	20 μs	5 μs

Fig. 7. Static order scheduling parametrization

The table in **Fig. 8** shows the resulting utilization numbers of the single microcontrollers executing the scheduled processes using the configurations from both parametrization tables. The sum of component utilization in Configurations 1c, 2c, and 3c is equal. If four processes are scheduled on one resource, four context switches are needed for the complete computation of 64 packets according to the cooperative scheduling depicted in Fig. 5. If two processes are scheduled to one processing unit,

CONF#	μC_1	μC_2	μC_3	μC_4
1c	36–52%			
1s	43–60%			
2s	44–60%			
3s	44–60%			
2c	25–40%	12%		
3c	22–23%	14–30%		
4c	56–82%			
5c			27–37%	
6c	17–19%			9–17%

Fig. 8. JPEG encoder experiment results

two context switches are needed on each of the two processing units. As result, the same number of context switches as in Configuration 1c is inherited. Due to the same micro architectural configuration, the same input rates and interrupt latencies, the sum of utilizations is constant (in spite of inaccuracy due to rounding errors). Further, the utilization numbers in Configuration 3c show a good load balancing on the computation resources, which can be important if input data rates shall be increased without wasting computation resources. In Configuration 4c, a 67% higher utilization can be stated at a 60% higher input rate due to the effect of different interrupt latencies. The ability to explore the underlying hardware resources is shown in Configuration 5c. Due to that the same parameters (d_i, l_{INT}) as in Configuration 4c are used and all processes are scheduled the same order on one a higher computation resource, a 55% less maximal utilization can be calculated. In Configuration 6c, both processes with the highest computation demand are scheduled on the resource with the highest computation performance. As result, the two cores μC_1 and μC_4 are nearly equally utilized which is a benefit for achieving a high system utilization. In comparison to Configuration 1c, Configuration 1s leads to higher system utilization due to increased context switching numbers with the scheduler process. Configuration 2s has a higher utilization due to higher scheduling overhead costs. The utilization of Configuration 3s is equal to the numbers of Configuration 2s due to that the resulting number of overall context switches is equal in both configurations. Please note that this refers to the utilization. The end-to-end latency will be increased, for example.

V. Conclusion and further work

In this paper, a novel approach for the integration of cooperative and static non-preemptive software scheduling in formal white box performance analysis was presented. The approach is based on an analysis methodology that abstracts communicating processes in SystemC and performs communication analysis with the objective to determine the temporal impact of communication protocols, synchronization by communication and blocking communication primitives. Extensions to the analysis model were made that allow representing hierarchical behavior in order to express static non-preemptive and cooperative software scheduling. Restrictions towards analyzability were formulated for reducing analysis complexity. An algorithm was presented that flattens process activation call hierarchies and transforms the scheduled processes to an equivalent representation on which previously developed analysis methods can be applied. The described design flow and analysis extensions were implemented and integrated in the *SysXplorer* framework. A promising case study analyzing a JPEG decoder was presented for showing the practical applicability of the presented approach for exploration of arbitrary scheduling and platform configurations. Further work will integrate additional access scheduling policies of computation and communication for enabling the applicability to a wider variety of domains and application areas.

References

[1] A. Donlin, "Transaction level modeling: flows and use models," in *CODES+ISSS '04*, 2004.

[2] J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel, "High-Performance Timing Simulation of Embedded Software," in *Proceedings of the Design Automation Conference (DAC)*, 2008.

[3] M. Radetzki and R. S. Khaligh, "Accuracy-Adaptive Simulation of Transaction Level Models," in *Proceedings of DATE*, 2008.

[4] P. Gerin, H. Shen, A. Chureau, A. Bouchhima, and A. Jerraya, "Flexible and executable hardware/software interface modeling for multiprocessor SoC design using SystemC," in *ASP-DAC '07: Proceedings of the 2007 conference on Asia South Pacific design automation*, 2007.

[5] A. Viehl, M. Schwarz, O. Bringmann, and W. Rosenstiel, "Probabilistic Performance Risk Analysis at System-Level," in *CODES+ISSS '07*, 2007.

[6] K. Albers, F. Bodmann, and F. Slomka, "Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems," in *ECRTS '06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*, 2006.

[7] W. Haid and L. Thiele, "Complex task activation schemes in system level performance analysis," in *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007.

[8] P. Pop, P. Eles, Z. Peng, and T. Pop, "Analysis and optimization of distributed real-time embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 3, 2006.

[9] M. Gonzalez Harbour, J. Gutierrez Garcia, J. Palencia Gutierrez, and J. Drake Moyano, "MAST: Modeling and analysis suite for real time applications," in *Proc. Real-Time Systems, 13th Euromicro Conference on*, J. Gutierrez Garcia, Ed., 2001.

[10] A. Viehl, T. Schönwald, O. Bringmann, and W. Rosenstiel, "Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design," in *Proceedings of the Design Automation and Test in Europe Conference (DATE), Munich, Germany*, 2006.

[11] S. Schliecker, S. Stein, and R. Ernst, "Performance Analysis of Complex Systems by Integration of Dataflow Graphs and Compositional Performance Analysis," in *Proc. Design, Automation & Test in Europe Conference & Exhibition DATE '07*, S. Stein, Ed., 2007.

[12] E. Wandeler and L. Thiele, "Optimal TDMA time slot and cycle length allocation for hard real-time systems," in *ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation*, 2006.

[13] T. Pop, P. Pop, P. Eles, and Z. Peng, "Bus access optimisation for FlexRay-based distributed embedded systems," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, 2007.

[14] M. Hendriks and M. Verhoef, "Timed automata based analysis of embedded system architectures," *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 25-29 April 2006.

[15] J. Trowitzsch, A. Zimmermann, and G. Hommel, "Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets." in *IPDPS*, 2005.

[16] K. Albers and F. Slomka, "Efficient Feasibility Analysis for Real-Time Systems with EDF Scheduling," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, 2005.

[17] A. Siebenborn, A. Viehl, O. Bringmann, and W. Rosenstiel, "Control-Flow Aware Communication and Conflict Analysis of Parallel Processes," in *Proceedings of the 12th Asia and South Pacific Design Automation Conference ASP-DAC 2007, Yokohama, Japan*, 2007.

[18] M. Krause, O. Bringmann, and W. Rosenstiel, "Target Software Generation: An Approach for Automatic Mapping of SystemC Specifications onto Real-Time Operating Systems," *Springer: Design Automation for Embedded Systems*, 2007.

[19] A. Hergenhan and W. Rosenstiel, "Static Timing Analysis of Embedded Software on Modern Processor Architectures," in *Proceedings of the DATE 2000 Conference, Paris, France*, 2000.