A Formal Approach to Robustness Maximization of Complex Heterogeneous Embedded Systems

Arne Hamann, Razvan Racu, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig, Germany
{hamann|racu|ernst}@ida.ing.tu-bs.de

ABSTRACT

Embedded system optimization typically considers objectives such as cost, timing, buffer sizes and power consumption. Robustness criteria, i.e. sensitivity of the system to variations of properties like execution and transmission delays, input data rates, CPU clock rates, etc., has found less attention despite its practical relevance.

In this paper we introduce robustness metrics and propose an algorithm considering these metrics in design space exploration and system optimization. The algorithm can optimize for static and for dynamic robustness, the latter including system or designer reactions to property variations. We explain several applications ranging from platform optimization to critical component identification.

By means of extensive experiments we show that design space exploration pursuing classical design goals does not necessarily yield robust systems, and that our method leads to systems with significantly higher design robustness.

Categories and Subject Descriptors

C.3 [**Special-Purpose and application-based systems**]: Real-time and embedded systems; C.4 [**Performance of systems**]: Modeling techniques; Performance attributes; Reliability, availability, and serviceability

General Terms

Algorithms, Design, Performance, Reliability, Verification

1. INTRODUCTION

Design robustness is a general concern that grows with system complexity. For instance, it is known that small task core execution time modifications in systems with complex performance dependencies can have drastic non-intuitive effects on the overall system performance, and might lead to constraint violations. Since a major change of the system in reaction to such performance degradation effects might not be possible at late design phases or in the field, it is important to early choose a system parameter configuration offering sufficient robustness with respect to system properties presumably subject to later modifications. Scenarios under which such system property variation can occur include late feature requests, product variants, software updates, and bug-fixes.

CODES+ISSS'06, October 22–25, 2006, Seoul, Korea.

Robustness evaluation using simulation is a tedious if not practically impossible task as simulation models do not support many of the possible property changes, including increased process execution times or modified communication volumes. Formal performance models are more appropriate, such as introduced in recent frameworks [8, 3] that are able to tightly determine various system performance properties including timing, buffer sizes, and power consumption.

In this paper we address the application of formal models to robustness optimization. We introduce and motivate two different robustness metrics that capture system robustness under different assumptions, and present a method explicitly taking these metrics into account during design space exploration. The results of the approach allow the designer to early choose a balanced system configuration offering large robustness for critical components. This way, system stability and maintainability can be significantly increased.

The remainder of the paper is structured as follows. We will first precisely formulate the problem we address in this paper (section 2). Afterwards we will give a short survey of related work (section 3). We then present our approaches for system robustness optimization, i.e. on the one hand appropriate cost functions (section 4.1) reflecting desired system robustness properties and on the other hand efficient design space exploration strategies (section 4.2). We then discuss a small but realistic case study and maximize its robustness using our approaches (section 5). Finally, we use large synthetic example systems to compare the efficiency of the presented approaches, i.e. quality of achieved results in relation to run-time (section 6).

2. PROBLEM FORMULATION

In this section we first briefly introduce the application model used in this paper (section 2.1). We then outline the robustness problem arising during the design of complex embedded systems due to system property variations (sections 2.2 and 2.3).

2.1 Application model

Generally, a system consists of an execution platform including *computation and communication resources*, and a set of *tasks* that run on this platform communicating either directly, e.g. over shared memory, or through logic *communication links*. Thereby, the tasks and communication links are mapped on the computation and communication resources, respectively.

The example system in figure 1 contains, for instance, the computation resource *DSP* with the mapped tasks upd, $fltr$, and $ctrl$, as well as the communication resource BUS with the logic communication links $c1$, $c2$, $c3$, $c4$, and $c5$.

All tasks are activated due to an activating event. Activating events can be generated in a multitude of ways, including expiration of a time trigger, external or internal interrupts, and event chains starting with an external event. Each task is assumed to

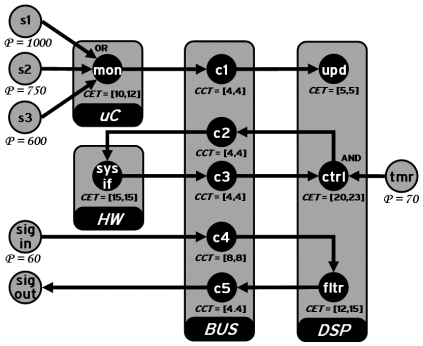


Figure 1: System on chip example

have one input FIFO, from where its activating data is read. At the end of its execution, the task writes data into the input FIFO of a dependent task or sends the data over a communication link.

In the formalism underlying this paper we use so called *standard event models* to describe the possible timing of activating events. They are described using several parameters. For example, a *strictly periodic event model* has one parameter $\mathcal{P}$ and states that each event exactly arrives periodically every $\mathcal{P}$ time units. This simple model can be extended with the notion of jitter, leading to a *periodic with jitter event model*. Such an event model is described by two parameters $(\mathcal{P}, \mathcal{J})$. It states that the events generally occur periodically, but can jitter around the exact position within a jitter interval $\mathcal{J}$. If the jitter is larger than the period, then two or more events can occur at the same time, leading to bursts. To describe a *bursty event model*, the periodic with jitter event model can be extended with the d^- parameter that captures the minimum distance between two events within the burst.

Furthermore, each task is associated with a *best-case* and a *worst-case core execution time (CET)*, i.e. its minimum and maximum execution time assuming no interrupts by tasks mapped on the same computation resource. Accordingly, each communication link is associated with a *best-case* and *worst-case core communication time (CCT)*, i.e. the minimum and maximum transmission time for one communication request assuming no concurrent requests on the same communication resource. Note that in the example SoC shown in figure 1 the core execution and communication times are given as intervals.

For the case that multiple tasks or communication links share the same resource, it may be claimed by more than one task or communication link at the same time. We need scheduling to arbitrate such conflicts. Therfore, a resource is associated with a scheduler selecting one task or communication link to which it grants the resource according to some scheduling policy.

Scheduling analysis calculates the *worst-case response time*, i.e. the time between activation and completion, for all tasks and communication links sharing a resource under the control of such a scheduler.

A more detailed discussion about the application model, standard event models and scheduling analysis of complex heterogeneous embedded systems can be found in [8].

2.2 System property variations

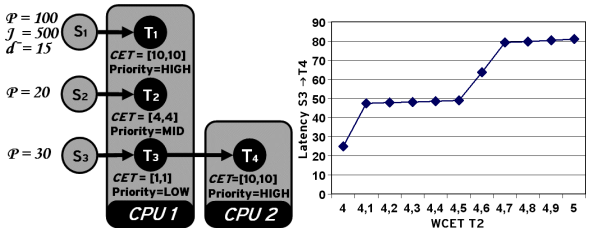
System properties variations put at risk the functioning and performance of a system since they invalidate the assumption under which it was originally dimensioned and configured.

Basically, there are two different kinds of system property variations: variations influencing the system load, and variations influencing the system service capacity.

Reasons for system load variations are mainly changes of software execution path lengths, communication volumes, and input data rates. Scenarios under which such load variations can occur during design time or even in the field include late feature requests, product variants, software updates, and bug-fixes.

That even small changes of a single load influencing system property can have drastic effect on the overall system performance, shall be shown by the example system given in figure 2(a). *CPU*1 and *CPU*2 are both scheduled with the static priority preemptive policy.

Figure 2(b) shows the impact of increasing *WCET* of T_2 on the path latency $S_3 \rightarrow T_4$. We observe that increasing the *WCET* of T_2 from 4 to 4.1 (= 2.5 %) time units leads to an increase of the path latency $S_3 \rightarrow T_4$ from 25 to 47.4 (= 89.6 %) time units. A further increase of the *WCET* of T_2 to 4.7 (= 17.5 %) lengthens the path latency $S_3 \rightarrow T_4$ to 79.2 (= 216.8 %) time units.



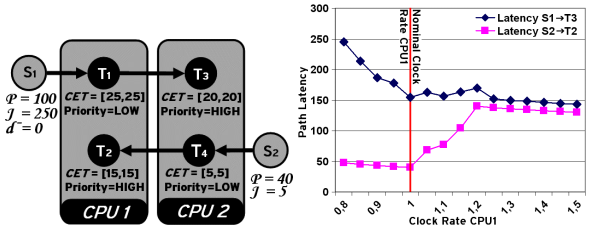
(a) Example system 1 (b) Latency $S_3 \rightarrow T_4$ dependant on the WCET of T_2

Figure 2: Effect of WCET variation

System service capacity variations are caused by modifications of the execution platform, e.g. processor or communication link performance changes. Such variations rarely occur in the field. However, they they are of particular interest during early design space exploration, where load requirements are still subject to changes, and where different alternative system components and architectures need to be evaluated.

The possible impact of changes to a resource's service capacity shall be demonstrated by the simple example given in figure 3(a). *CPU*1 and *CPU*2 are both scheduled with a static priority preemptive policy. The given worst-case execution times correspond to nominal CPU clock rate.

Figure 3(b) shows the impact of changes to the speed of *CPU*1 on the path latencies $S_1 \rightarrow T_3$ and $S_2 \rightarrow T_2$. We observe, that both path latencies increase as we slow down *CPU*1, which is not surprising. However, even speeding up *CPU*1 leads to a deterioration of system performance. A clock rate speed-up of 20% leads to an increase of the path latency $S_2 \rightarrow T_2$ from 40 to 140 time units (= 250 %), and to an increase of the path latency $S_1 \rightarrow T_3$ from 155 to 170 time units (= 9.68 %).



(a) Example system 2 (b) Latencies dependant on the speed of *CPU*1

Figure 3: Effect of resource speed variation

2.3 Robustness to property variations

The example systems in figures 2(a) and 3(a) show that both load and service capacity influencing property variations can have non-intuitive effects on the system performance. Also, small but practical industrial case studies [9] have proven that the influences of

unpredicted property variations on the system performance is an important issue during the design of complex embedded systems.

Therefore, we present in this paper methods helping the designer finding *system parameter configurations*, including the assignment of free system parameters like scheduling (priorities, time slots, etc.), leading to systems with high robustness (low sensitivity) to property variations. In other words, the system shall meet its timing constraints even under considerable changes of system properties, including worst-case execution and communication times, CPU clock rates, input data rates, etc. Note that we focus in this paper on hard real-time constraints.

3. RELATED WORK

In [4] a method for robustness maximization of embedded real-time systems is presented. More precisely, the authors address the allocation problem for tasks with environment-dependant workload requirements. Accordingly, the work aims at finding task allocations, which are robust to environmental changes. The authors' system model assumes homogeneous multiprocessor systems with rate monotonic scheduling and independent strictly periodic tasks with deadlines equal to the periods. Due to this simple system model, the results are of limited use for the robustness maximization of realistic embedded systems.

In [1] an approach measuring the robustness of resource allocations with respect to perturbations in system and environmental conditions is presented. One robustness metric derived in the paper measures the robustness of a system that allocates a set of independent tasks to a set of machines with static non-preemptive scheduling. The overall completion time for the entire set of tasks is required to be robust against errors in execution time estimates. The authors define the system robustness as the minimal euclidian distance between the original system configuration and the hyperplane separating system configurations satisfying certain robustness requirements from the systems with insufficient robustness. This definition of robustness is very intuitive. However, in the case of realistic application models with complex dependencies and more sophisticated scheduling strategies, the required hyperplane can only be determined with extremely large computational effort if several tasks are considered. Therefore, the approach seems not applicable for the robustness maximization of realistic embedded systems.

Adaptive scheduling strategies [6] can be utilized to improve system robustness to property variations. One possibility is to take the past system behavior into account for future scheduling decisions. However, often application specific knowledge needs to be integrated into such adaptive scheduling strategies to achieve good results, which requires additional engineering effort. Also, the target platform needs to support the customization of the scheduling policy for such adaptive approaches to be applicable, which is rarely the case in industrial practice. Therefore, we do not include the choice of the scheduling policy in the robustness maximization approach presented in this paper.

Sensitivity analyses [7, 10] can capture the global effects of single system property variations. They are capable of calculating the boundaries representing the transition between conforming and non-conforming systems for a large variety of system properties, including worst-case execution times, communication volumes, input data rates, processor and communication link performance, etc.

These sensitivity analyses, however, only report the reaction of a system in response to a single design parameter variation. In order to optimize the robustness of a system, we need more comprehensive metrics. Such robustness metrics are defined in the following section 4.1 and used to maximize system robustness in section 4.2.

4. MAXIMIZING SYSTEM ROBUSTNESS

In order to optimize robustness we need on the one hand appropriate cost functions (section 4.1) reflecting the desired robustness properties and on the other hand an efficient design space exploration strategy (section 4.2).

4.1 System robustness metrics

The robustness metrics presented in this section are based on the notion of *slack*. The slack of a system property describes its "head room" for a specific parameter configuration, i.e. the percentage by which its value may be increased (resp. decreased) without violating any system constraint. Depending on the considered system property higher values (e.g. worst-case core execution times) or lower values (e.g. resource speeds) correspond to more slack.

Definition 1 (Slack) *We consider a constrained system S with parameter configuration c and a system property $p \in S$ with the original value $v(p)$.*

Given $v_c^+(p)$, denoting the extreme system property value for p not leading to constraint violations in S configured according to c, the slack of p is defined as follows:

$$\text{slack}_{p;c} = \frac{v_c^+(p) - v(p)}{v(p)} \times 100$$

Note that $v_c^+(p)$ can be calculated by sensitivity analysis algorithms as described in section 3.

4.1.1 Static Design Robustness

The *static design robustness (SDR)* metric expresses the robustness of a fixed parameter configuration for a given constrained system with respect to a set of critical system properties. The SDR metric is relevant for the design scenario where parameters are defined and fixed early at design time and cannot be modified later to reach compatibility for variants, bug-fixes, and updates.

Note that the SDR metric is only defined for working parameter configurations, i.e. configurations satisfying all system constraints.

Definition 2 (Static Design Robustness) *We consider a constrained system $\mathcal{S}$ with parameter configuration c and a set of system properties $\mathcal{P} = \{p_1, \ldots, p_n\}$.*

Given the slacks $\text{slack}_{p_1;c} \ldots \text{slack}_{p_n;c}$ for the properties in $\mathcal{P}$, a set of weights $w_1, \ldots, w_n$ with $\forall_i w_i > 0$ and $w = \sum_{i=1}^{n} w_i$, and a real number k, the static design robustness (SDR) of the configuration c with respect to $\mathcal{P}$ is defined as follows:

$$SDR_{\mathcal{P};c} = \begin{cases} \sqrt[w]{\prod_{i=1}^{n} \mid \text{slack}_{p_i;c}^{w_i} \mid} & , k = 0 \\ \sqrt[k]{\frac{1}{w} \times \sum_{i=1}^{n} \left(w_i \times \mid \text{slack}_{p_i;c}^{k} \mid \right)} & , \textit{otherwise} \end{cases}$$

The SDR metric can be parameterized in two ways.

First of all, the SDR metric allows to weight the influence of the included system properties. This enables the designer to account for different levels of relevance linked to each of the considered system properties. Criteria for her to assign these weights include the estimated probability of future changes and the impact on the overall system performance.

Secondly, the SDR metric can be configured to advantage system properties with either low or high slacks by modifying k.

Figure 4 visualizes the impact of k on the SDR metric for two hypothetical system properties p_1 and p_2 (both weighted 1) for different slack values.

We observe that decreasing and increasing values for k increase the impact of the lower or the higher slack value on the SDR metric, respectively.

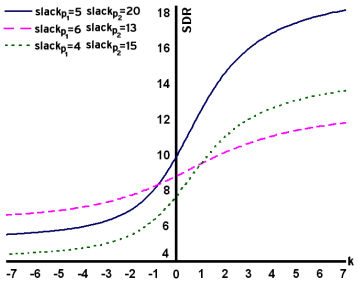


Figure 4: Behavior of the SDR metric for different k

In particular, choosing low values for k weakens the SDR metric value for system configurations with unbalanced robustness properties, i.e. buying extremely high slack for a single property with low slacks for the other properties.

4.1.2 Dynamic Design Robustness

While static design robustness assumes a static system with fixed parameter configuration, *dynamic design robustness (DDR)* includes potential designer or system counteractions in reaction to system property variations. In other words, DDR describes the robustness potential of a given system with respect to the variation of a specific property, which can be achieved by system reconfiguration, i.e. for instance scheduling parameter adaptation. Consequently, the DDR metric is relevant for a design scenario where parameters can be modified during product life time or in the field.

Definition 3 (Dynamic Design Robustness) *We consider a constrained system $\mathcal{S}$ and a system property $p \in \mathcal{S}$.*

Given possible parameter configurations $\mathcal{C} = \{c_1, c_n, \ldots, c_n\}$ of $\mathcal{S}$ and the corresponding slack vector $\mathcal{V} = (\text{slack}_{p;c_1}, \ldots, \text{slack}_{p;c_n})$, the dynamic design robustness (DDR) of S with respect to p is defined as the L_∞ norm of $\mathcal{V}$:

$$DDR_{p;\mathcal{C}} = \|\mathcal{V}\|_\infty = \max_{c \in \mathcal{C}} |\text{slack}_{p;c}|$$

The dynamic design robustness depends on the set of possible configurations $\mathcal{C}$. As an example, it can be possible to react to property changes by adaptation of scheduling parameters or by remapping parts of the application.

DDR can obviously be used to evaluate dynamic systems, but it can more generally be used for the evaluation of the design risk connected to specific components in a given system. More precisely, already early in the design flow the DDR metric allows the designer to determine boundaries for properties of specific components, allowing their integration into the system. This information effectively facilitates feasibility and requirements analysis and greatly assists the designer in pointing out critical system components requiring special focus during specification and implementation.

Another usage scenario for the DDR metric concerns reconfigurable systems. In such a scenario the designer can use the DDR metric to determine the theoretical robustness head room of crucial system components with respect to future changes of their properties. By early choosing a system architecture offering high DDR values for these crucial components the designer can significantly increase system stability and maintainability.

4.2 Exploration control

In this section we present two different approaches for robustness maximization. The first approach (section 4.2.1) is heuristic in nature. It is based on the assumption that large slack for timing properties leads to high system robustness with respect to variations of properties influencing these timing properties. Consequently, the achieved system robustness corresponds to that obtained by classical optimization approaches without explicit consideration of robustness criteria.

In section 6 the first approach is taken as baseline for the evaluation of the second approach (section 4.2.2), which directly uses the robustness metrics presented in section 4.1 to control the optimization.

Note that both approaches utilize a framework for multi-dimensional design space exploration [5], which is based on evolutionary search techniques [2], and sensitivity analysis algorithms [7].

4.2.1 Separated approach

Figure 5(a) visualizes the separated robustness optimization approach.

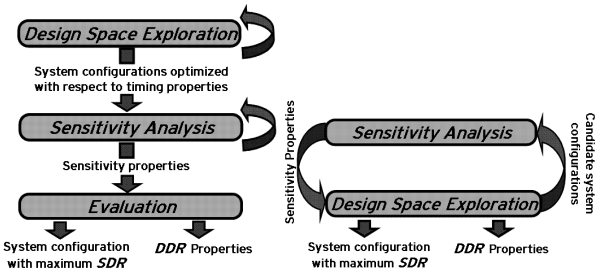
It divides the robustness optimization problem into two steps. The first step is system optimization with respect to its constrained timing properties. Depending on the system this might be the minimization of the lateness, i.e. the difference between latency and deadline, for each constrained end-to-end path in the system, the minimization of jitter constraints, etc.

In the second step, sensitivity properties are determined for system parameter configuration obtained in the first step. These are then evaluated, and the optimal system parameter configurations with respect to the chosen robustness metrics are returned to the user.

4.2.2 Integrated approach

Figure 5(b) visualizes the integrated robustness optimization approach.

In contrast to the separated robustness optimization approach presented in the previous section, i.e. optimization of the system regarding its timing properties and subsequent robustness evaluation, the integrated robustness optimization approach uses robustness evaluation as a function of design space exploration to guide the search towards systems parameter configurations possessing desired robustness properties. This means that exploration is not guided heuristically on the basis of timing properties but directly by robustness properties as primary optimization goal.



(a) Separated Approach (b) Integrated Approach

Figure 5: Robustness Optimization Approaches

5. CASE STUDY

In this section we use the SoC example given in figure 1 as case study to compare both robustness optimization approaches with respect to quality of achieved results. Note that, experiments using large synthetical example systems to compare the average efficiency, i.e. quality of achieved results in relation to run-time, can be found in section 6.

The system in figure 1 represents a SoC consisting of a microcontroller (*uC*), a digital signal processor (*DSP*) and dedicated hardware (*HW*), all connected via an on-chip bus (*BUS*). The *HW* acts as an interface to a physical system. It runs one task (*sys_if*) which

issues actuator commands to the physical system and collects sensor readings. *sys_if* is controlled by controller task *ctrl*, which evaluates the sensor data and calculates the necessary actuator commands. *ctrl* is activated by a periodic timer (*tmr*) and by the arrival of new sensor data (AND-activation in a cycle).

The physical system is additionally monitored by 3 smart sensors ($sens_1$ - $sens_3$), which produce data sporadically as a reaction to irregular system events. This data is registered by an OR-activated monitor task (*mon*) on the *uC*, which decides how to update the control algorithm. This information is sent to task *upd* on the *DSP*, which writes the updated controller parameters into shared memory.

The *DSP* additionally executes a signal-processing task (*fltr*), which filters a stream of data arriving at input *sig_in*, and sends the processed data via output *sig_out*. All communication (with the exception of shared-memory on the *DSP*) is carried out by communication tasks *c1* - *c5* over the on-chip *BUS*.

In order to function correctly, the system has to satisfy maximum latency constraints for the following paths: $sens_i \rightarrow upd$ (denoted $\mathcal{T}_{l_1}$) 70 time units, $sig\_in \rightarrow sig\_out$ (denoted $\mathcal{T}_{l_2}$) 60 time units, and $ctrl \rightarrow ctrl$ (denoted $\mathcal{T}_{l_3}$) 140 time units. Additionally, the jitter at the system output *sig_out* (denoted $\mathcal{J}_{out}$) must not exceed 25 time units. In the following we assume that the *DSP* as well as the *BUS* are scheduled according to a static priority preemptive policy.

5.1 Separated approach

We now optimize the design robustness of the SoC example with the separated approach described above. We assume that the designer is mainly interested in robustness properties for the tasks *ctrl* and *mon* as well as for the communication channels *c2* and *c5*.

We perform several experiments.

First, we want to find system configurations with high static design robustness. Therefore, we perform several optimization runs each focusing the SDR metric on one of the considered tasks or communication channels. More precisely, we choose the weights for the SDR metric in such a way that the task or communication channel we focus on has exactly as much influence on the metric as all others together. Consequently, the focused task or communication channel is weighted 3, whereas all others are weighted 1. Note that for these experiments the SDR metric is configured with $k = 1$.

Secondly, we are interested in the dynamic design robustness properties of the system with respect to each of the considered tasks and communication channels. Here, the priority assignments on the resources *DSP* and *BUS* are taken as dynamic parameters that can be changed later in the design process or, in case of dynamic system control, dynamically at run time.

Table 1 lists the pareto-optimal system configurations with respect to timing properties obtained in the first step of the separated robustness optimization approach.

	Priorities		Timing properties			
#	*BUS* tasks	*DSP* tasks	$\mathcal{T}_{l_1}$	$\mathcal{T}_{l_2}$	$\mathcal{T}_{l_3}$	$\mathcal{J}_{out}$
1	*c3>c4>c5>c2>c1*	*upd>fltr>ctrl*	**55**	42	120	18
2	*c4>c3>c5>c1>c2*	*upd>fltr>ctrl*	59	42	112	18
3	*c4>c2>c5>c3>c1*	*upd>fltr>ctrl*	59	46	108	22
4	*c5>c3>c4>c1>c2*	*upd>fltr>ctrl*	63	42	96	18
5	*c5>c2>c4>c3>c1*	*upd>fltr>ctrl*	63	46	**92**	22
6	*c2>c3>c5>c4>c1*	*fltr>upd>ctrl*	64	39	140	15
7	*c3>c4>c5>c2>c1*	*fltr>upd>ctrl*	70	**27**	120	**3**

Table 1: Pareto-optimal system configurations

Figure 6(a) shows the static design robustness of the obtained system configurations for the different experiments. The maximum reached values are annotated and emphasized. We observe that system configuration 2 possesses the highest static design robustness regardless on which task or communication channel the SDR metric is focused on.

Figure 6(b) shows the dynamic design robustness properties of the system determined with the separated approach. The determined DDR values for each of the considered tasks and communication channels are annotated and emphasized. We observe that system configuration 2 yields the highest robustness for the communication channels $c2$ and $c5$, whereas configurations 2 and 3 offer more robustness for the tasks *ctrl* and *mon*, respectively.

5.2 Integrated approach

We now perform the same set of experiments as in the previous section pursuing the integrated robustness optimization approach. Table 2 lists additionally obtained system configuration not considered by the separated approach.

	Priorities		Timing properties			
#	*BUS* tasks	*DSP* tasks	$\mathcal{T}_{l_1}$	$\mathcal{T}_{l_2}$	$\mathcal{T}_{l_3}$	$\mathcal{J}_{out}$
8	*c4>c5>c3>c2>c1*	*upd>fltr>ctrl*	63	42	126	18
9	*c4>c5>c2>c3>c1*	*upd>fltr>ctrl*	63	46	122	22
10	*c3>c5>c4>c2>c1*	*upd>fltr>ctrl*	**55**	42	134	18
11	*c2>c4>c5>c1>c3*	*fltr>upd>ctrl*	70	31	120	7
12	*c3>c1>c5>c4>c2*	*fltr>upd>ctrl*	70	43	123	19

Table 2: Additional system configurations considered by the integrated approach

Figure 6(c) shows the static design robustness for the different experiments determined by the integrated approach. We observe that two of the newly discovered system configurations possess improved static design robustness properties. System configuration 9 improves the static design robustness focused on the communication channel $c2$ by more than 47% compared to the best value reached with the separated approach. Also for the tasks *ctrl* and *mon* new system configurations could be found with improved SDR values. No improvement could be achieved for the static design robustness focused on communication channel $c5$, the previously obtained system configuration 2 was already optimal.

Figure 6(d) shows the dynamic design robustness properties of the system determined with the integrated approach. Compared to the dynamic design robustness properties determined with the stepwise optimization approach we discovered that the actual dynamic design robustness of the system is higher for the communication channels $c2$ and $c5$ as well as for the task *ctrl*. For the communication channel $c2$ the underestimation was nearly 95%. Also for the communication channel $c2$ and the task *ctrl* the dynamic design robustness was underestimated by 4.7% and 30.8%, respectively.

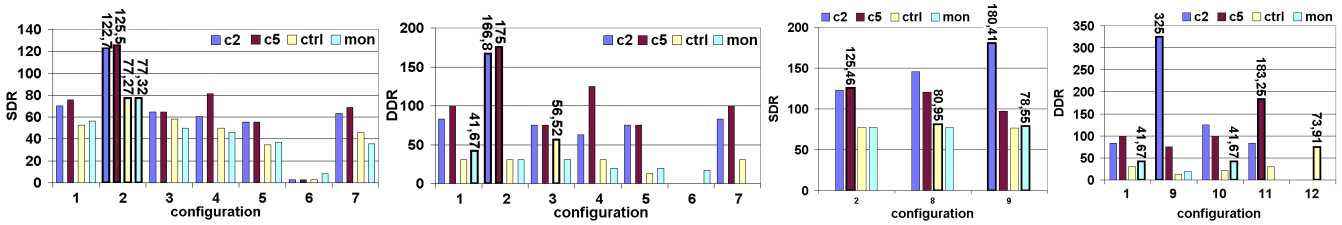
6. EXPERIMENTS

In this section we compare the separated and the integrated robustness optimization approaches with respect to the quality of achieved results and runtime. Therefore, we randomly generated complex systems with 20 tasks and communication channels and optimized randomly chosen static and dynamic robustness properties.

Figure 7(a) shows the average performance of the separated approach. The results where obtained by performing 1000 exploration runs for 10 randomly generated systems. Thereby, each exploration run considered a maximum of 500 different system configurations.

We observe that in 64.79% of the exploration runs we found a system configuration with optimal robustness properties, whereas in 35.12% of the cases only sub-optimal results could be achieved. Note that optimal in this case means maximum robustness reachable with the separated approach, and that these optimal values were determined separately by exhaustive search for each considered system.

Figure 7(b) shows the performance of the integrated robustness optimization approach in comparison to the optimal results which can be achieved by the separated approach. The results were ob-

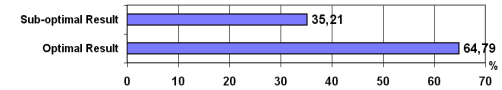


(a) Sep. Approach - SDR (b) Sep. Approach - DDR (c) Int. Approach - SDR (d) Int. Approach - DDR

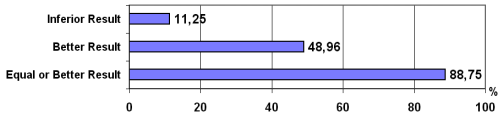
Figure 6: Robustness Optimization Results - Separated and Integrated Approach

tained considering the same systems and performing the same number and type of exploration runs as in the previous experiment.

We observe that in 88.75% of the exploration runs the integrated approach yielded better or equal results compared to the optimum attainable with the separated approach. In nearly 50% of the cases even better results were achieved, and only in 11.25% of all experiments inferior results were obtained in comparison to the optimum of the separated approach.



(a) Performance of the separated approach



(b) Performance of the integrated approach compared to the optimal result of the separated approach

Figure 7: Performance evaluation of the approaches

The runtime of both approaches for one exploration run performed in the context of the above conducted experiments is compared in figure 8. For both approaches the runtime increases with the number of system properties which are included into the calculated robustness metrics. However, the runtime of the integrated approach increases faster. This is due to the fact that the integrated approach calculates the slack of each considered system property for every system configuration tested during exploration. This translates in a runtime which is 2.5 times longer for a single considered system property and approximately 5 times longer for 10 considered system properties.

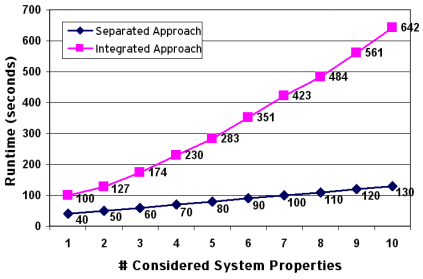


Figure 8: Runtime comparison of the two approaches

7. CONCLUSION

In this paper we have demonstrated that small system property variations in complex embedded systems can have drastic unintuitive effects on system performance. In extreme cases small variations of a single system property can significantly decrease system performance. Since property variation can occur during all design stages and even in the field, it is crucial to consider robustness criteria as early as possible.

A case study and experiments with synthetical examples showed that designing complex embedded systems with focus on classical design goals, e.g. timing properties, is not sufficient to cover robustness requirements.

We therefore presented robustness metrics and proposed an approach to consider these metrics during design space exploration. Experiments showed that our approach can significantly improve static and dynamic design robustness.

8. REFERENCES

[1] S. Ali, A.A. Maciejewski, H.J. Siegel, and J. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, July 2004.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA — a platform and programming language independent interface for search algorithms. *http://www.tik.ee.ethz.ch/pisa/*.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, March 2003.

[4] D. Gu, F. Drews, and L. Welch. Robust task allocation for dynamic distributed real-time systems subject to multiple environmental parameters. In *Proc. of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Columbus, Ohio, USA, June 2005.

[5] A. Hamann, M. Jersak, K. Richter, and R. Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. *Real-Time Systems Journal*, 33(1-3):101–137, July 2006.

[6] C. Lu, J.A. Stankovic, S.H. Son, and G. Tao. Feedback control real-time scheduling: framework, modeling, and algorithms. *Real-Time Systems*, 23(1-2):85-126, 2002.

[7] R. Racu, A. Hamann, and R. Ernst. A Formal Approach to Multi-Dimensional Sensitivity Analysis of Embedded Real-Time Systems. In *Proc. of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[8] K. Richter. *Compositional Performance Analysis*. PhD thesis, Technical University of Braunschweig, 2004.

[9] M. Verhoef, E. Wandeler, L. Thiele, and P. Lieverse. System architecture evaluation using modular performance analysis - a case study. In *Proc. of the 1st IEEE/ACM International Symposium on Leveraging Applications of Formal Methods (ISOLA)*, Pafos, Cyprus, Oct 2004.

[10] Steve Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4), april 1994.