AUTOMOTIVE SYSTEM OPTIMIZATION USING SENSITIVITY ANALYSIS

Razvan Racu, Arne Hamann, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig, Germany

Abstract There have been major advances in formal methods and related tools in embedded system design in recent years that support analysis and optimization of heterogeneous automotive architectures. We demonstrate the application of the SymTA/S methodology to an automotive example where we analyze the sensitivity of different system properties and determine optimal system configurations. The use of this methodology in the early phases of system design conduct to results that cannot be obtained using simulation or prototyping.

1. Introduction

With increasing automotive system complexity, formal methods become more important as a complement to simulation and prototyping. Formal methods have already been used in early architecture design, e.g. to optimize bus architectures as such models can be applied before executable models are available. Recent advances in real-time system analysis have extended the scope of formal models to heterogeneous networks with different protocols and gateways, and with electronic control units (ECUs) running different scheduling algorithms. These formal models can be used to optimize those systems and analyze the sensitivity to later changes in the design process or to run time events such as retransmission due to errors. In this paper, we give a brief introduction to the underlying model and algorithms of the tool, SymTA/S, and give a small example for its application to system integration.

The remainder of this paper is structured as follows. First, we will give a short overview of existing formal techniques for system level performance analysis (Section 2). Afterwards, we briefly present the SymTA/S compositional analysis methodology based on event model interfacing and propagation (Section 3).

In Section 4 we present the main scheduling concepts of the ERCOSEK operating system and the CAN bus protocol.

Finally, we introduce a small example of an automotive system consisting of two independent subsystems (Section 5) and demonstrate the application of the SymTA/S exploration and sensitivity analysis frameworks to system integration (Sections 6 to 9).

2. Formal Techniques in System Performance Analysis

In this section, we briefly review existing analysis approaches from real-time research for formal system performance analysis of heterogeneous distributed systems and MpSoC.

The holistic analysis approach developed by Tindell [12] systematically extended the classical local analysis techniques, considering the scheduling influences along functional paths in the system. He proposed a performance verification model for distributed real-time systems with preemptive task sets communicating via message passing and shared data areas. Eles *et al.* [7] extended this approach for systems consisting of fixed-priority scheduled CPUs connected via a TDMA scheduled bus. Later on, Palencia *et al.* [6] extended the analysis for tasks with precedence relations and activation offsets.

Gresser [4] and Thiele [11] established a different view on scheduling analysis. The individual components or subsystems are seen as entities which interact, or communicate, via event streams. Mathematically speaking, the stream representations are used to capture the dependencies between the equations (or equations sets) that describe the individual components timing. The difference to the holistic approach (that also captures the timing using system-level equations) is that the compositional models are well-structured with respect to the architecture. This is considered a key benefit, since the structuring significantly helps designers to understand the complex dependencies in the system, and it enables a surprisingly simple solution. In the "compositional" approach, an output event stream of one component turns into an input event stream of a connected component. Schedulability analysis, then, can be seen as a flow-analysis problem for event streams that, in principle, can be solved iteratively using event stream propagation.

3. The SymTA/S approach

SymTA/S [10] is a formal system-level performance and timing analysis methodology for heterogeneous SoCs and distributed systems. The key novelty of the SymTA/S approach is that it uses intuitive event models from real-time system research to describe the timing behavior the activating events, rather than introducing new, complex stream representations. SymTA/S uses a six-class *standard event models* and a mathematical formalism that easily allow the interfacing and transformations between models [9].

Periodic events or event streams with jitter and bursts are examples of standard models that can be found in literature. The SymTA/S technology extracts the stream parameters from a given schedule and automatically adapts the event stream to the specific needs within these standard models, so that designers can safely apply existing subsystem techniques of choice without compromising global analysis.

The compositional performance analysis methodology defined in SymTA/S [9] alternates local scheduling analysis and event model propagation during system-level analysis. The system analysis is performed iteratively in several global analysis steps. A global analysis step consists of two phases. In the first phase local scheduling analysis is performed for each resource and output event models are calculated. In the second phase, all output event models are propagated. Then, it is checked if the first phase has to be repeated because some activating event models are no longer up-to-date, meaning that a newly propagated output event model is different from the output event models that was propagated in the previous global analysis step. Analysis completes if either all event models did not change during last propagation phase, or an abort condition, e. g. the violation of a timing constraint, has been reached.

The propagation phase requires the modeling of possible timing of the output events of a component. The standard event models allow to specify simple rules to obtain output event models from the parameters of the input event models, considering the scheduling influences.

4. Automotive embedded technology

In the following sections we give a brief overview about the most important scheduling concepts of ERCOSEK operating system and CAN bus protocol. Both arbitration techniques are well established in the automotive industry and deployed in various car platforms and product lines.

4.1 ERCOSEK

The ERCOSEK [3] operating system builds upon the core ideas of static-priority preemptive scheduling. However, this underlying scheduling policy is extended by a variety of additional concepts.

ERCOSEK distinguishes *hardware tasks (interrupts)*, *preemptive software tasks* and *cooperative software tasks*. Software tasks are comprised of processes that are sequentially executed. In contrast to preemptive software tasks, cooperative software tasks are preemptive only at process boundaries. This reduces the context switch overhead but results in additional blocking for the higher priority cooperative tasks.

Certain scheduling-related OS routines can request a considerable amount of execution time at various priority levels. For instance, the *activate task* and *terminate task* routines are called by the OS before and after task execution, respectively. Both OS routines are executed with the so-called *kernel priority*, which is higher than the priority of all software tasks.

Furthermore, task activation can be initiated in a variety of ways. *Time tables* allow the specification of periodic tasks with phase offsets (startup delays) between them. *Alarms* use more dynamic time-out mechanisms and can be issued and disabled at any point in time. Finally, tasks can be activated from software tasks and interrupts (hardware tasks) dynamically and bursty.

4.2 CAN (Controller Area Network)

The CAN bus protocol [2] is also based on static priorities but the message transmission is non-preemptive, as typical for serial line protocols.

Compared to ERCOSEK's characteristics, the actual CAN protocol is relatively simply. However, due to cost reasons, CAN interfaces are typically realized with a very limited number of sender buffers, so-called *message objects*. A message, once written into such a buffer, can not be "overtaken" by another higher-priority message that is generated later. This behavior can turn the static-priority scheme into a complex queuing scheme when it comes to scheduling analysis.

Furthermore, CAN messages inherit the time-table-like behavior of the tasks that generate the messages. In combination with dynamic phase shifts between request and acknowledge frames in an end-to-end path this leads to complex best-case and worst-case scheduling scenarios.

Finally, transmission errors can enforce retransmissions that increase the overall load and message latency.

5. Automotive Example System

Figure 1 shows a SymTA/S model of two electronic subsystems. The two subsystems are functionally independent, and are designed separately by two different electronic suppliers.

The automotive OEM would like to integrate both subsystems in a vehicle.

Note that $ECU1$ is arbitrated by the ERCOSEK operating system, whereas all other ECUs are arbitrated according to the static priority preemptive policy (SPP). The buses used to exchange messages between the ECUs in both subsystems are running the CAN protocol.

The core execution times of the software-functions running on the ECUs are given in Table 1. The communication delays of the messages exchanged over the CAN buses, assuming no concurrent communication requests, are given in Table 2. Tables 1 and 2 additionally contain the priorities of each software-function as well as the IDs of the CAN messages (representing also the priorities). Except for the ERCOSEK scheduled software-functions lower values correspond to higher priorities.

The gray, rounded boxes model the activation of the software-functions (in this case, timers). The activation periods are given in table 3.

Note that both subsystems need to satisfy certain timing constraints in order to function correctly:

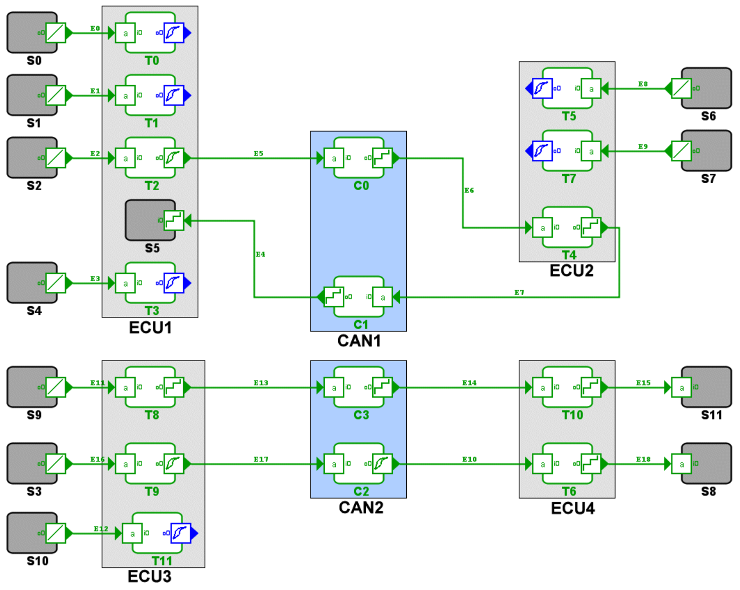


Figure 1. Automotive system example: independent subsystems

Table 1. Computational Tasks

Task	*Execution time*	*Priority*
	ECU1 (ERCOSEK)	
T0	[0.2,0.3]	10 (preemptive)
T1	[0.2,0.3]	5 (preemptive)
T2	[0.1,0.2]	2 (cooperative)
T3	[1.2,2.3]	1 (cooperative)
	ECU2 (SPP)	
T4	[0.7,0.8]	2
T5	[0.1,0.2]	3
T7	[0.1,0.6]	1
	ECU3 (SPP)	
T8	[0.3,0.4]	2
T9	[1,2]	1
T11	[3,5]	3
	ECU4 (SPP)	
T6	[1.1,1.5]	2
T10	[0.3,0.6]	1

– path $S2 \longrightarrow S5$ has an end-to-end deadline equal to 15.
– path $S9 \longrightarrow S11$ has an end-to-end deadline equal to 15.
– path $S3 \longrightarrow S8$ has an end-to-end deadline equal to 15.

Figure 1 shows both subsystems before integration. Both the upper and the lower subsystem are implemented on 2 ECUs (vertical square boxes on the left

Table 2. Communication Tasks

Channel	*Communication time*	*Priority*
CAN1 (Controller Area Network)		
C0	[1.08,1.32]	1
C1	[0.76,0.92]	3
CAN2 (Controller Area Network)		
C2	[0.6,0.72]	4
C3	[0.68,0.82]	2

Table 3. Input Event Models

Input	*Event Model*	*Parameters*
$S0$	periodic	$\mathcal{P}_1 = 1$
$S1$	periodic	$\mathcal{P}_1 = 2$
$S2$	periodic	$\mathcal{P}_2 = 5$
$S3$	periodic	$\mathcal{P}_3 = 15$
$S4$	periodic	$\mathcal{P}_3 = 20$
$S6$	periodic	$\mathcal{P}_3 = 5$
$S7$	periodic	$\mathcal{P}_3 = 7$
$S9$	periodic	$\mathcal{P}_9 = 2$
$S10$	periodic	$\mathcal{P}_{10} = 10$

and right) that exchange messages over a dedicated CAN bus (vertical square boxes in the middle).

Timing and performance analysis of both subsystems reveals the following worst-case delays for the constrained paths:

– 12.01 time units for the path $S2 \rightarrow S5$

– 5.12 time units for the path $S9 \rightarrow S11$

– 9.92 time units for the path $S3 \rightarrow S8$

If we compare these worst-case delays with the deadlines imposed by system specification, we observe that both suppliers implemented their subsystems so that all timing constraints are satisfied.

6. Sensitivity Analysis

The robustness of an architecture to parameter changes is a major concern in the design of embedded real-time systems. Robustness is important in early design stages to identify if and in how far a system can accommodate later changes or updates. Furthermore, system robustness is an important metric in the later design phases during subsystem or third-party component integration. In general, the system robustness is defined by the available headroom or slack corresponding to different properties of the system. These can be task execution demands, channel communication times, the parameters of the activation models, the speed of the computation resources or the throughput of the communication buses.

Sensitivity analysis allows the system designer to permanently keep track of the system robustness, and thus to quickly asses the impact of changes of individual hardware or software components on system performance. Section 6.1 gives a short overview on the sensitivity analysis framework implemented in SymTA/S. In Section 6.2 we determine the robustness of the independent subsystems presented in Figure 1 with respect to variations of different parameters.

6.1 Sensitivity Analysis Framework

The sensitivity analysis framework implemented in SymTA/S combines a binary search technique and the compositional analysis model presented in Section 3. The binary search technique is known as a simple and fast search algorithm used to determine a specific values within an ordered set of data. Since the variations of specific system parameters like execution demands, activation periods, resource speeds have a monotonic impact on the set of system timing properties, the binary search can quickly determine the values of these parameters that leading to conforming system configurations. A detailed description of the sensitivity analysis framework is presented in [8].

Parameters for sensitivity analysis are the system properties that may vary during design process. Very common are variations of execution times, the parameters of the activation models, like period, jitter and offset, communication volumes, bus and processor speeds. The variations of these parameters affect different system performance metrics, like task response times, end-to-end latencies, output jitters, buffer sizes or deadline miss-ratio in case of soft real-time systems.

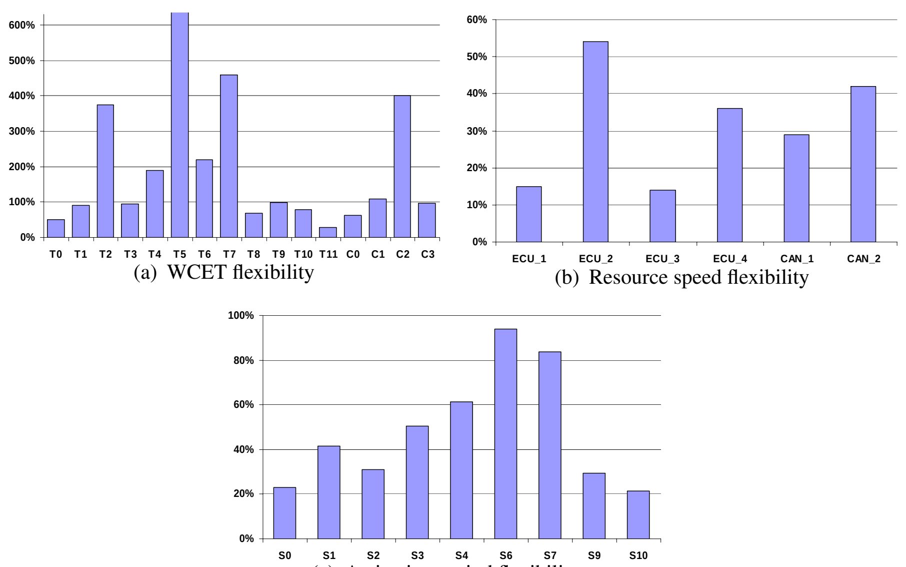
6.2 The Robustness of the Independent Subsystems

In this section we determine the robustness of the independent subsystems presented in Figure 1. Firstly, we investigate the available slacks of the execution times of the tasks mapped on the ECUs and of the communication channels mapped on the CAN resources. The results are presented in Figure 2(a). We observe that tasks $T2$, $T5$, $T7$ and channel $C2$ have a very large flexibility compared with the other tasks and channels. This is explained by the fact that these tasks belong to functional paths without or with loose timing constraints.

Figure 2(b) shows the flexibility of the operational speeds of computation and communication resources. The minimum speed of these resources is determined on one hand by the utilization factor, and, on the other hand, by the timing constraints defined for the tasks executed on these resources.

The last investigated set of parameters are the task execution rates defined at system inputs. Figure 2(c) shows the maximum decrease permitted for task activation periods without violating the set of constraints or the system schedulability.

The resulting slacks of investigated system parameters represent an additional argument for the feasible integration of the two independent subsystems. The available resource headroom allows all messages to be transmitted on a single bus without disturbing too much the performance of the other system components.



(a) WCET flexibility

(b) Resource speed flexibility

(c) Activation period flexibility

Figure 2. Flexibility of the initial system configuration

7. Integrating both subsystems

We now integrate both independent subsystems. Figure 3 shows the system after integration. Instead of utilizing a dedicated CAN bus for each of the subsystem, all messages are now transmitted over a single CAN bus. Of course, this leads to additional load and potentially longer blocking of low-priority messages.

Again we verify performance and timing of the system with SymTA/S, and obtain the following worst-case delays for the constrained paths:

- 20.18 time units for the path $S2 \rightarrow S5$
- 7.92 time units for the path $S9 \rightarrow S11$
- 33.09 time units for the path $S3 \rightarrow S8$

We observe that the constrained paths $S2 \rightarrow S5$ and $S3 \rightarrow S8$ exceed their deadlines by 34.5% and 120.6%, respectively.

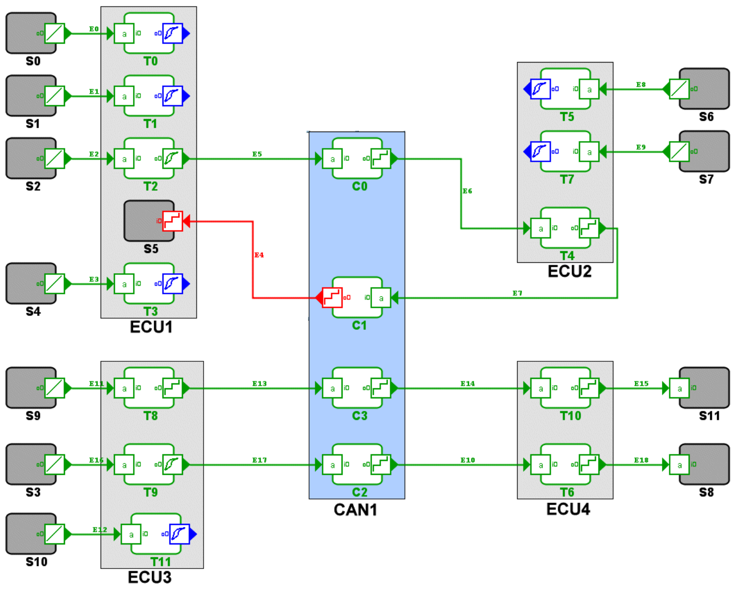


Figure 3. Automotive system example: after system integration

8. System Optimization

In section 7 we have seen that the integration of two independently working subsystems is usually not possible in a straight-forward manner. In practice, system parameters like CAN message IDs, task priorities, and time slots need to be adapted to successfully integrate several subsystems.

In the following we first introduce the design space exploration framework of SymTA/S, assisting the designer in exploring the configuration space of complex distributed systems and pointing out pareto-optimal system configurations with respect to an arbitrary numbers of optimization criteria, including timing properties, power consumption, buffer sizes, etc. In the second part, we use this framework to explore the integrated automotive example system.

8.1 Design Space Exploration Framework

Figure 4 shows the design space exploration framework [5] of SymTA/S. The *Optimization Controller* is the central element. It is connected to the scheduling analysis of SymTA/S and to an evolutionary multi-objective optimizer. SymTA/S checks the validity of a given system parameter set, that is represented by an individual, in the context of the overall heterogeneous system. The evolutionary multi-objective optimizer is responsible for the problem-independent part of the optimization problem, i.e. elimination of individuals and selection of interesting individuals for variation. Currently, we use SPEA2 (Strength Pareto Evolutionary Algorithm 2) [13] for this part, which is coupled via PISA (Platform and Programming Language Independent Interface for Search Algorithms) [1].

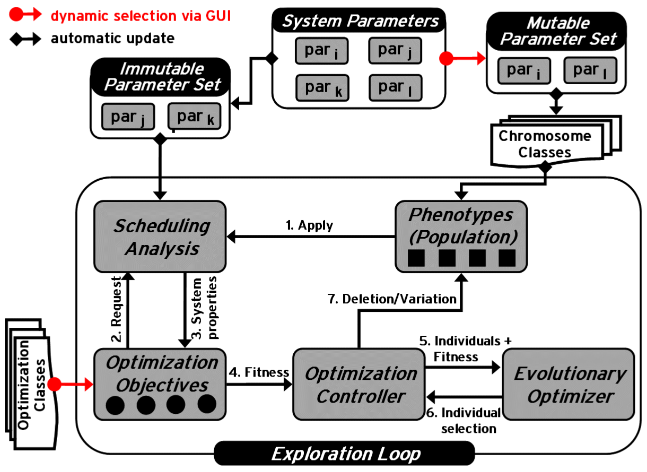


Figure 4. Exploration Framework

Different parameters of a system, such as priorities or time slots, are encoded on separate *chromosomes*. The user selects a set of parameters for optimization. The chromosomes of these parameters form an *individual*, and are included in the evolutionary optimization while all others are fixed and immutable. The variation operators of the evolutionary algorithm are applied chromosome-wise for these individuals.

8.2 Exploring the integrated system

In this section we use the design space exploration framework of SymTA/S to explore the integrated automotive system in Figure 3.

The search space of our exploration consists of the priority assignments on all ECUs as well as the assignment of CAN message IDs on the bus. Since the straight-forward integration of the two subsystems lead to the violation of hard timing constraints, our primary optimization objective is to find a working system configuration. Additionally we are interested in pareto-optimal tradeoffs between the three constrained end-to-end paths.

In the real world, no single design team has full control over the entire system. Instead, numerous design teams from different companies contribute along an automotive supply chain. Each team is in control of only part of the system. Therefore, system-level exploration across team-boundaries is a complicated task. The OEM as the bus integrator, for instance, usually controls CAN message IDs, but has very limited insight into the configuration of the ECUs.

Unfortunately, dynamic behavior between several subsystems usually cannot be observed until late in the design process when first ECU prototypes become available. By that time, it is very costly to re-assign system parameters like priorities of software-functions or CAN message IDs.

In order to account for the difficulty and the high cost of system parameter modifications at system integration time, we add the minimization of parameter changes to the optimization objectives considered during design space exploration. More precisely, we are interested in finding working system configurations with as few parameter changes as possible.

Table 4 shows the pareto-optimal system configurations obtained by an exploration considering 1000 system configurations (40 generations with 25 individuals each). Note that this exploration took approximately 70 seconds on a standard PC running at a clock-rate of 2.4 GHz.

Table 4. Pareto-optimal solutions

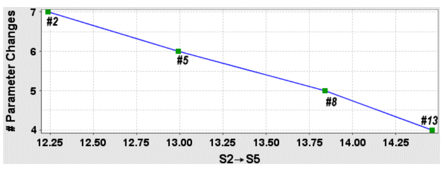
#	$S2 \rightarrow S5$	$S9 \rightarrow S11$	$S3 \rightarrow S8$	*# param. changes*
1	12.24	10.12	11.52	9
2	12.24	13.72	10.72	7
3	12.86	10.12	11.52	7
4	12.99	10.12	8.87	8
5	12.99	13.72	8.07	6
6	13.84	8.42	12.97	8
7	13.84	10.12	6.57	10
8	13.84	12.12	14.57	5
9	13.84	13.82	5.77	7
10	14.06	9.37	12.97	7
11	14.46	9.37	12.97	5
12	14.46	11.07	6.57	7
13	14.46	12.97	14.57	4

We observe that we found 13 pareto-optimal working system configurations, each of them representing an optimal trade-off between the constrained timing properties and the number of parameter changes with respect to the initial configuration.

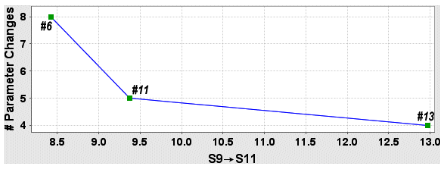
In order to decide which system configuration to adopt, the system integrator needs to interpret the pareto-set. Depending on special requirements to the system she can consider additional information such as system sensitivity to property variations (see Section 9) to make a decision.

Figures 5(a) and 5(b) show the 2-dimensional Pareto-fronts representing the optimal trade-offs between the constrained timing properties and the number of parameter changes necessary to achieve them.

If we consider, for instance, the path $S2 \rightarrow S5$, we can see from the pareto-front in figure 5(a), that the minimum number of necessary parameter changes



(a) Pareto-Front: #Param changes - $S2 \longrightarrow S5$



(b) Pareto-Front: #Param changes - $S9 \longrightarrow S11$

Figure 5. Optimization results

to obtain a working system is 4 (config. $\#13$), corresponding to an delay of 14.46 time units, which is short of the constraint (15 time units).

Increasing the number of allowed parameter changes leads to shorter end-to-end delays. With 5 (config. $\#8$) and 6 (config. $\#5$) parameter changes we can obtain end-to-end delays of 13.84 and 12.99 time units for the path $S2 \longrightarrow S5$, respectively. The shortest end-to-end delay for the path $S2 \longrightarrow S5$, 12.24 time units, can be achieved with a minimum number of 7 parameter changes.

9. Sensitivity Analysis of the Integrated System

In this section we determine the sensitivity of the pareto-optimal system configurations presented in Section 8.2. Figure 6 shows the flexibility of the task execution times and channels communication times. From the set of pareto-optimal configurations we removed those that dominate the other configurations in at least one computed parameter slack. At a closer look we observe that configurations $\#4$ and $\#7$ determine the maximum available slack for most execution times. In general, comparing the results presented in Figure 6 with the results obtained for the two independent subsystems (Figure 2(a)) we observe that the slacks of the execution times of the system tasks have decreased only by a small amount after system integration. The slack of the communication channels has evidently decreased due to the load increase of $CAN1$.

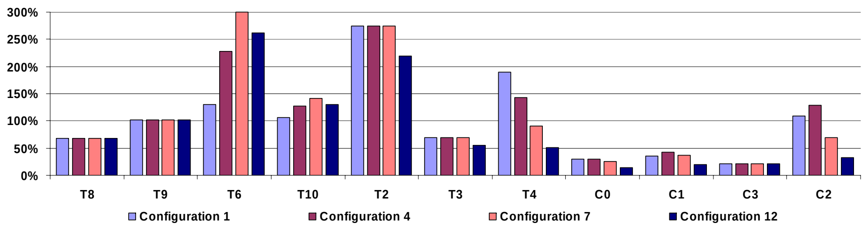


Figure 6. WCET Flexibility

Figure 7 shows the slack values obtained for the hardware resource speed. Again, we selected only those configurations that have a high overall robustness or those which dominate all other configurations in at least one computed parameter slack. Compared to the results presented in Figure 2(b) the only resource with a noticeable smaller slack is the communication bus, $CAN1$. The reason is again the increase of the overall resource utilization after system integration.

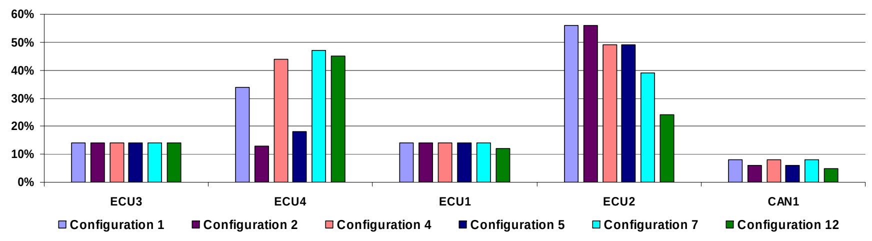


Figure 7. Resource Speed Flexibility

Lastly, we determine the available slack corresponding to the task activation periods. Figure 8 shows the values obtained for some configuration selected from the set of pareto-optimal configurations obtained in Section 8.2. Comparing these results with the results obtained before system integration (Figure 2(c)) we observe that the only periods whose slacks clearly decreased are $S2$, $S3$ and $S9$. Since these periods obviously determine the communication rates of the channels on $CAN1$ and, consequently, automatically the utilization of this bus, and since the overall load on $CAN1$ has increased after subsystem integration, the available headroom of these periods decreased accordingly.

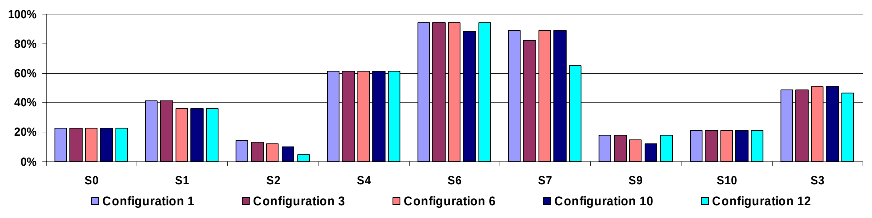


Figure 8. Activation Period Flexibility

10. Conclusion

Formal models are an ideal basis to determine system properties that are not amenable to simulation, such as system robustness, and they allow rapid design space exploration. In this paper we showed how sensitivity analyis can help in quickly dimensioning and optimizing automotive platforms. In the future, formal models based techniques are considered to play a major role in automotive design.

References

[1] 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/.

[2] CAN in Automation (CiA). *Controller Area Network*. http://www.can-cia.org.

[3] ETAS. $ERCOS^{EK}$ *Real-Time Operating System*. http://www.etas.de.

[4] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

[5] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proc. 25th International Real-Time Systems Symposium (RTSS'04)*, Lisbon, Portugal, December 2004.

[6] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS)*, Madrid, Spain, 1998.

[7] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Tenth International Symposium on Hardware/Software Codesign (CODES)*, Estes Park, Colorado, USA, May 2002.

[8] Razvan Racu, Marek Jersak, and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, USA, 2005.

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

[10] SymTA/S - Symbolic Timing Analysis for Systems. http://www.symta.org.

[11] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, Geneva, Switzerland, 2000.

[12] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[13] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.