Halmstad University Post-Print

Towards a tool for derivation of implementation constraints

Mattias Weckstén, Jonas Vasell and Magnus Jonsson

N.B.: When citing this work, cite the original article.

©2004 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Wecksten M, Vasell J, Jonsson M. Towards a tool for derivation of implementation constraints. In: Ninth IEEE International Conference on Engineering Complex Computer Systems, 2004. Proceedings. Piscataway, N.J.: IEEE; 2004. p. 119-127.

DOI: <http://dx.doi.org/10.1109/ICECCS.2004.1310910>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-392>

Towards a Tool for Derivation of Implementation Constraints

P. Mattias Weckstén
Mattias.Wecksten@ide.hh.se

Jonas Vasell
Jonas.Vasell@ide.hh.se

Magnus Jonsson
Magnus.Jonsson@ide.hh.se

*Centre for Research on Embedded Systems (CERES)
Halmstad University, Halmstad, Sweden*

Abstract

An increasing concern in the development of embedded systems is that fundamental design problems often remain undetected until the final tests, after implementation and integration of all components, or maybe even later - at runtime. This is particularly important when it comes to meeting non-functional constraints such as performance or resource utilization requirements. Correcting problems with their sources in design, after implementation, may be very costly as it often requires both redesign and re-implementation. Therefore, much effort has been put into the development of methods and tools that help system designers and developers to detect problems as early as possible during system development. This paper contributes with an addition to that field by presenting and evaluating the practical usefulness of a method that makes it possible to detect problems in system design and dimensioning, even before all components of the system have been fully implemented. Evaluation of the proposed method has been done through 17 different case studies with different characteristics, focusing particularly on real-time latency requirements for tasks on homogeneous single bus platforms. The evaluation indicates a practical method that can be turned into a powerful tool. The presented principles can be extended to wider classes of constraints and systems.

1. Introduction

An increasing concern in the development of embedded systems is that fundamental design problems often remain undetected until the final tests, after implementation and integration of all components, or maybe even later - at runtime. This is particularly important when it comes to meeting non-functional constraints such as performance or resource utilization requirements. Correcting problems, with their sources in design, after implementation may be very costly as it often requires both redesign and re-implementation, and the probability to

make bad design decisions increases as system complexity (the number of interactions and dependencies between different system components and system requirements) increases. Therefore, much effort has been put into the development of methods and tools that help system designers and developers to detect problems as early as possible during system development. The traditional method for handling (in particular) non-functional constraints in embedded system development has been ad hoc. Very little systematic or formal analysis of, for example, real-time properties (such as worst case response time and the ability to meet deadlines) has been made. Such properties have instead often been checked through more or less thorough testing. In recent years, some new analysis techniques have been proposed, which in a more systematic and formal way allow some properties to be predicted. Especially in the real-time case, such analysis often relies on the assumption that some *runtime constraints* (RTC) (e.g., individual task deadlines) are enforced. A well-known example of a real-time analysis technique is Rate Monotonic Analysis (RMA), first described in [1]. In conjunction with these analysis techniques, systematic methods to derive the required runtime constraints have also been proposed, see [2][3]. These new means of predicting whether a system will meet its constraints are helpful in that they make it possible to avoid runtime problems that can be associated with very high costs or severe safety concerns. They may also be of help when it comes to pointing out where a problem hides in the design or implementation of the system. However, in general, these methods typically depend on a finished implementation of a system's software components by, for example, requiring that the worst case execution time (WCET) of all components is known. This means that fundamental design problems still can remain undetected until all components have been implemented. Since no implementation has been done when the design is decided, it would be useful to have a design-time method to derive guidelines, or budgets, for the implementation of each task, such that keeping within those budgets will maximize the proba-

bility that the overall system requirements can be met. In that way, failure to keep a budget will be an immediate warning; meaning that further component development should be postponed until any necessary modifications of the design and/or budget assignments has been made. Budgets may also serve as guides for the way components are implemented or compiled by suggesting trade-offs between, for example, execution time and memory consumption. The *budgets* are referred to as *implementation-time constraints* (ITCs). This should not be confused with the RTCs mentioned above. In this paper, we both present a method for ITC generation and the results of a preliminary evaluation of a prototype implementation. The prototype includes the method in a software tool to determine whether the method may be practical. The suggested approach for ITC generation is to make a design space exploration and optimize the solutions with respect to the relative amount of end-to-end work compared to the end-to-end constraints. The validation of the method for small and medium sized systems shows that generation of ITCs is possible in a practical way in limited time, based on relative estimates of the execution time and end-to-end timing constraints. If an ITC for a task does not hold, there is a set of implementation directives to choose an alternative ITC from. If there is no budget in the set that holds, it is possible to rerun the budgeting method, using updated execution time estimates. Our solution is in some central parts inspired by the real-time budgeting algorithm described in [4] which focuses on how to derive constraints or budgets for implementation based on end-to-end latencies and how to provide a measure of the "flexibility" of each budget, the tightness. In [5], a framework for a systems engineering method is presented, focusing on keeping the correctness at all steps of the formal process from specification and dimensioning to implementation. There are a number of methods for real-time constraint derivation, such as the method described in [6], but focusing on the implementation validation rather than design. Another closely related area is that of the different system co-design methods that are presented in [7] [8]. The main focus of co-design methods is to simultaneously generate hardware architecture and the software that runs on it, where the architecture may consist of a mix of programmable CPUs, DSPs, and ASICs. The rest of the paper is organized as follows. In Section 2 the problem statement is presented. In Section 3 the budgeting method is presented in more detail, along with a step by step example and method improvement suggestions. Finally, Section 4 presents the evaluation of the budgeting method, followed by the conclusions in Section 5.

Table 1. The properties for a typical task graph. Each task, M_v , has an estimated execution time, v_e . The end-to-end constraints are noted as offsets, v_o , and deadlines, v_d , respectively.

M_v	v_o	v_d	v_e	Graph
n-0	0	?	25	
n-1	10	?	40	
n-2	?	?	50	
n-3	?	?	20	
n-4	?	150	50	
n-5	?	100	25	

2. Problem statement and assumptions

The problem is to find a method that, based on a functional system description with end-to-end constraints and a given platform, generate a set of task level ITCs (which optimally will maximize the implementation success probability). The functional system description is in the form of a directed acyclic task graph, where the designer has estimated the execution time for each task in the graph, in relative terms. Each task can have optional timing constraints (i.e., in the form of release time and/or deadline) and optional locality constraints limiting the possible processor allocations. For each precedence constraint of a task, an optional communication package of estimated relative length can be defined. An example of what this would look like is seen in Table 1. The platform is modeled as a network with a number of processors connected to it. In this paper, we assume homogeneous processors connected to a single bus, meaning that the maximum allowed number of processors is the only platform parameter. The motivation for this restriction is that although the problem is somewhat simplified, it will still be enough to validate the usefulness of the method. The output from the method is a set of budgets where each budgets in the set consists of ITCs for each task and the associated communication tasks. Each budget in the set is optimal or close to optimal in the sense that the lowest ITC in the specific budget is searched. In this paper, ITCs are limited to include execution time and processor allocation constraints. It is also assumed that the deadline for a task is shorter than the task's period.

3. The budget generation method

To be able to find optimal or near optimal budgets it is necessary to search through all possible processor allocations and execution orders (schedules) for tasks according to the constraints. The reasons for checking all

Figure 1. Flowchart for the suggested implementation of the method. The schedules are turned into budgets and the n best are kept.

possible schedules are that (i) execution order and allocation are the two parameters that affects the optimization metric, and (ii) that we would like to know if we, after implementation, will be able to generate a schedule for the system that enables us to make guarantees about whether it will be possible to fulfill the end-to-end constraints. An overview of the method is presented in Figure 1, further described below. Initial data formatting is done by the preprocessor that generates constrained task graphs, based on the input data (resource parameters, performance restrictions, and task set). The generation of schedules (scheduler in the figure) for the constrained task graph consists of two steps, ordering and allocation. The purpose of the ordering step is to generate execution orders of the tasks according to the precedence constraints in the task graph. As part of the allocation, communication tasks are generated. The al-

location of the tasks in the task graph is quite straight forward, generating allowed allocations according to the locality constraints. When allocation is decided, communication tasks can be generated based on the information about which tasks that have to communicate using the bus. Since the number of possible schedules is quite large and scales quite bad with the number of tasks in the task graph, just a plain search is not good enough. Several different heuristics are therefore evaluated (see Section 4.1). Finally, ITCs (budgets) are generated based on the scheduled task graphs. The top n budgets with minimal tightness are saved in the set of budgets that is the out-data from the budget generation method. To calculate the optimization metric, the budget tightness T , for the scheduled task graph, G (of tasks, V , and precedence constraints, E) the tightness for each path, $p \in P$, in the graph has to be found. As mentioned earlier, for each task there is an estimated execution time V_e . For (at least) the first and last task on the path, there is also a corresponding offset V_o and deadline V_d , respectively. The tightness for a path p_T , $p \in P$ is equal to the fraction between the work W , that is the sum of the task's estimates, and the path length L , that is the difference between the path's deadline and offset. The path's offset is equal to the first task's offset; while the path's deadline is equal to the last task's deadline. In summary we have the following definitions:

$$G = \langle V, E \rangle \quad (1)$$

$$W = \sum_{v \in p} v_e \quad (2)$$

$$L = p_d - p_o \quad (3)$$

$$p_T = \frac{W}{L} \quad (4)$$

$$T = \max(p_T : p \in P) \quad (5)$$

When the tightest path has been found, the ITCs for the tasks on this path can be generated. The sum of the ITCs for the tasks on the tightest path shall fill out the whole length L of the path. This is done by, for each task on the path, dividing the estimated execution time, v_e , with the path tightness, p_T , giving us the allowed execution time (AET):

$$v_{AET} = \frac{v_e}{p_T} \quad (6)$$

The tasks on the tightest path may now be removed since they all have been given implicit offsets and deadlines (to be used as implementation budgets). One way to calculate the offset and the deadline explicitly for a task on the tightest path is to set the offset for a task to the previous task's deadline, and the deadline to its task's offset

plus its AET:

$$v_{i_o} = v_{(i-1)_d} \quad (7)$$

$$v_{i_d} = v_{i_o} + v_{i_{AET}} \quad (8)$$

Since all tasks on the tightest path have explicit offsets and deadlines, it is possible to remove them from the task graph, replacing them with offsets and deadlines. This means that all parents of the task we are going to remove will get new deadlines equal to the removed task's offset. Similarly, all the children of the task we are going to remove will get new offsets equal to the removed task's deadline. When all tasks on the tightest path have been removed from the task graph, the second tightest path is calculated and the corresponding AETs generated. This procedure is then repeated until all tasks have been assigned AETs. Note that only the first tightest path's tightness is of interest when describing the tightness of the whole task graph budget. The consecutive paths' tightnesses are just calculated to be able to generate all tasks' AETs. Although the implementation space (success probability) is maximized in some sense, using the global tightness as optimization metric, there is no guarantee that the implementation will be possible. To avoid the need to start the whole budgeting process over, just because a certain budget did not hold, multiple promising budgets are generated and kept for later use. Below, we present an example of how the method (in Section 3.1) is used and improvements of the method (in Section 3.2).

3.1. Example

To illustrate how budgets for a task graph are generated, we calculate the ITCs step by step for the task graph in Table 1. In this example, tasks M_{n-0} , M_{n-2} , and M_{n-4} are allocated on processor 1, while tasks M_{n-1} , M_{n-3} , and M_{n-5} are allocated on processor 2. The execution order on each processor is (in this case) given explicitly by the precedence constraints in the task graph. However, since the two communicating tasks M_{n-0} and M_{n-3} are allocated on different processors we have to create a communication task $C_{n-0,n-3}$, allocated on the communication processor (the bus seen as a processor). Let us assume that, according to the specification, the new task $C_{n-0,n-3}$ has the estimated communication time 5. To calculate the tightness and assign AETs for all tasks we have to find the tightest path in the graph. There are three different paths in this example, $p_{n-0,n-4}$ with the total work of 125 and the length 150, $p_{n-0,n-5}$ with the total work of 75 (including the communication task $C_{n-0,n-3}$) and the length 100, and $p_{n-1,n-5}$ with the total work of 85 and the length 90. The tightness is calculated for each path (0.667, 0.75,

Table 2. The tightest path has been found. The end-to-end constraints are derived into AET's, v_{AET} , for each task on the path.

M_v	v_o	v_d	v_e	v_{AET}	Graph
n-1	10	52	40	42
n-3	52	73	20	21	
n-5	73	100	25	27	

Table 3. The second tightest path has been found. The end-to-end constraints are derived into AET's.

M_v	v_o	v_d	v_e	v_{AET}	Graph
n-0	0	30	25	30
n-2	30	90	50	60	
n-4	90	150	50	60	

and 0.944, respectively), which gives that $P_{n-1,n-5}$ is the tightest path with a tightness of 0.944.

Since the calculations show that the path $\{M_{n-1}, M_{n-3}, M_{n-5}\}$ is be the tightest it will also be the first to be assigned AETs (see Table 2). The assigned tasks are then removed from the task graph and replaced with release times and deadlines where this applies. In this case it means that the communication task $C_{n-0,n-3}$ will get a deadline of $100 - 27 - 21 = 52$. The tightness calculations are now repeated for the remaining tasks in the graph. The path $\{M_{n-0}, M_{n-2}, M_{n-4}\}$ is now the tightest and therefore assigned AETs (see Table 3). The tasks is then removed from the task graph and replaced with a release time for $C_{n-0,n-3}$ that will start at $0 + 30 = 30$. Now there is just one task left ($C_{n-0,n-3}$) that have a release time and a deadline, which makes the generation of it's AET trivial ($52 - 30 = 22$). All the tasks in the graph have now been assigned AETs (see Table 4) and the tightness for this graph is equal to the tightness for the tightest path (0.944).

3.2. Improvements

Since the search problem scales very quickly with the size of the task graph, the search space needs to be reduced using some kind of heuristic (the problem

Table 4. All tasks have been assigned AET's. Note the generated communication task (labelled *C*).

M_v	v_o	v_d	v_e	v_{AET}	Graph
n-0	0	30	25	30	
C	30	52	5	22	
n-1	10	52	40	42	
n-2	30	90	50	60	
n-3	52	74	20	21	
n-4	90	150	50	60	
n-5	74	100	25	27	

is similar to multiprocessor allocation, which is an NP-hard problem).

To reduce the number of possible orders during the scheduling, explicit ordering constraints are added where it is identified that two tasks has an implicit execution order, based on estimated execution times and end-to-end timing constraints - or in other words, avoid unschedulable task orders.

Allocation suffers similar problems. It makes no sense to add tasks to a processor that will get more than 100% utilization. To achieve a reduction of possible allocations, an optional maximum utilization level is introduced.

Since the estimated execution time is supposed to be just a rough estimate it would be more reasonable to assign the allowed execution time proportional to the uncertainty in the estimate also instead of only being based on the size of the estimate. This could be done by a simple transformation of the tightness metric and will be evaluated in future work.

4. Evaluation

As described in the introduction, the method for ITC derivation is intended to be used as part of the system design process in the form of a tool that can be used before and during the implementation of software components. It is therefore important that the method can be implemented in a way that is useful in practice, and that the budget, or ITC set, it produces will give good support to the implementation process. In this section we present an evaluation of the tool's practical usefulness.

The main concern regarding the practical usefulness of the method is the fact that optimal ITC derivation encompasses a very extensive search which increases rapidly as the system becomes more complex. The extent of the necessary search translates directly into the amount of time it takes to find a sufficiently good solution. The main purpose of the evaluation presented here is thus to determine how close to an optimal so-

lution it is possible to come within a time that is short enough to make the tool useful as a provider of quick feedback to a system designer about the feasibility of a certain design choice. We also present an analysis of the efficiency of a number of different search strategies. The evaluation presented here is based on a prototype Java implementation of our ITC derivation method. This means that there is much room for further improvements of the tool's practical usefulness. We have then applied the tool on a number of example cases found in the literature. Using this prototype and the set of test cases, we have addressed three questions:

- What is the optimal solution in each case? In many cases, the search space has been so large that we have had to settle for approximations based on the best solutions found during a very long run.
- Is our method practical? This translates into the question whether it is possible to find a solution in a short time (in a few minutes).
- How do some possible search heuristics compare?

Our experiments also give some insight into how the problem scales with the complexity of the system design it is applied to. The rest of the section is organized as follows. Section 4.1 describes the evaluation methodology. In Section 4.2 the results from the evaluation are presented. In Section 4.3 the presented results are compared, motivated and explained.

4.1. Methodology

To evaluate the method, a number of cases (i.e., task graphs with different constraints) have been entered in the budgeting tool. The test cases are a mix of previously known and tested systems from the literature and constructed systems, all of various sizes and with different parameters, as seen in Table 5. Three different search methods have been applied for each of the cases, using three different experiment durations. Since the only way (found so far) to find the optimal tightness for a task graph is to search through the whole design space, we have to make approximations about the real optimum. The approximation method used in this evaluation is to make a search that lasts for relatively long duration of time (10 hours) and assume that a near optimal solution will be found. The different search heuristics evaluated in the tests are the following:

Linear Linear search is performed by picking one possible solution after another. Eventually (too long to wait for) all possible solutions have been evaluated.

MLinear Multi-linear search is really a number of linear searches, all with different start points. In our

Table 5. The table shows, for each case, the number of tasks, $|V|$, and the maximum allowed number of processors to allocate the tasks on, P_{max} .

Case	$ V $	P_{max}
tind	48	10
combination	43	8
simple_control	28	3
reconfigurable	25	4
radar_matrix	12	3
base_station (v1)	17	3
base_station (v2)	17	3
base_station (v3)	8	3
simple_reg (v1)	15	4
simple_reg (v2)	15	4
safety_critical	12	3
multi_reg	12	4
simple_reg (v3)	10	4
slack	10	5
neural_net	8	3
base_station (v4)	8	3
small	5	2

specific evaluation we also apply pruning to this group of searches, we search for a period of time and then prunes away the worst solutions so far, giving more time for the other searches.

Random Random search makes point hits in the search space during the search. This results in many quite different, independent, solutions.

The experiments are divided into three run-time classes, each of which the classes with three, three, and one search method respectively, meaning that each case has been simulated seven times with different parameters. The three run-time classes are the following:

Long Three long experiments that were allowed to run for 10 hours (i.e., one experiment for each heuristic).

Short Three short experiments that were allowed to run for 1 hour (i.e., one experiment for each heuristic).

UShort One ultra-short experiments that were allowed to run for 5 minutes (i.e., for the random heuristic).

The two run-time classes long and short contains three different parameter sets for the experiments; linear search, multi-linear search with random start points and pruning, and multiple random single shot search. The experiment using the short random search method is repeated nine times to get some information about the

Figure 2. The theoretical tightness limit for the different cases. The values are normalized to the minimum tightness in the long experiments (i.e., the estimated optimal tightness), for each case.

average outcome since all random searches will give different results for each run and this will be especially remarkable in the short simulations.

The purpose of the tests has been to prove the usability for our algorithm, why the collection of examples reflects a wide variety of typical problem types and sizes. Details about the cases and their properties are found in [9]. Design cases of many types are part of the set, ranging from data intensive radar computations to distributed control systems.

4.2. Results

From the experiments, we have results for the different search heuristics, for different experiment run-times. For comparison, each case is normalized relative to the minimal achieved tightness (over all experiments, for each case) and sorted descending according to number of tasks in the graph (without communication nodes). To get an indication about how close to optimum the minimum tightness for a case is, the theoretical limit for the tightness has been calculated. The limit is calculated as the tightness for the case where all tasks are allocated on different processors (maximum parallelism) but do not invoke any communication (no penalties). As seen in Figure 2, the minimal tightness is close to the theo-

Figure 3. The results from the 10 hour experiments, using all the different search methods. The values are normalized to the minimum tightness in the long experiments (i.e., the estimated optimal tightness), for each case. The linear and multi-linear search method did not find a feasible solution for the *reconfigurable* case.

retical limit in most cases and thus must be to optimal. In a few of the cases the minimal tightness is not close to the limit, and in these cases we cannot say whether they are close to optimal or not. As seen in the Figure 3, the different search heuristics shows quite small deviations, and in most cases the random method outperforms the linear and multi-linear method. Closer examination of the cases where the random method performs worse than the two other methods has shown that these cases are strictly constrained, which makes it hard to find feasible solutions at random. As seen in Figure 4, the ten times shorter experiments show about equally good results for the random method, and slightly worse results for a few of the experiments using the linear and multi-linear method. Finally, when the results are compared for the experiments using the random method for different duration (as seen in Figure 5), it is seen that the increase in duration does not affect the tightness in more than two experiments.

To summarize, the figures show that the tightness for the short experiments are quite close to the tightness for the long experiments. It is clearly shown that the completely random search method is the best choice in most cases. It is even shown that the very short random searches are as good as the longest ones.

4.3. Discussion

As system size and complexity grows, the simple linear search may not be efficient enough to handle the problem size and some sort of search heuristic may be needed. Although simulated annealing and genetic algorithms are used for similar problems they do not fit this problem since we have to generate a schedule for the system before we can evaluate the tightness. Neighborhood search on the other hand seems to be a useful method to apply. With a ordered allocation (schedule) present it will be an easy task to make minor changes to the order or allocation and evaluate these neighbors.

Another interesting question is how meaningful the results would be if the input estimates are uncertain anyway? One could guess that they would be quite useful anyway since even if implementation budgets do not hold we get a quick feed back if something is wrong and we could reiterate or maybe pick another budget. In the case where implementation budgets are not used, problems like this will be noticed first at the scheduling of the system, which could be too late.

5. Conclusions

In this paper we have argued for the usefulness of a tool that helps designers and implementers of embed-

Figure 4. The results from the 1 hour experiments, using all the different search methods. The values are normalized to the minimum tightness in the long experiments (i.e., the estimated optimal tightness), for each case. The linear and multi-linear search method did not find a feasible solution for the *reconfigurable* case.

Figure 5. The results from the experiments using the random search method for different durations. The values are normalized to the minimum tightness in the long experiments (i.e., the estimated optimal tightness), for each case.

ded systems to make early judgments about the possibilities to meet the system's non-functional constraints, thus making it possible to reduce the costs and risks in system development. We have been able to present some results showing that it is possible to turn it into a tool that is practical in many cases. This practicality has primarily been measured by investigating whether near optimal budgets or ITCs can be found in reasonable time. However, investigations also need to be made regarding how much implementation effort can be saved by using derived ITCs as implementation guidelines and to detect design problems in early development phases. Preliminary results of investigations in that direction indicate that much unnecessary implementation effort can be saved, especially in cases with tough end-to-end constraints and very limited resources, but further investigations (preferably including real, documented cases) are necessary to reach conclusive results. Given the method's practical usefulness, we see several possible extensions and improvements that are possible to investigate further. One remaining question is whether the method will be practical for even larger and more complex systems too. The experiments we have performed indicate that the method we have proposed is practically useful in the sense that near optimal results may be found in short time, independent of system size or complexity. To prove this, further analysis of the methods complexity will be done. Some of the future extensions of particular interest are how to handle memory budgets (using a method more or less identical to the method used for derivation of time budgets), how to handle tasks with different periodicity, and how to handle the problem with the quickly increasing search space. Since the search problem get quite large, even for moderate sized systems, we would like to develop more efficient search methods to speed up the search and add search heuristics to reduce the search space. We have developed, and with experiments proven the usability, of a new design method taking non-functional constraints in account. The guidelines derived, using the tool, can be used to verify the design before implementation is started and to guide the process during the implementation phase, guaranteeing that there will be at least one feasible schedule for the implemented system. The tests so far show that the implementation budget derivation method is a useful tool that manages the complexity of non-functional constraints for a wide variety of projects.

Acknowledgements

We would like to thank professor Bertil Svensson for his valuable comments on this work.

References

- [1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [2] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proc. IEEE Real-Time Systems Symposium*, 1994, pp. 216–227.
- [3] M. Saksena and S. Hong, "Resource conscious design of distributed real-time systems – an end-to-end approach," in *Proc. IEEE International Conference on Engineering of Complex Computer Systems*, Oct. 1996, pp. 306–313.
- [4] Y. Zhao, "Derivation of local timing constraints in early design stages," Master's thesis, Chalmers University of Technology, Apr. 1999.
- [5] G. Le Lann, "A methodology for designing and dimensioning critical complex computing systems," in *Proc. IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, Mar. 1996, pp. 332–339.
- [6] K. Kuchcinski, "Embedded system synthesis by timing constraint solving," in *Proc. 10th International Symposium on System Synthesis*, Sept. 1997, pp. 50–57.
- [7] G. de Jong, "A UML-based design methodology for real-time and embedded systems," in *Proc. of Design, Automation and Test in Europe*, 2002, pp. 776–779.
- [8] J. Plantin and E. Stoy, "Aspects of system-level design," in *Proceedings of the 7th International Workshop on Hardware/software Codesign*. ACM Press, 1999, pp. 209–210.
- [9] M. Wecksten and J. Vasell, "System design problems and case study definitions," Halmstad University, Technical Report IDE0349, Sept. 2003.