Verification of Printer Datapaths using Timed Automata*

Georgeta Igna and Frits Vaandrager

Institute for Computing and Information Sciences
Radboud University Nijmegen, the Netherlands
{g.igna,f.vaandrager}@cs.ru.nl

Abstract. In multiprocessor systems with many data-intensive tasks, a bus may be among the most critical resources. Typically, allocation of bandwidth to one (high-priority) task may lead to a reduction of the bandwidth of other tasks, and thereby effectively slow down these tasks. WCET analysis for these types of systems is a major research challenge. In this paper, we show how the dynamic behavior of a memory bus and a USB in a realistic printer application can be faithfully modeled using timed automata. We analyze, using Uppaal, the worst case latency of scan jobs with uncertain arrival times in a setting where the printer is concurrently processing an infinite stream of print jobs.

1 Introduction

Modern embedded systems are characterized by distributed implementation platforms that include a heterogeneous mix of several processors, one or more buses for communication, and a variety of sensing and actuating devices. They have to operate in dynamic and interactive environments, and need to carry out a mix of data-intensive computational tasks and event-processing control tasks. Not only functional correctness is important, but also quantitative properties related to timeliness, quality-of-service, resource usage and energy consumption. The complexity of embedded systems and their development trajectories is thus increasing rapidly. At the same time, development trajectories are expected to deliver products that are inexpensive and performing, while meeting stringent time-to-market constraints. The complexity of the designs and the constraints imposed on the development trajectory dictate a systematic, model-driven design approach that leverages reuse and is supported by tooling whenever possible.

In multiprocessor systems with many data-intensive tasks, a bus may be among the most critical resources, and severely degrade the timing predictability. The problem is that allocation of bandwidth to one (high-priority) task may lead to a reduction of the bandwidth of other tasks, and thereby effectively slow down these tasks. If we do not want this to occur, for instance in the case of

* Research supported by the Netherlands Ministry of Economic Affairs under the Bsik program within the Octopus project, and by the European CommunityâĂŹs Seventh Framework Programme under grant agreement no 214755 (QUASIMODO).

safety critical systems, then we may use e.g. a time division multiple access (TDMA) strategy on the buses in order to give each task a guaranteed bandwidth. However, for most systems such a solution is too expensive. According to Williams et al. [1], for the foreseeable future off-chip memory bandwidth will often be the constraining resource in system performance of multicore computers. Clearly, WCET analysis for such systems is a major research challenge. Existing performance analysis techniques are not able to accurately predict WCETs for systems with this type of highly dynamic resource behavior. Simulation of detailed models certainly provides insight, but fails to provide WCETs in settings with uncertain job arrival times, dynamic and interactive environments and/or uncertain processing times.

In this paper, we show how the dynamic behavior of a memory bus and a USB in a realistic printer application can be faithfully modeled using timed automata. In addition, we show how to compute WCETs (latencies) for the application using the model checker Uppaal [2–4]. The case study that we describe here originates from the Octopus [5] project. Octopus is a cooperation between Océ Technologies, the Embedded Systems Institute and several academic research groups in the Netherlands. Its objective is the development of new methods and techniques to support model-driven design space exploration for embedded systems. Some preliminary work from the Octopus project was reported in [6]. There, we considered a simplified version of an Océ printer architecture. Using this architecture, we studied the differences among three modeling formalisms and supporting tools used in the project: Uppaal [2, 4, 3], Colored Petri Nets [7, 8] and Synchronous Dataflow Graphs [9, 10]. In this paper, we present a detailed model of a realistic printer design which, in particular, includes a description of the scheduling rules used by the Océ printer controller.[1] We analyze, using Uppaal, the worst case latency of scan jobs with uncertain arrival times in a setting where the printer is concurrently processing an infinite stream of print jobs.

The purpose of this paper is to show that the Uppaal model checker can handle the complexity of dynamic memory bus behavior in a realistic model of a complex industrial application. To the best of our knowledge, no other analysis technique/tool, except maybe the hybrid method of [11], is currently able to do a performance analysis for this type of systems (involving a dynamic memory bus and uncertain arrival times). Existing techniques for WCET analysis of distributed embedded systems, such as Modular Performance Analysis [12, 13], SymTA/S [14] and MAST [15] are not applicable since they lead to overly conservative analysis results. In [11], a hybrid method is proposed for analyzing embedded real-time systems that integrates modular performance analysis and timed automata. It would be interesting to use our detailed Uppaal models of the memory bus and USB as part of this hybrid method.

[1] For reasons of confidentiality, we have changed resource names, some other details and all the numerical data in our model. As a consequence, the outcomes of our analysis do not apply to any design of Océ. However, as part of the project we have succeeded to carry out a similar analysis for an actual Océ design.

The structure of this paper is as follows. The next section introduces the printer case study. In Section 3, the timed automata models are described. The analysis results is detailed in Section 4. Concluding remarks and discussions of future work follow in Section 5.

2 Case Study

The hardware architecture analyzed in this paper is depicted in Figure 1. A user can utilize this machine for copying, scanning or printing. He can either use paper or digital files. In case of paper, he must first scan it. With digital files, he can connect to Data Store both remotely and locally via USB. The upload and download through the USB behave differently depending on the bus usage: if the transfer is unidirectional, it is faster than when it is bidirectional. A user has a large variety of image processing (IP) options like zooming, rotation, or filtering, etc. Depending on these preferences, there are different components needed to process the files. A *datapath* is the complete trajectory of an image data from source (i.e. Scanner) to target (i.e. Printer). The performance of the various datapaths is of critical importance in the Océ printer design.

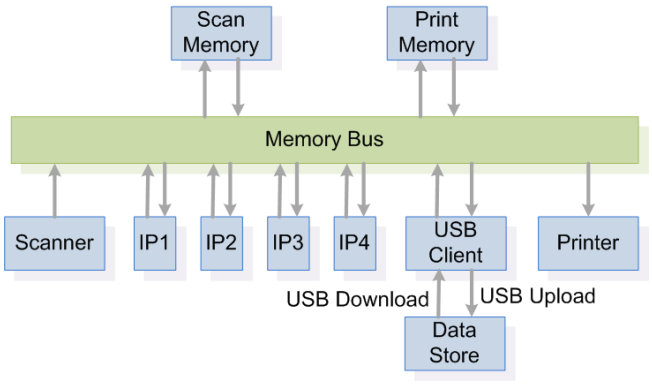
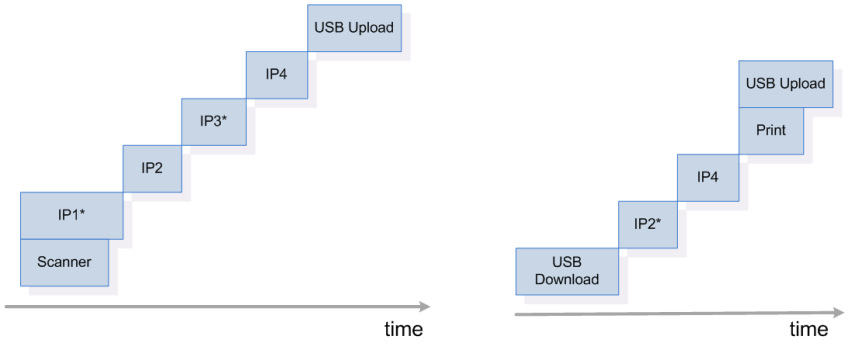


Fig. 1: An Océ Printer Architecture

Here we analyze two common datapaths[2] (Figures 2a and 2b). Since they are often used in practice, it is important to see what can happen in the worst case. In the figures, we can also observe the dependencies among the resources used. They are of three types. In the first category, two resources end in the same time (see Scanner - IP1). The second refers to the sequential dependency between two resources (e.g IP2 - IP3). The last case is the parallel execution between Upload and Printer.

[2] The resources marked with * are optional



(a) Scan to Email (b) Print From Docbox

Fig. 2: Datapaths

A user specifies his input in the form of a job. A *job* is a tuple made of an input file, a datapath and some image processing settings. We use the term *scan job* for a job which uses the Scan to File datapath and *print job* for a job which uses the Print from DocBox datapath. In addition, the scan jobs utilize Scan Memory and print jobs only Print Memory. These memories limit the number of concurrent images in the system.

The machine uses specific scheduling rules to solve the conflicts that may occur among the concurrent jobs. We present here the most important ones, which we have also implemented in our model.

The first rule is *files non-overtaking*: files that use the same datapath are processed in the order they enter the system.

The second rule is referred to as *bus throttling*. Memory bus is shared by all the resources when they transfer data to memories. Each resource claims different percentage of the memory bus, but the maximum bandwidth available is not enough for all the resources. Further, the execution time of a resource is limited by the bandwidth it occupies, the internal processing time being negligible. Therefore, a good bandwidth management is important for improving system's performance. When more than one job occupies the system, often incoming jobs do not have enough bandwidth available to start. In such situations, the Océ bus throttling rule is applicable. It uses a priority list to solve the bus conflicts (Figure 3). The resources at the top of the list have the highest priority. If a resource with high priority needs some bandwidth to process a job, it receives it. This potentially enforces a reduction of the bandwidth used by other running resources with lower priority, such that the total bandwidth used does not exceed a maximum (Algorithm 1). For the case when the resource has low priority, it gets the difference between the maximum bandwidth allowed and the current bandwidth used (line 11). Similarly, when one resource with high priority has finished a job, it releases the bandwidth, which is then redistributed back to other running resources based on their priority level (Algorithm 2). The bandwidth redistribution is reflected in the resource processing speed. As mentioned

above, the speed of a resource directly depends on the bandwidth used. This means that, whenever its bandwidth is modified, the expected completion time for the current job is also changed. From another perspective, this rule induces dynamic behavior in the Océ machines, which is not easy to predict.

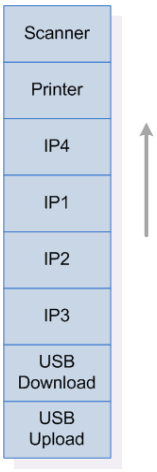


Fig. 3: Resource priority order

bandwidth allocation algorithm 1 void allocateBandwidth(resource r, bw_claimed)

```
1: bandwidth(r)=bw_claimed;
2: bandwidth available− = bandwidth(r);
3: if (bandwidth available ≤ 0 then
4:    if there are resources with lower priority in the priority list then
5:       repeat
6:          take the resources from the bottom up
7:          adjust their bandwidth
8:       until the bandwidth available gets greater or
9:       equal to 0 or we arrived at resource r
10:   else
11:      bandwidth(r)=-bandwidth available;
12:      bandwidth available = 0;
13:   end if
14: end if
```

The third rule is 'upload in order'. The USB client is one of the slowest resources in the system. Therefore often many jobs wait for uploading and they

bandwidth deallocation algorithm 2 void deallocateBandwidth(resource r, bw_claimed)

1: $available\_bandwidth+ = bandwidth(r)$
2: bandwidth(r)=0;
3: **if** there are resources with lower priority in the priority list and their bandwidth used is less than the maximum bandwidth that they can use **then**
4: **repeat**
5: take them from top down
6: increase their bandwidth up to their maximum
7: **until** the bandwidth available gets equal to 0
8: **end if**

should be served in order. A list is used for keeping track of the waiting jobs. There is a strict rule when jobs are added to it. A scan job is inserted after the scanner has completed it, whereas the jobs which use the Print from Doc Box datapath are added after IP4 has processed them. The same also happens when IP2 is shared but in this case we add scan jobs after scanning and print jobs after downloading.

The next rule modeled refers to the conflicts between the scan and the print jobs in case of shared resources. Whenever a scan and a print job claim a resource simultaneously, the print job gets priority.

Finally, we assume that all the resources in the system are non-lazy. This means that if a resource is available when a job claims it, it should be granted immediately. This restriction greatly reduces the complexity of the control software, and also the state space.

3 Model description

The timed automata model is structured as follows. Each resource is described by a specific automaton (Figure 4), except the two memories and the memory bus, which are simply modeled as shared integer variables. A resource stays in the IDLE location until is claimed by a job. When a job grabs it, the resource computes the bandwidth it can use and its processing speed, applying the bus throttling rule when needed. Then, the resource stays in the RUNNING location until either the job is processed or its speed is changed. For the latter case, the transition between RUNNING and UPDATE_WORK is urgently taken. On the transition back to the RUNNING state, the current job's data is updated. First we under-approximate the clock which monitors the execution time of the job to the closest integer lower or equal to the clock value (select statement: $i:int[0,max\_exec\_time]$ and guard: $i \leq x$ && $i+1>x$). Then the current job's unprocessed data is computed. Using this resource template, the model is accurate. However, the state space is fragmented with every speed change, which is a problem for scalability. All the resources, except for the Printer and the Scanner, use this template. The Printer and the Scanner are never interrupted after they start. Therefore, they do not need the UPDATE_WORK location.

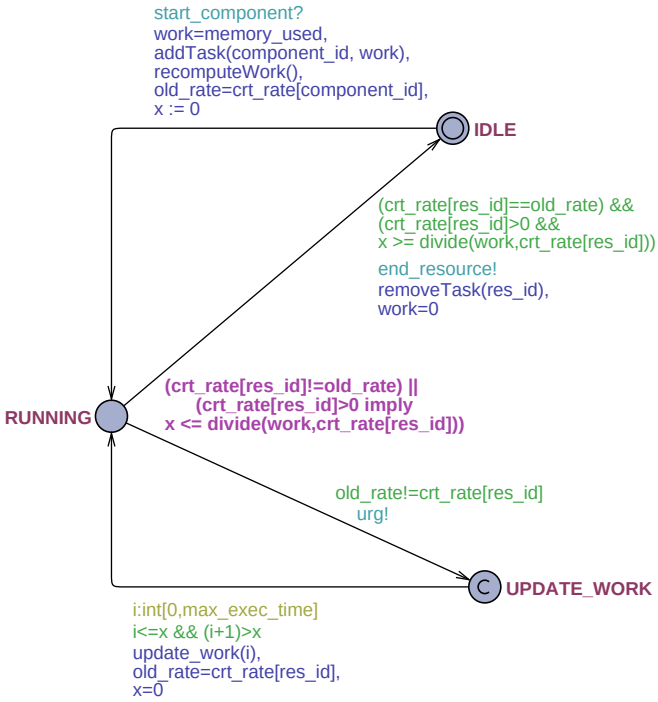


Fig. 4: Resource Automaton

Each job type is modeled as a separate automaton. Figure 5 displays an automaton representing a simplified version of the Print from DocBox job. We see there actions specific to both the datapath and memory management. In addition to these, we also observe the implementation of the upload in order scheduling rule. For simplicity, the variables regarding job non-overtaking are not specified.

4 Verification

We considered the following concurrent datapaths:

Scan To Email: Scanner $\rightsquigarrow$ IP1 $\rightsquigarrow$ IP2 $\rightsquigarrow$IP4 $\rightsquigarrow$ USB Upload

Print From Docbox: USB Download $\rightsquigarrow$ IP4 $\rightsquigarrow$ Print $\rightsquigarrow$ USB Upload, with dependencies as in Figures 2a and 2b. Our goal was to find the worst case latency for the files coming from Scanner when they had uncertain arrival time and the print jobs formed an infinite stream[3].

The scheduling rules implemented in our model simplified the analysis. However, the model was nondeterministic. One cause was the uncertain arrival times of scan jobs. The other cause was the lack of partial order reduction techniques

[3] All the experiments were performed using Uppaal version 4.1.2 on a Sun Fire X4440 server with 16 cores (AMD Opteron 8356, 2.3GHz), with 128 GB of DDR2 RAM.

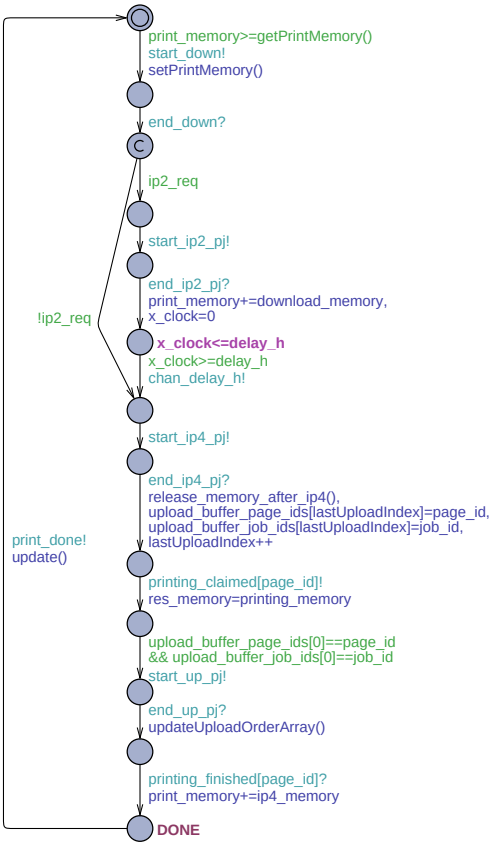


Fig. 5: Print from Doc Box Automaton

implemented in Uppaal: when multiple independent actions occurred simultaneously, Uppaal analyzed all the alternatives, and this led to state space explosion. Therefore, we searched for modalities to simplify the latter type of nondeterminism. The solution adopted was to specify priorities among all the channels declared in the system. However, when scan and print jobs accessed shared resources, they used the same channels. Due to this, we declared separate channels and gave higher priority to the channels employed for the communication between print jobs and resources.

The property verified was

```
A[]  ((forall (i:int[0,max_scan_jobs-1])
!A1S(i).INIT imply A1S(i).latency_clock<=worst_latency)),
```

where worst_latency was found manually.

Figure 6 shows the monotonic increase of the worst latency with the increase of the number of concurrent scan jobs. In these experiments, no print job was allowed in. As we can see, the increase stops when the machine reaches the maximum scan job capacity that it can process, 19 in this case. The last point

Fig. 6: Worst scan job latency without print jobs in the system

in the figure shows that, if the system is fully loaded with scan jobs, a user has to wait more than two times for his outcome comparing to the case when there are no other concurrent jobs. In these experiments, the Uppaal running time is insignificant and bus throttling is not needed.

The analysis results of the worst scan job latency in the presence of print jobs is listed in Table 1. All these experiments contained 19 scan jobs with uncertain arrival times and a number of infinite concurrent print jobs indicated in the first column. Table 1 also shows the Uppaal analysis details. Unfortunately, due to long Uppaal running time, we could not observe the worst scan job latency in combinations with high number of infinite print jobs. The current model configuration allows far more than 20 print jobs present in the system. However, the important observation which we can make is that the influence of the print jobs upon the worst case latency of scan jobs is not negligible.

Infinite Print Job No	Peak Mem Usage(KB)	Running Time(s)	States Explored	Worst Latency(ms)
0	121784	4376.03	723	5341
5	473952	6459.28	349951	6711
10	1107012	15274.40	996693	9843
15	1561524	8934.50	897307	12411
20	-	no result in 24 hours	-	-

Table 1: Worst scan job latency with print jobs in the system

During the analysis, we observed that the bus throttling rule had two negative effects. On one hand, as Table 1 also shows, this bus arbitration rule worsened

the scan job latency by slowing the execution time of the resource required within their datapath. On the other hand, it increased the analysis time due to many changes in the speed of some resources which the analysis had to explore.

Infinite Print Job No	Peak Mem Usage(KB)	Running Time(s)	States Explored	Worst Latency(ms)
5	195700	2082.01	147806	4941
10	315540	2232.54	294789	6426
12	384896	2315.72	464711	6767
13	434572	2478.34	555770	6940
14				6940

Table 2: Worst scan job latency of the dimensioned model

Infinite Print Job No	Peak Mem Usage(KB)	Running Time(s)	States Explored	Worst Latency(ms)
5	205684	1940.63	153259	4701
10	304464	2074.39	271351	5933
12	339864	2240.14	331047	6274
13	539340	2539.96	921847	6445

Table 3: Worst scan job latency with the improved bus throttling rule

Further, we searched for improvements in the bus throttling rule but firstly we dimensioned the model such that we were able to analyze it fully with Uppaal. In this sense, the scan memory was reduced by a factor of 2, whereas the print memory was reduced by a factor of 4. Except for these, nothing else was changed (the scan jobs had an uncertain arrival time and the print jobs formed an infinite stream). As a result, fewer jobs occupied the machine at some point in time. The analysis of this configuration is detailed in Table 2. This model allowed maximum 5 concurrent scan jobs and 13 concurrent print jobs.

When we searched for changes of the bus arbitration rule, we had to take into account that only the last four resources in the priority list (see Figure 3) could be interrupted while processing a job. The optimization we found, was to switch the order between USB upload and USB download. With this simple change, we obtained the worst latencies showed in Table 3. The improvement was between 4.8% and 7.6%.

To conclude, we analyzed a timed automata model built for an Océ printer architecture. The model contained many design details. We searched for the worst latency of one of the concurrent applications. During the analysis the Uppaal running time was long, on the order of days. Further, for a better understanding of the system, we dimensioned the model such that we saw the peak value

of the worst latency and searched for optimization of one important scheduling rule. Currently, we discuss this improvement with the Océ engineers if it can be implemented in the controller of a printer with similar characteristics.

5 Conclusions

We have analyzed an Océ printing machine and two of its datapaths. We computed the worst latency of one datapath which has uncertain arrival time and the other datapath is infinitely often used. Our results show a strong dependency between the two datapaths.

As usual with model checking, long running time was a key issue within our case study. In order to be able to do the model checking (within reasonable time), we had to slightly scale down some of the parameters in the model. Still, the current version of Uppaal is close to the point where it can handle the complexity of industrial designs. One technical issue that we faced is that although essentially the behavior of the model is fully deterministic when all the scheduling rules are added, the resulting Uppaal model is not (and suffers from state space explosion) due to interleaving of internal actions of the various resources. We resolved this by using the channel and process priorities from Uppaal, but a better solution would be to extend Uppaal with support for confluence detection and/or partial order reduction.

We computed the worst latency by repeatedly checking an invariant property. Using a binary search we managed to find the exact value of certain parameters. However, this type of parametric analysis requires a lot of time and it would be most helpful to mechanize it using Uppaal, possibly using multiple processors to parallelize computations.

A lesson that we have learned is that it is extremely difficult to maintain correctness of the model in a setting where the object of modeling has such a high complexity. There was not a single document describing the design. In fact there was not a single person who was able to answer all our questions: the knowledge was spread over a large design team. For the engineers it is difficult to understand the intricacies of our Uppaal model. The syntax of Uppaal is not sufficiently expressive to describe the design in such a way that a small change in the design corresponds to a small change in the model. Due to these difficulties, the Octopus project has decided to develop a high level language for describing the designs, together with a translation to Uppaal: on one hand this will make it much easier to communicate with the engineers, and on the other hand it will reduce the chances of introducing errors in the Uppaal model.

References

1. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4) (2009) 65–76
2. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Third International Conference on the Quantitative

Evaluation of SysTems (QEST 2006), 11-14 September 2006, Riverside, CA, USA, IEEE Computer Society (2006) 125–126
3. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer **1**(1–2) (1997) 134–152
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Formal Methods for the Design of Real-Time Systems. Volume 3185 of Lecture Notes in Computer Science. Springer-Verlag (2004) 200–236
5. project, O.: Homepage `www.esi.nl/short/octopus` (2010)
6. Igna, G., Kannan, V., Yang, Y., Basten, T., Geilen, M., Vaandrager, F., Voorhoeve, M., Smet, S., Somers, L.: Formal modeling and scheduling of datapaths of digital document printers. In: FORMATS 08, Berlin, Heidelberg, Springer-Verlag (2008) 170–187
7. Jensen, K., Michael, L., Wells, K.L., Jensen, K., Kristensen, L.M.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. In: International Journal on Software Tools for Technology Transfer. (2007) 2007
8. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. EATCS Series. Springer Verlag (1992)
9. Ghamarian, A.H., Geilen, M.C.W., Stuijk, S., Basten, T., Moonen, A.J.M., Bekooij, M.J.G., Theelen, B.D., Mousavi, M.R.: Throughput analysis of synchronous data flow graphs. In: In ACSDâĂŹ06, Proc. (2006), IEEE, IEEE (2006) 25–34
10. Stuijk, E., Geilen, M., Basten, T.: Sdf 3 : Sdf for free. In: Application of Concurrency to System Design, ACSD 06, Proceedings. IEEE. (2006) 276–278
11. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In Chakraborty, S., Halbwachs, N., eds.: Proceedings of the 9th ACM & IEEE International conference on Embedded software, EMSOFT 2009, Grenoble, France, October 12-16, 2009, ACM (2009) 107–116
12. Chakraborty, S., Kunzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: DATE '03: Proceedings of the conference on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2003) 10190
13. Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping applications to tiled multiprocessor embedded systems. In: Proceedings of the Seventh International Conference on Application of Concurrency to System Design, Washington, DC, USA, IEEE Computer Society (2007) 29–40
14. Hamann, A., Jersak, M., Richter, K., Ernst, R.: Design space exploration and system optimization with symta/s– symbolic timing analysis for systems. In: Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004), 5-8 December 2004, Lisbon, Portugal, IEEE Computer Society (2004) 469–478
15. Harbour, M.G., García, J.G., Gutiérrez, J.P., Moyano, J.D.: Mast: Modeling and analysis suite for real time applications. In: 13th Euromicro Conference on Real-Time Systems (ECRTS 2001), 13-15 June 2001, Delft, The Netherlands, Proceedings, IEEE Computer Society (2001) 125–134