TIME PREDICTABLE CPU AND DMA SHARED MEMORY ACCESS

* *Christof Pitter*

Institute of Computer Engineering
Vienna University of Technology, Austria
cpitter@mail.tuwien.ac.at

Martin Schoeberl

Institute of Computer Engineering
Vienna University of Technology, Austria
mschoebe@mail.tuwien.ac.at

ABSTRACT

In this paper, we propose a first step towards a time predictable computer architecture for single-chip multiprocessing (CMP). CMP is the actual trend in server and desktop systems. CMP is even considered for embedded real-time systems, where worst-case execution time (WCET) estimates are of primary importance. We attack the problem of WCET analysis for several processing units accessing a shared resource (the main memory) by support from the hardware. In this paper, we combine a time predictable Java processor and a direct memory access (DMA) unit with a regular access pattern (VGA controller). We analyze and evaluate different arbitration schemes with respect to schedulability analysis and WCET analysis. We also implement the various combinations in an FPGA. An FPGA is the ideal platform to verify the different concepts and evaluate the results by running applications with industrial background in real hardware.

1. INTRODUCTION

This paper presents a hard real-time system consisting of a hard real-time application running on a time predictable Java Optimized Processor (JOP) [1] and an additionally direct memory access (DMA) unit with a regular access pattern. This unit is represented by a video graphics array (VGA) controller. Both the CPU and the DMA unit share the main memory of the system. Meeting the deadlines of the tasks of the real-time application is of utmost importance. Therefore, the task set of the system requires a timing validation by schedulability analysis.

A real-time computer system has to produce logical correct results within a specified period of time. If a correct result of a computation is late, the result is considered useless. Such real-time systems (RTS) or safety-critical systems have to handle concurrent tasks, such as communication, calculating values for a control loop, user interface and supervision in embedded systems. A natural way to handle these concurrent jobs is to split them up into individual tasks. Every hard real-time system contains at least one so-called hard real-time task. A task is classified a hard real-time task when a missed deadline may cause a critical failure of the system. Non safety-critical tasks in such a system are soft real-time tasks. If a deadline is missed, the system will still produce correct results [2] but with degraded service.

Safety-critical systems must be predictable in the time domain. It is of utmost importance to be able to analyze the maximal time or WCET of the task. If and only if these upper bounds can be calculated, the schedulability analysis can be performed. They are necessary to test whether a task set can be scheduled on the target system or not [3].

There exist two possibilities for modeling and implementing the RTS consisting of the application running on the CPU and the DMA controller sharing the main memory:

- DMA as soft real-time task
- DMA as hard real-time task

In the first approach, the task of the DMA controller is represented as a soft real-time task. As a consequence, some interference (e.g. flickering on the VGA display) may occur when the deadline of the DMA is violated. Hence, the DMA task performs in a best-effort manner and can be excluded from schedulability tests. Increasing the buffer size in the DMA controller can help to support a smooth communication between the shared memory and the DMA controller. Nevertheless, this kind of a system is of minor importance because hard RTSs are the target of this paper.

In the second attempt the DMA task as well as the hard real-time application running on the CPU has to meet its deadline. As a consequence, the system contains the application hard real-time tasks and the memory-streaming hard real-time task that are performed simultaneously. Although a VGA display is usually not accounted as a hard real-time task, it serves as a good example demanding a constant amount of data within a known period of time. In the future, the VGA will be replaced by another CPU.

The rest of the paper considers the second approach (hard RTS) and is structured as follows. Section 2 presents the related work. In Section 3, we explain the two basic options of analyzing the behavior of the RTS and describe how schedulability tests are carried out. Section 4 describes the JopVga system and the worst-case analysis results of several experiments. At the end, it discusses the acquired outcome of the paper. Finally, Section 5 concludes the paper and gives guidelines for future work.

*The TPCM-project received support from the Austrian FIT-IT SoC initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 813039.

2. RELATED WORK

In [4] Atanassov and Puschner describe the impact of dynamic RAM refresh on the execution time of real-time tasks. The use of DRAM memory in RTSs involves a major drawback because these memory cells have to be periodically refreshed. During the refresh, no memory request can be processed. As a consequence, the request is delayed and the execution time of the task increases.

Many researchers in the real-time community have turned their attention to timing-analysis tools described by Wilhelm et al. [3]. The problem is that the available results for uniprocessors are not applicable to modern processor architectures because the WCET analysis is hardware dependent [3, 5]. Multiprocessor systems consisting of shared memories and busses are hard to predict. In addition, the WCET of each individual task depends on the global system schedule.

Even though so much research has been done on multiprocessors, the timing analysis of the systems has been neglected. An example represents the scalable, homogeneous multiprocessor system by Gaisler Research AB. It consists of a centralized shared memory and up to four LEON processor cores that are based on the SPARC V8 architecture [6]. This embedded system is made available as a synthesizable VHDL model and therefore is well suited for SoC designs. Leon is introduced for European space projects as well as for military and demanding consumer applications. Nevertheless, no literature concerning WCET analysis regarding the multiprocessor has been found.

Another example depicts the ARM11 MPCore [7]. It introduces a pre-integrated symmetric multiprocessor consisting of up to four ARM11 microarchitecture processors. The 8-stage pipeline architecture, independent data and instruction caches and a memory management unit for the shared memory make a timing analysis difficult.

We believe that the impact of WCET of real-time tasks sharing a main memory has not received enough coverage in literature yet. This paper is a first step towards a time-predictable multiprocessor. Providing WCET guarantees and reliable schedules for a multiprocessor system becomes a great challenge.

3. CPU/DMA SHARED MEMORY ACCESS

In Section 1, we presented two possibilities of integrating the DMA task into the system. Either the DMA controller represents a soft real-time task or it is considered a hard real-time task. This paper addresses the second solution.

3.1. Implementation of the DMA hardware controller

The DMA controller accesses the shared memory to read or write data autonomously of the CPU. The volume of the data transfer depends on the I/O-device. Therefore, the application of the device defines the quantity of memory requests within a fixed period of time. Two different options

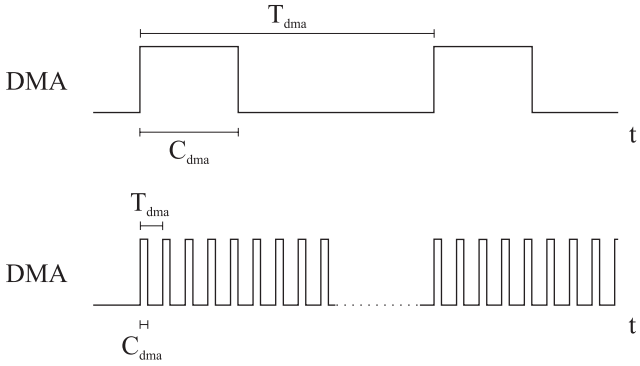

Fig. 1. The blocked and the spread memory access scheme of the DMA task.

are evaluated to implement the memory access scheme of the DMA controller (see Figure 1):

1. The DMA controller accesses the shared memory in a blocked scheme. This approach is used to copy fast large blocks of the main memory to another device. Assume this task has the smallest period of all tasks within the system. Hence using fixed-priority scheduling [8] it has the highest priority. After the DMA task is completed, the other tasks of the CPU get permission to access the shared memory depending on their priority. This approach is a good representation of a multimedia task, such as streaming data, performed via DMA.

2. The DMA controller accesses the memory in a timely spread scheme denotes the other extreme. It generates a smaller period because all memory requests are timely spread on the original period. The highest priority of this task is required. If this task does not hold the highest priority, the system will not function correctly because the DMA task will starve when the CPU's software tasks make extensive memory requests. Consequently, it would perform like the system with a soft real-time DMA task described in Section 1.

3.2. Task vs. WCET based Analysis

There exist two possibilities to analyze the timing behavior of the RTS:

- DMA access represents an additional real-time task
- DMA access is included in the WCET analysis of each individual application task

The first method considers the DMA task in the schedulability analysis. Hence, all the tasks of the application running on the CPU and the DMA task have to be considered. This simple task set consists of independent periodic tasks with fixed priority. The DMA controller must have the highest priority. The resulting task set can be used for schedulability tests to analyze the timing behavior of the application running on the CPU.

The second approach models the RTS in a different way. The DMA controller and the CPU are accessing shared memory. We are interested in the WCET of the application in spite of the memory communication of the DMA. Therefore, the blocking delay, caused by each possible read or write access of the DMA controller, has to be added to the WCET estimations of the real-time tasks running on the CPU. The results serve bounded WCET estimates of each individual task. As a consequence, the WCET values for the tasks increase, but the DMA task can be omitted from the schedulability analysis.

3.3. Schedulability Analysis

The major goal of this paper is the analysis of the timing behavior of the RTS depending on the different options of the memory access of the DMA controller.

Assume that the CPU of the system runs several real-time tasks that are accessing the shared memory. Additionally the DMA controller requests data of the main memory with a regular access pattern. Therefore, the system consists of the DMA task and the tasks of the real-time application running on the CPU. Using fixed-priority scheduling [8], the priorities of the tasks are ordered rate monotonic. The smaller the period the higher is the priority of the task. In order to ensure that all tasks can be completed within their deadlines schedulability tests are carried out.

Utilization-based schedulability test

In [8] it has been shown that a simple schedulability test can be carried out by taking the utilization of the several tasks into account. The utilization is the result of dividing the computation time by the period of the task. If Equation 1 holds then all tasks will meet their deadlines. Otherwise, the task set may or may not fail at run-time. C_i denotes the computation time of the task τ_i, T_i is the period of task τ_i and N stands for the number of the tasks to schedule.

$$\sum_{i=1}^{N}(C_i/T_i) \leq N(2^{1/N}-1) \tag{1}$$

If the task set fails, the utilization-based schedulability test cannot guarantee that all tasks will meet their deadlines. Nevertheless, the task set may not fail at run-time.

Response time analysis

A more exact schedulability test by Joseph and Pandya is presented in [9]. The result of this response time analysis for a set of independent tasks provides a necessary and sufficient condition. If the result is positive the task set will be schedulable at run-time. The task set will not be schedulable if the test fails. The worst-case response time R_i of each individual task is calculated and then compared with the task's deadline or period respectively. The equation for the response time is:

$$R_i = C_i + \sum_{j \in hp(i)} \lceil R_i/T_j \rceil \cdot C_j \tag{2}$$

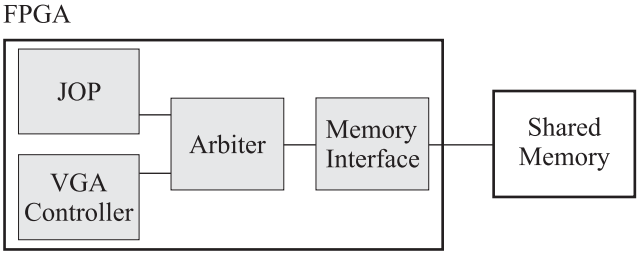

Fig. 2. JopVga system.

The expression $hp(i)$ denotes all tasks with a higher priority than the task τ_i. The smallest R_i that solves Equation 2 is the worst-case response time of τ_i. A recurrence relationship can be formed that allows the calculation of the response time [10]:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \lceil w_i^n/T_j \rceil \cdot C_j \tag{3}$$

The solution is found when $w_i^{n+1} = w_i^n$. Then w_i^n represents R_i. If one task i has a larger response time than its deadline (or period T_i) the task set cannot be scheduled.

4. EVALUATION

This section provides an overview of the JopVga system and the corresponding sample application. Furthermore the timing behavior of the RTS is analyzed. The results of the experiments and calculations are compared and classified.

4.1. JopVga System

The JopVga system is the hardware used for our experiments. It consists of a time-predictable processor called JOP [1], a VGA controller, an arbiter, a memory interface and an SRAM memory. JOP, the VGA controller, the memory arbiter and the memory interface are implemented on an Altera Cyclone FPGA. As illustrated in Figure 2 the external memory is connected to the memory interface. This 1 MByte 32 Bit external SRAM device represents the shared memory of the JopVga system. A SoC bus, called SimpCon [11], connects JOP and the VGA controller with the arbiter. The arbiter is connected via SimpCon to the memory interface.

The arbiter is responsible for setting up the communication between the shared memory and the VGA controller and JOP respectively. It schedules the memory communication of both masters. The shared main memory of the system is divided into two segments: 640 KByte are dedicated to JOP and the remaining 384 KByte are associated with the VGA. Both JOP and the VGA controller run at a clock frequency of 80 MHz, resulting in a period of 12.5 ns per cycle. The SRAM-based shared memory has an access time of 15 ns per 32 bit word. Hence every memory access needs at least 2 cycles. At the moment the arbiter as well as the VGA controller is not capable of using the pipelining approach of the memory access that

Table 1. Task set.

τ	T (μs)	C (μs)	Priority
τ_{vga}	17	4.8	3
τ_{lift}	500	162.7	2
τ_{kfl}	3000	782.2	1

is introduced by the SimpCon interface [11]. Each memory request of JOP takes 4 cycles and every VGA request takes 3 cycles. The bandwidth of the memory (BW_{mem}) calculates to:

$$BW_{mem} = 4Byte/2.5 \cdot 10^{-8}s = 160MByte/s \quad (4)$$

The VGA controller uses a VGA resolution of $1024 \cdot 768$ pixels with each pixel consisting of 4 bits. As a consequence the memory used for the VGA display results in 384 KByte ($1024 \cdot 768 \cdot 0.5$ Byte). The horizontal frequency is 60 kHz, which results in a horizontal period of about 17 μs per line and 17 ns per pixel. The vertical frequency is 75 Hz. Therefore the bandwidth used by the VGA calculates to:

$$BW_{vga} = (128 \cdot 4Byte)/1.7 \cdot 10^{-8}s = 30.12MByte/s \quad (5)$$

Dividing BW_{vga} by BW_{mem} results in 18.83% of the memory bandwidth for the VGA controller. JOP has a bandwidth of 80 MByte/s which results in 50% of the memory bandwidth available.

Application

In order to estimate the worst-case execution time of a RTS all real-time tasks have to be taken into account. In the following the task set illustrated in Table 1 represents the system under test. It consists of the VGA task and two tasks running on JOP. The VGA task can be modeled as a periodic task with the highest priority and a fixed runtime. The computation time C_{vga} is calculated by multiplying 128 memory accesses times 3 cycles. Using a clock frequency of 80 MHz results in 4.8 μs. T_{vga} is predetermined by the horizontal period of 17 μs.

Two real-world examples with industrial background represent the two tasks running on JOP. `Lift` is a lift controller used in an automation factory. `Kfl` is one node of a distributed RTS to tilt the line over a train for easier loading and unloading of goods wagon. Both applications consist of a main loop that is executed periodically. In our experiments we use both applications to represent two independent real-time tasks. The WCET of these two tasks are inferred from the WCET analysis tool [5]. In the second column of Table 2 the WCET estimates of the two tasks are given in clock cycles. Multiplying those estimates with the clock period result in τ_{lift} = 162.7 μs and τ_{kfl} = 782.2 μs.

The periods and the computation times of the tasks τ_{vga}, τ_{lift} and τ_{kfl} define our system under test. The priority of 3 depicts the highest prior task. This simple task set illustrates a RTS that is further analyzed using schedulability tests.

4.2. Analysis using the Task Approach

In this section the VGA controller represents another real-time task of the system that is taken into account in the analysis of the timing behavior. Both the blocked memory access scheme and the spread memory access scheme, as described in Section 3.1, are evaluated.

Blocked

Using the values of Table 1 the utilization of each individual task is calculated dividing the computation time C_i by the corresponding period T_i resulting in U_{vga} = 28.24%, U_{lift} = 32.54% and U_{kfl} = 26.07%. Applying the utilizations to Equation 1 results in an overall utilization of 86.85%. The overall utilization may not be more than $3(2^{1/3} - 1) = 78.00\%$ because three tasks are involved. The condition does not hold and consequently this task set fails the utilization-based schedulability test. It cannot be guaranteed that all tasks meet their deadlines. Therefore, a response time analysis is carried out next.

The response time R_{vga} is the same as the computation time because this task has the highest priority.

$$R_{vga} = C_{vga} = 4.8\mu s \quad (6)$$

The response time of the next lower prior task τ_{lift}, denoted as R_{lift}, is the addition of the computation time C_{lift} and the time of interference of all higher prior tasks (in that case the interference of τ_{vga}).

$$w_{lift}^{n+1} = C_{lift} + \lceil w_{lift}^{n}/T_{vga} \rceil \cdot C_{vga} \quad (7)$$

The response time is calculated using the values from Table 1. The response time of R_{lift} is found when w_{lift}^{n+1} equals to w_{lift}^{n}. It is 229.9 μs which is less than T_{lift}. Finally, R_{kfl} has to be calculated. R_{kfl} is the addition of the computation time C_{kfl} and the time of interference of the two higher priority tasks τ_{vga} and τ_{lift}. The result of R_{kfl} is 1999.4 μs. All the response times are smaller than their appropriate periods and hence the response time analysis has a positive outcome. This response time calculation ensures that the tasks will meet their deadlines because the successful analysis is sufficient and necessary [2] even though the utilization-based schedulability test could not be passed.

Spread

The VGA task accesses the memory in a timely spread scheme. The new values for the period and the computation time of τ_{vga} are calculated by dividing the original period T_{vga} = 17μs of Table 1 by the cycle time of 12.5ns. It results in 1360 cycles. We need 128 memory requests for each line on the VGA. Hence T_{vga} = 10 cycles and C_{vga} = 3 cycles. The remaining 80 cycles are not used. Hence the values for the VGA task change to T_{vga} = 125 ns and C_{vga} = 37.5 ns.

Table 2. WCET estimates given in clock cycles.

App	JOP only	JOP with VGA	Increase
Kfl	62573	83131	33%
Lift	13016	16118	24%

Table 3. Task set for the WCET method.

τ	T (μs)	C (μs)	Priority
τ_{lift}	500	201.5	2
τ_{kfl}	3000	1039.1	1

Using these values for JOP's tasks and the values for τ_{lift} and τ_{kfl} the utilization-based schedulability test logically results in a similar overall utilization of 88.61% as described in the previous section. A small divergence between the results can be explained by the remaining 80 cycles that are not used in this memory access scheme. Again, the utilization test is negative.

Therefore, the response time analysis is used. R_{vga} is similar to C_{vga} = 37.5 ns. R_{lift} calculates to 232.5 μs and R_{kfl} = 2279.6 μs. The positive result of the response time analysis shows that the task set can be scheduled. As in the utilization-based test, the remaining 80 cycles affect the results of R_{lift} and R_{kfl}. Both response times are larger than in the blocked memory access scheme. As a consequence, the spread memory access scheme is worse than the blocked scheme.

4.3. Analysis using the WCET method

The second approach to include the DMA unit in the schedulability analysis is to model the DMA access in the WCET values for memory access of the software tasks. Each instruction that accesses memory has to include the maximum delay due to a possible memory access by the DMA unit.

Blocked

Using the blocked memory access scheme and the WCET method for analysis is not a reasonable approach. One would have to account the delay of the whole block of memory requests of the VGA to each memory access of JOP. That results in a very conservative WCET for each memory access of JOP. Hence, it is not further investigated.

Spread

The WCET analysis tool [5] can be parameterized with respect to the memory access time (the wait states for memory read, memory write, and the cache load). The memory access from the VGA unit takes 2 cycles plus 1 cycle in the arbiter to switch between the two masters. Therefore, we add 3 cycles to the wait states.

Table 2 shows the WCET values in clock cycles for different applications for the stand-alone processor and when adding the DMA device. We see an increase of 24% to 33% of the WCET for the tasks.

Table 4. Comparison of the response times of the task approach with blocked DMA (C_1 and R_1) and the WCET method with spread DMA access (C_2 and R_2).

τ	T (μs)	C_1 (μs)	R_1 (μs)	C_2 (μs)	R_2 (μs)
τ_{vga}	17	4.8	4.8	–	–
τ_{lift}	500	162.7	229.9	201.5	201.5
τ_{kfl}	3000	782.2	1999.4	1039.1	1845.1

These values are conservative as each memory access is modeled with the maximum blocking time. For a single memory access (such as bytecode `getfield`) this is the best we can do without further analysis of the instruction pattern. However, for cache loading we could include the access pattern (in our example one access per 10 clock cycles) into the analysis of the cache load time. Previously experiments showed that the load time for the method cache produces most of the memory requests. Consequently, with inclusion of the access pattern into the WCET analyzer tool, we can provide tighter WCET values. This is a new approach to WCET analysis, as in all current approaches the WCET analysis is independent from the schedulability analysis. Schedulability analysis is usually the next step and assumes known WCET values.

The computation time values of Table 3 are calculated by multiplying the WCET estimates of τ_{lift} and τ_{kfl} from Table 2 with the clock period of 12.5 ns. The values for those two tasks are the basis for the schedulability test. The utilization-based test results in 74.94%. Only two tasks are taken into account and hence this result is less than $2(2^{1/2}-1) = 82.84\%$. Even though this test is positive, we also investigate the response time analysis.

The response time analysis for the WCET estimates of the system with the VGA results in tighter response times than in the analysis using the task approach of Section 4.2. R_{lift} is the same as C_{lift} = 201.5 μs and R_{kfl} calculates to 1845.1 μs.

4.4. Discussion

Table 4 shows the results of the evaluation for both analyses: column 3 and 4 (C_1 and R_1) for the DMA task approach and column 5 and 6 (C_2 and R_2) for the DMA-WCET approach. We can see that both τ_{lift} and τ_{kfl} have a higher WCET (C_2) in the DMA-WCET approach. As we do not have to include the VGA task in the response time analysis, the response time R_2 is less for both tasks despite the fact that C_2 is higher.

The result shows that the inclusion of the DMA access into the WCET analysis provides tighter worst-case response times than considering the DMA as an additional task. The difference can be explained as follows: most instructions on JOP do not access the main memory; they use the internal stack cache for data. The pipeline is filled from the instruction cache most of the time. In the task approach, all instructions are *blocked* by the VGA task.

Table 5. Comparison of the system performances in iterations/s.

App	JOP only	blocked VGA	spread VGA
Kfl	12163	11562	11628
Lift	9643	9194	9356

The WCET analysis is more exact as it delays only those instructions, which do actually access the main memory and the cache load events.

To validate our calculations and measurements we run all mentioned task sets on real hardware and the JopVga system respectively. No deadline violation of any task could ever be observed when performing these experiments.

4.5. Benchmarks

Although the solution is aimed at RTSs, i.e. a time predictable system, the average case performance is still interesting. The system under test is the JopVga system. The FPGA platform enables us to compare the performance of the system with an enabled VGA controller versus one with a disabled VGA controller. The results are achieved by running real applications in real hardware. For our measurements, we use the embedded Java benchmark suite JavaBenchEmbedded as described in [12]. The result is iterations per second, which means a higher value illustrates a better performance. In Table 5, the benchmark results are shown.

The `Kfl` application is slowed down just by 4.4% due to the memory contention with the spread VGA memory access scheme. The WCET estimates of the same task resulted in an increase of up to 33% (see Table 2). Another benchmark called `Lift` experiences an even smaller slowdown of 3.0% due to the contention with the VGA task. The WCET estimates of the same task resulted in an increase of 24%.

To recapitulate, although we use about one third of the memory bandwidth for the DMA unit both applications suffer less than that one third in their execution time. This result is a promising indication that the memory system is not the bottleneck of the single CPU with the VGA. There is enough headroom for further devices. The result is promising for our further plans on a CMP version with several JOPs sharing a single memory.

5. CONCLUSION AND FUTURE WORK

In this paper, we have analyzed a system with a processor and a DMA unit with respect to WCET and schedulability. We have found two ways to model the influence of the DMA unit to the application tasks: 1.) the DMA access as an additional real-time task and 2.) include the DMA memory access in the WCET analysis of the individual application tasks. We found that the second approach results in a tighter estimation and enables more processing resources for the application.

We will investigate the possibility to include a known memory access pattern into the WCET analysis of the cache loading to find tighter WCET estimates. On a cache load, we can guarantee that only a maximum number of memory loads can conflict with the DMA unit. The next step is the application of our findings to a system with several CPUs – the CMP JOP system. In that case, the memory access pattern is less predictable. Therefore, the shared resource can only be modeled by the WCET approach.

6. REFERENCES

[1] M. Schoeberl, *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. PhD thesis, Vienna University of Technology, 2005.

[2] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. International computer science series, pub-AW:adr: Addison-Wesley, third ed., 2001. Revised edition of *Real-time systems and their programming languages*, 1990.

[3] P. Puschner and A. Burns, "A review of worst-case execution-time analysis," *Journal of Real-Time Systems*, vol. 18, pp. 115–128, May 2000.

[4] P. Atanassov and P. Puschner, "Impact of dram refresh on the execution time of real-time tasks," in *Proc. IEEE International Workshop on Application of Reliable Computing and Communication*, pp. 29–34, Dec. 2001.

[5] M. Schoeberl and R. Pedersen, "Wcet analysis for a java processor," in *JTRES '06: Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems*, (New York, NY, USA), pp. 202–211, ACM Press, 2006.

[6] S. I. Inc., *The SPARC Architecture Manual: Version 8*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1992.

[7] ARM, "Arm11 mpcore processor, technical reference manual." http://www.arm.com, August 2006.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J*, vol. 29, no. 5, pp. 390–395, 1986.

[10] N. C. Audsley, A. Burns, R. I. Davis, K. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Systems*, vol. 8, no. 2-3, pp. 173–198, 1995.

[11] M. Schoeberl, "SimpCon - a simple and efficient SoC interconnect." Available at: http://www.opencores.org/, 2007.

[12] M. Schoeberl, "Evaluation of a Java processor," in *Tagungsband Austrochip 2005*, (Vienna, Austria), pp. 127–134, October 2005.