

Performance Analysis of a RTOS by Emulation of an Embedded System

Karlheinz Weiß, Thorsten Steckstor
Lehrstuhl für Technische Informatik
University of Tübingen
Am Sand 13, 72076 Tübingen, GERMANY
weiss@fzi.de, stecki@fzi.de

Prof. Dr. Wolfgang Rosenstiel
Lehrstuhl für Technische Informatik
University of Tübingen
Am Sand 13, 72076 Tübingen, GERMANY
rosen@informatik.uni-tuebingen.de

Abstract

This paper analyzes the performance of two different real-time operating systems. Therefore, we used a real benchmark embedded system design with fast external reaction times of about 220 μ s. We show that for such fast reactive systems, the software overhead of a real-time operating system becomes a limiting factor. We analyze the influence of novel microcontroller features, e.g., different on-chip caches, which tend to accelerate execution, but make it less predictable. These investigations have been conducted using our own emulation environment called SPYDER-CORE-P1.

1 Introduction

Most of today's existing technical applications are controlled by so-called embedded systems¹. Many different application areas exist which demand their own specific embedded system architecture. Therefore, as described in [1], a common definition of an embedded system does not find wide acceptance.

The benchmark embedded system presented in this paper is used in the area of industrial automation. In this domain, an embedded system architecture consists of an application specific hardware part, which interacts with the environment. At the same time, an application specific software part runs on a microcontroller. Especially the interaction with the system environment demands hard real-time requirements.

In the last few years, the rapid progress in microelectronic technology has reduced component costs while simultaneously speeding up the introduction of the 32 bit embedded microcontroller [2] in a widespread area of embedded system design. Additionally, powerful on-chip features, like data and instruction caches, programmable bus

interfaces and higher clock frequencies, speed up performance significantly and simplify system design. These hardware fundamentals allow Real-Time Operating Systems (RTOS) to be implemented, which leads to the rapid increase in total system performance and functional complexity.

Nevertheless, if fast reaction times of about 220 μ s must be guaranteed, the software overhead due to task switching becomes a limiting performance factor and the effect of the caches makes the performance analysis very difficult. These issues will be addressed in this paper, which is organized as follows:

In Chapter 2, we give a short overview of the state of the art and our own previous work. Chapter 3 introduces the emulation environment SPYDER-CORE-P1, which was developed for the emulation of sophisticated embedded systems. Chapter 4 describes the benchmark application and the different software tasks, which run under the control of the two investigated real-time operating systems *VxWorks* and *RTEMS*. Chapter 5 analyzes the performance effect of the different on-chip features provided by novel microcontrollers on overall performance and outlines the system bottlenecks. Chapter 6 describes the results of the investigation and Chapter 7 summarizes the entire paper.

2 Previous Work

2.1 State of the Art

Hard real-time requirements heavily influence the hardware/software partitioning in order to find a good solution with respect to performance and cost. The traditional method applied by most system designers today can be described as a two step solution. First, a printed circuit board containing all the selected chips is developed, and after production, the necessary application software is written in the second step. The main disadvantage of this method is the lack of detailed knowledge about the internal embedded system behavior, which can only be determined very late in the design process. If system requirements are not met, corrections must be made late in the development process, significantly increasing design time and total costs. Sometimes, the en-

¹This work was supported in part with funds of the Deutsche Forschungsgemeinschaft under reference number 3221040 within the priority program "Design and design methodology of embedded systems".

tire project must be cancelled due to time or budget restrictions.

The more scientific approach tries to describe the behavior of a system at a high level of abstraction. After the design entry step, some performance evaluation is done, followed by co-synthesis techniques for software and hardware as described in [5][6]. The result is verified using one or more combined simulators. This approach suffers from the lack of an appropriate design language to describe all the aspects of an embedded system. A mixture of different languages must therefore be used. Each language addresses a separate part of the system, which makes design entry a time consuming and difficult task. Additionally, the complex interaction between the system and its environment must be modelled for simulation. That work is also very difficult, and in some cases impossible without an unreasonable amount of resources, tools and design time.

If we compare the two design methods mentioned, we can say that using the traditional method leads to a system without no detailed knowledge of behavior during the design process and is mainly based on the experience of the designers to make it work. The scientific approach recognizes that fact, but suffers itself from a lack of appropriate tools to apply the method. That was the motivation for us to search for an applicable method for getting detailed information based on understandable measurements of the system early in the design stage. Furthermore, the method should be very close to the final target system and avoid estimations about the embedded system behavior to decrease the risk involved with the development.

An approach which is able to provide these features is embedded system emulation. It offers the possibility to find the best hw/sw partitioning and the best usage of system resources early in the design process. Many different emulation environments exist. A few environments [7][8] are principally very flexible, but because of the high degree of flexibility, are not suitable for an embedded system like the type of fast reactive embedded systems under consideration here. This is due to the great expansion of the resulting system, leading to the loss of real-time capability. Therefore, we have developed the new emulation environment explained in chapter 3.

2.2 Our Basic Work

The past two years were marked by the development of innovative embedded systems in the area of industrial automation and communication. This was done in conjunction with different companies, as these embedded systems are needed for industrial applications. This work serves as a basis for the current research work, which investigates more effective methods for embedded system design.

A major project was done in cooperation with different industrial companies and led to the development of an Actuator Sensor Interface (ASI), a so-called ASI-Master. ASI is a new system which allows the connection of up to 128 binary actuator and sensor devices to a suitable control unit via a single bifilar cable. An additional key feature of this work is the global access to the ASI-Master via the Internet, which leads to more valuable services as described in [4]. Currently that project uses the Real-Time Operating System (RTOS) *VxWorks*, which runs on SPYDER-CORE-P1 [3]. Additionally, we use *RTEMS* as a second RTOS to compare and validate the results of the analysis obtained by *VxWorks*. The RTOS is used for the scheduling of different tasks. This application, which serves as a benchmark, is explained in Chapter 4 and was selected due to three major characteristics:

1. Sophisticated software task architecture (RTOS)
2. Novel microcontroller architecture with caches
3. Fast reaction times to external events cause that task switching and interrupt reaction times become a major performance bottleneck

These characteristics result in a complex internal system behavior, which is suitable for demonstrating the benefits of the emulation method in order to give answers to four important questions which a system designer might ask:

1. What is the optimum clock speed in order to meet all real-time constraints and to prevent an oversized or an undersized system?
2. How much computation performance is consumed by the RTOS in a fast reactive system?
3. How great is the performance enhancement of the caches measured at the point of the optimum clock speed and what can be done with this cache enhancement?
4. What is the effect of different cache sizes on the important RTOS task-switching and interrupt reaction times?

This paper shows that these questions are not limited to that specific application, but are general questions about many applications in the mentioned area.

3 Emulation Platform SPYDER-CORE-P1

The current version of the Spyder emulation environment consists of two boards. SPYDER-ASIC-X1 [9] is one part of the tool set and mainly addresses the emulation of large VHDL-based ASIC designs on a FPGA. SPYDER-CORE-P1 is the second part of the tool set and it is used for the emulation of the entire embedded system architecture. Both boards are compatible and can easily be connected to each other via a backplane. For the work described here, only SPYDER-CORE-P1 was used, thus only this board is described in detail (see Figure 1). The SPYDER-CORE-P1 environment provides all the key components needed for embedded systems in the area of industrial automation

within a flexible, but still compact architecture, which guarantees high clock speeds. This enables a real-time emulation, which is very close to the final target system.

A 32 bit RISC embedded PowerPC 403GA/GCX microcontroller [2] is used. It provides the following four features:

1. It is available in a wide clock frequency range. This allows a wide spectrum of applications to be emulated. Analysis can achieve the right values to satisfy all real-time requirements and prevent an oversized or undersized system.
2. The microcontroller architecture provides different types of on-chip caches (instruction and data). Additionally, the different types of family members provides different cache sizes, e.g. the PPC403GA offers 1KB D-cache and 2KB I-cache, the PPC403GCX offers 8KB D-cache and 16KB I-cache. The analysis of the performance improvements with respect to different cache configurations in fast real-time systems, which run under control of a RTOS, is an interesting research topic.
3. A flexible programmable bus interface provides a direct interface to most peripheral devices. This enables the simple implementation of a low or high end system without additional glue logic.
4. The CPU-core provides sophisticated execution units with pipelines, super-scalar execution and branch prediction.

These PowerPC features are characteristic properties of novel microcontroller architectures, which dramatically speed up execution performance. On the other hand, these features make the embedded system behavior less predictable, especially by systems which have to meet hard real-time constraints. Therefore, worst case analysis has to be done using the emulation method.

The high-speed microcontroller bus is connected to the main memory and the interface FPGA. It can be used to connect the entire microcontroller bus to external devices via the extension header 2 or it can be used to emulate additional application specific interface hardware. Each microcontroller bus signal is available on the extension header 1. They provide the means to connect a co-processor, e.g., a reconfigurable XC6000 FPGA, to the microcontroller very closely or to connect debugging equipment, e.g., logic analyzer, to the emulation environment.

A driver device implements a buffer between the high-speed 32 bit wide microcontroller bus and the slower 8 bit I/O bus. It connects some frequently used communication and memory devices to the microcontroller. The 8MByte FLASH-bank is used to store the RTOS boot kernel and different FPGA-images. We extend the standard RTOS features with routines to reconfigure the FPGA images during

run-time. Therefore, SPYDER-CORE-P1 provides the global Run-Time Reconfiguration (RTR) capability.

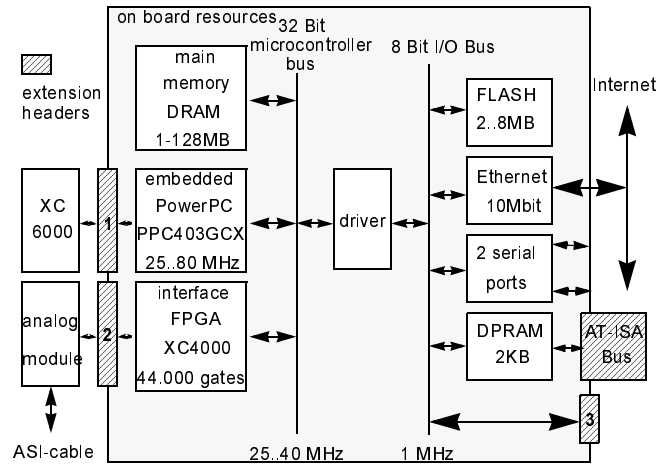


Figure 1: Architecture of SPYDER-CORE-P1

The 10MBit ethernet interface is mainly used by *Vx-Works* for connecting the target with the development environment called *Tornado*, which runs on a host PC. The serial ports can be used for standard terminal communication and the Dual Ported RAM (DPRAM) for debug support and download operations via the AT-ISA bus. This capability can be used during the customizing of the FPGA images and the program code. After finishing the development, SPYDER-CORE-P1 can be removed from the AT-ISA slot and is able to run standalone.

4 Internet Controlled ASI-Master

The Actuator Sensor Interface (ASI) connects up to 32 ASI slave chips via a single, bifilar cable with a ASI master unit (see Figure 2).

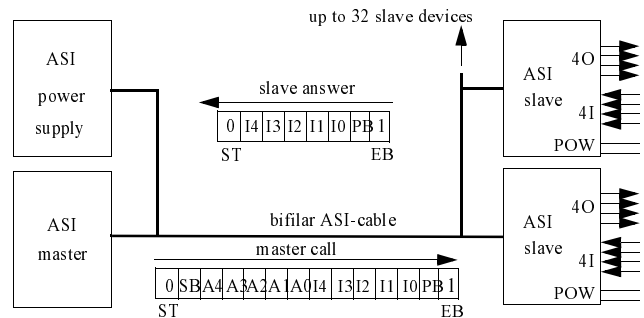


Figure 2: ASI Communication System

That master unit calls in a polling cycle including each connected ASI slave with its own address (bits A4.A0), followed by the output data image (bits I4.I0). The ASI slaves

respond if the address in the master call matches their own address, and they transfer the input data image back to the master. Each slave can be connected to up to four binary sensors (4I) and up to four (4O) actuators. The serial master call protocol must be modulated on the ASI cable using a dedicated analog module (see Figure 1). An ASI slave is available as a single chip solution which has the same capability on-chip. Additionally, an ASI power supply unit provides 30 DC voltage for the slaves via the same cable.

The ASI master is emulated on SPYDER-CORE-P1 very close to the final target system without any major changes. The hardware connection between the microcontroller and the analog module (two wires, serial in and out) is implemented as dedicated ASI hardware in the interface FPGA XC4000 (see Figure 3). The microcontroller writes the output data image to the register file and the ASI-UART performs a parallel to serial conversion with Manchester encoding. The digital serial data output is modulated to the ASI cable by the analog module. The receive path operates analogously to the send path.

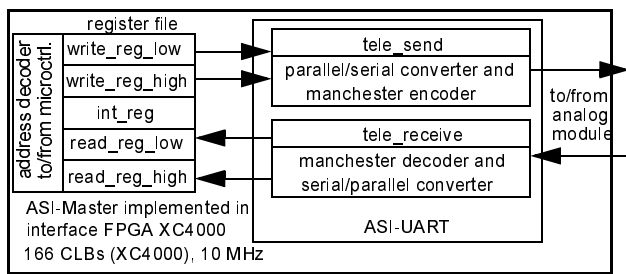


Figure 3: ASI Interface Hardware

The Real Time Operating System (RTOS) *VxWorks* [10] runs on the SPYDER-CORE-P1. It provides a TCP/IP stack for communication via ethernet. This interface is used as a link to the *VxWorks* development environment running on a host PC. Additionally, the ASI application uses that interface as a link to the Internet.

Four tasks run on the RTOS: two are hard real-time tasks, and the two other have no real-time constraints (see Figure 4).

1. The *int_service* task is a hard real-time constraint task and is responsible for the data exchange among the slaves. It generates the current process data image.
2. The *control* task is also a hard real-time constraint task and uses the current process data image to calculate the control commands.
3. The *C-server* task has no real-time requirements and is responsible for data and command exchange via the Internet.

4. The embedded *http-server* task also has no real time requirements and transfers JAVA applets to the calling client computer.

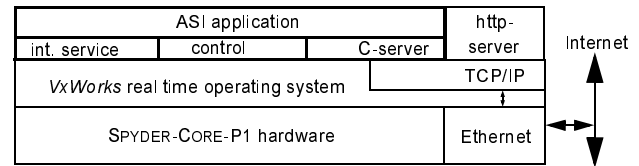


Figure 4: Software Architecture

For a more detailed description, refer to [4].

5 System Performance Analysis

Figure 5 shows a timing diagram of the hard real-time tasks, measured with a logic analyzer. The *int*-signal shows the interrupt communication between the ASI specific hardware implemented in the interface FPGA and the microcontroller. The *int-service* and *control*-tasks mark their beginning and end states by accessing a simple I/O-device (I/O-signal), which is recorded by the logic analyzer. The ASI standard defines the time delay between two serial bits on the ASI cable during the master call and slave answer with $6\mu s$. Together with master and slave break times, the microcontroller must exchange data with a slave every $220\mu s$. Therefore, this value is an ASI specific real-time critical constant. During this time, the microcontroller must execute the following sequence: interrupt reaction, *int-service*, task change, run *control* task and take a semaphore (*semTake*) as shown in Figure 5.

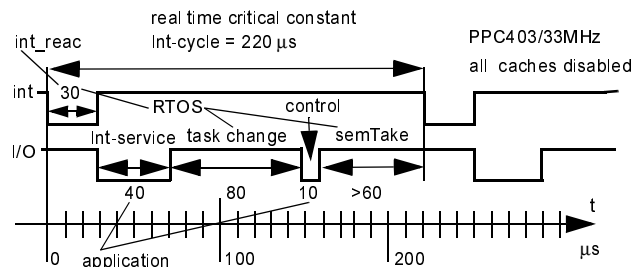


Figure 5: Execution Times

To determine the minimum clock speed, all caches must be disabled due to the fact, that this sequence runs under a hard real-time constraint and a worst case analysis has to be done. This sequence must be executed within a time range which is equal or less than the real-time critical constant of $220\mu s$ in order to guarantee the real-time requirement. Figure 5 shows that if the worst case scenario occurs, it will still run successfully with a minimum clock speed of 33MHz or more. If the clock speed decreases, the next interruption (*int*)

= low) happens before the semTake currently being executed has finished. This means, the real-time requirement can no longer be satisfied.

Figure 6 shows the clock frequency range from 25MHz up to 80MHz on the x-axis. The y-axis shows the ratio of the value of the total real-time required for the execution of the mentioned task sequence (see Figure 5) and the real-time critical constant. The upper curve depicts the case without all on-chip caches, which must be used to determine the worst case scenario for hard real-time conditions. If the PPC403 operates at 33MHz and the worst case scenario occurs (total cache misses and flushes), it is still able to guarantee the real-time requirement. If the clock frequency decreases, the y-value increases above one. That means, the microcontroller performance is undersized. If the frequency is higher than 33MHz, the system is oversized. This is why the Working Point (WP) at 33MHz is also called the optimal WP.

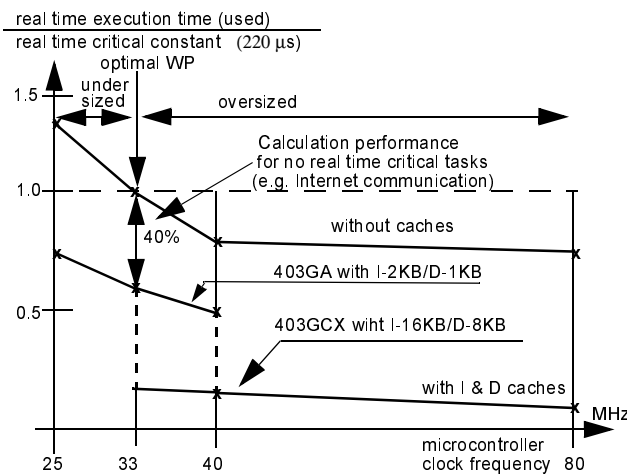


Figure 6: System Execution Resources

The lower curve shows the behavior with enabled D & I caches. This is the configuration in the normal operation state. The frequency at the optimal WP (33MHz) provides an average gain of about 40% in execution performance by using a PPC403GA. This 40% gain of the microcontroller execution performance can not be guaranteed in all cases. Therefore, these resources can only be used for non real-time tasks. In the range of 33MHz up to 80MHz, a further significant performance gain can be verified. This is due to the fact that the 403GCX type in that speed range has caches eight times larger than the 403GA type.

6 Results of the Analysis

We further analyzed the cache behavior at the optimal WP at 33MHz. In general, performance enhancements through caches are measured in dhrystones/sec and depict-

ed in the microcontroller data sheet. This chapter will show that this value cannot be used as an indicator to evaluate the performance for a fast reactive embedded system. From a system designers point of view, two key values are important to design such a system. The first one is the task switching time, which indicates the execution time used by the RTOS for suspending the current task and start the next available ready-state task. The second value is the interrupt reaction time, which indicates the time used by the RTOS between the encounter of the interrupt signal initiated by an external device and the first instruction executed within the *int*-service task.

The Table 1 shows these two values depending on different cache configurations by the PowerPC PPC403GA, which provides 2KB instruction and 1KB data caches.

PPC403GA 33MHz (WP)	without I-Cache without D-Cache	without I-Cache with D-Cache	with I-Cache without D-Cache	with I-Cache with D-Cache
task switching time	100% (87μs)	-1%	+46%	+60%
interrupt reaction time	100% (2.7μs)	-4%	+50%	+43%
dhrystones	100% (6211)	+10%	+187%	+455%

Table 1: Performance gain analysis for PowerPC PPC403GA with 2KByte I-Cache and 1KByte D-Cache

The main thing to be noted from Table 1 is the fact that the mentioned two key values increase only by a factor of 60% and 43% respectively in comparison to the performance without any caches, which is used as a base level. In contrast, the dhrystone value increases by a factor of 455%, promising a much higher performance gain. This gain is not applicable in real applications with fast external events and is about 10 times lower. A further interesting effect can be seen in the column without I-cache and with only D-cache. If the D-cache is very small (PPC403GA provide 1KB), there is no increase in performance, or there can even be a slight decrease.

Table 2 shows the influence of increasing cache sizes. This was done by using a PowerPC PPC403GCX-type, running under the same unchanged environment at the WP. This type differs from the PPC403GA only by the fact, that it provides cache sizes eight times larger.

PPC403GCX 33MHz (WP)	without I-Cache without D-Cache	without I-Cache with D-Cache	with I-Cache without D-Cache	with I-Cache with D-Cache
task switching time	100% (87μs)	+10%	+152%	+340%
interrupt reaction time	100% (2.7μs)	+12%	+205%	+377%
dhrystones	100% (6211)	+11%	+207%	+529%

Table 2: Performance gain analysis for PowerPC 403GCX with 16KByte I-Cache and 8KByte D-Cache

Table 2 stresses the fact, that if cache sizes increase (I-16KB/D-8KB), the performance gain for the two key values rises significantly, up to 340% and 377% respectively, which is in the same range of the promised gain by the dhrystone value of 529%.

The basic result of that investigation is that only large cache sizes like in the PPC403GCX leads to a significant performance gain when using a RTOS in fast reactive embedded systems. This is important due to the fact, that today most popular embedded microcontrollers, like the Motorola's MPC8xx- and 5xx-families or Intel's i960, provide relatively small cache sizes similar to the IBM PPC403GA.

In order to verify the results, which were obtained in the first step during the investigation of *VxWorks*, we implemented a second RTOS on SPYDER-CORE-P1. Therefore, we used the public domain RTOS *RTEMS* [11]. In this second step, we changed only the RTOS and then we repeated the measurements under the same environment conditions. That means we used the same hardware, benchmark application and code generation tool (gnu-C-Compiler with equal optimization level). The obtained results using *RTEMS* show the same behavior as using *VxWorks* with only very small differences (less than 5%). That result emphasizes, that the observed behavior is not bounded to a special RTOS, but is common behavior for a RTOS used in a fast reactive embedded system.

7 Summary

System performance is influenced by many indeterministic parameters. In order to meet all real-time requirements and exploit all available system resources, a detailed view of the internal behavior is necessary. Therefore, this paper presents the emulation of a fast reactive embedded system, running under control of two different RTOS, as a powerful method to solve the problems mentioned and shorten the design time and risks.

The results give answers to the four questions formulated in Section 2.2 and can be summarized as follows:

1. The emulation gives a detailed view of internal behavior in an early design stage and shows the optimal working frequency at 33MHz, which avoids an oversized or undersized system. The emulation reflects the final target system very closely without any change.
2. State of the art RTOS achieves task switching times of about 80 μ s and interrupt reaction times of about 30 μ s under worst case conditions. As shown in Figure 5, the total time the RTOS uses is 170 μ s and the total time for the application is 50 μ s. Therefore, 77% of the total execution time (220 μ s) is consumed by the RTOS. That means, if the external environment of a fast reactive system forces interactions (here 220 μ s) to the same de-

gree as the task switching time (here 80 μ s), the software overhead of the RTOS becomes a limiting factor for the total embedded system performance.

3. Caches, which are enabled, improve execution performance and provide a gain of 40%. If a system requires hard real-time conditions, this is done without cache enhancements. Such enhancements can only be used for non-real-time dependent system services, e.g., network communication via the Internet.
4. The cache sizes should be in the range of about 8-16KByte to provide a significant performance gain, if the application software is running under the control of a RTOS.

References

- [1] W. Wolf: *Hardware-Software Co-Design of Embedded Systems*. Proceedings of the IEEE, Vol. 82, No.7, July 1994
- [2] ____: *PPC403GA/GCX Embedded Controller - User Manual*. IBM Corporation 1997
- [3] Karlheinz Weiß, Thorsten Steckstor, Carsten Oetker, Ronny Kistner: *Data Sheet and User Manuel SPYDER-CORE-P1*. <http://www.fzi.de/weiss.html>, FZI & University Tübingen 1998
- [4] A. Hergenhan, Christoph Weiler, Karlheinz Weiß, Wolfgang Rosenstiel: *Value-Added Services in the Industrial Automation*. ACoS'98, Lisabon Portugal, April 1998
- [5] Sanjaya Kumar, James H. Aylor, Barry W. Johnson, Wm. A. Wulf: *The Codesign of Embedded Systems*. Kluwer Academic Publishers, 1996
- [6] Rajesh Kumar Gupta: *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Kluwer Academic Publishers, 1995
- [7] S. Hauck, G. Borriello, C. Ebeling: *Springbok: A Rapid-Prototyping System for Board-Level Designs*. ACM/SIGDA 2nd. International Workshop an Field-Programmable Gate Arrays, Berkley, Feb. 1994
- [8] G. Koch, U. Kebschull, W. Rosenstiel: *A Prototyping Environment for Hardware/Software Codesign in the COBRA Project*. Third International Workshop on Hardware/Software Codesign, 1994
- [9] Karlheinz Weiß, Thorsten Steckstor, Carsten Oetker, Ronny Kistner: *Data Sheet and User Manuel SPYDER-ASIC-X1*. <http://www.fzi.de/weiss.html>, FZI & University Tübingen 1998
- [10] ____: *VxWorks Reference Manual and Programmer's Guide*. WindRiver Systems, Edition 1, 1997
- [11] ____: *Real Time Executive for Multiprocessor Systems (RT-MS) Development Environment Guide*. U.S. Army Missile Command, Redstone Arsenal, Alabama 35898-5254, Release 3.5.1, January 1996