Hardware Support for Real-Time Systems – an Overview

Susanna Nordström
Mälardalen University
+46 (0)21 470 21 95
susanna.nordstrom@realfast.se

ABSTRACT

Research at the Computer Architecture Laboratory (CAL) at Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, has been performed in implementing a real-time kernel in hardware in order to increase system predictability, determinism and performance. The first article was published in 1991 and since then the real-time kernel has been further developed, adjusted to multiprocessor systems and been exposed to different kinds of benchmarks and comparisons to software implemented real-time operating systems. It has also been developed to a commercial product as an intellectual property component. This article is a survey focusing on describing previous work on the real-time kernel at CAL, also covering most important related work and the conclusions drawn from the research over the years.

Keywords

real-time operating system, real-time kernel, multiprocessor system, scheduling co-processor, hardware architecture

1. INTRODUCTION

A real-time system is a system that reacts on events in the environment and executes functions based on these within a precise time. In these systems, time is a vital parameter and the behaviour of the system is only considered correct if the correct result is presented within a specified time limit. Real-time systems are classified in soft- and hard real-time systems. Soft real-time systems can tolerate timing failure to a certain degree while a hard real-time system must fulfil both functional and timing demands completely or it could lead to disastrous events. [1] A real-time operating system (RTOS) is an operating system that is implemented for real-time systems in order to simplify design, execution and maintenance of real-time systems and applications. The RTOS provides the designer with a programming interface to the underlying hardware. [2]

The most common way of implementing real-time operating systems is to do it completely in software. In the Computer Architecture Laboratory (CAL) at Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, research has been performed on a real-time kernel in hardware, i.e. it is implemented in VHDL and used in integrated circuits, Field Programmable Gate Arrays (FPGA) or Application Specific Integrated Circuit (ASIC). The FPGA technology, released in 1985 introduced flexibility to the field of integrated circuits. An FPGA can be reprogrammed an infinite number of times and this has made it possible to implement established software algorithms in hardware, in this research that is the real-time kernel activities. [3] This means that scheduling, inter process communication, interrupt management, resource management, synchronization and time management control are implemented in hardware. This makes it possible to utilise hardware characteristics such as parallelism and determinism that consequently decreases system overhead, improves predictability and increases response time.

The first article was published in 1991 [4] describing how the kernel is implemented in a single processor system. Since then the real-time kernel has been further developed, adjusted to multiprocessor systems and been exposed to different kinds of benchmarks and comparisons to software implemented RTOS. It has also been developed to a commercial product as an intellectual property component [5].

There have been a few similar publications on real-time kernels implemented in hardware but most related work is special purpose RTOS co-processors or RTOS standard processors.

This paper is organized as follows. Section 2 is an overview of the real-time kernel developed at CAL. Section 3 covers the research-projects where the CAL kernel has been in focus, section 4 describes related work of other research projects and section 5 is a summary of conclusions from mainly previous work but also from related work.

2. OVERVIEW

A real-time kernel handles an operating system's scheduling activities. In the real-time kernel developed at CAL, this is implemented in hardware and is used together with a software driver of 2 Kb code (also called API, Application Programmers Interface) which makes it possible for the programmer to utilise the hardware, i.e. handle the service calls to the kernel.

The hardware part together with the software driver makes a complete RTOS kernel. It can also be used together with another pure software RTOS, working as an operating system accelerator.

The scheduling in hardware is pre-emptive and is executed in parallel to the CPU and the CPU is interrupted only when a task switch is to take place. The communication with the CPU is carried out through registers.

Except for the first prototype, the kernel is communicating with the CPU through the bus. The outer technology dependent bus interface can be adjusted to support any CPU.

During the years of development, the real-time kernel has had many configurations but generally it consists of internal components (see figure1); an interface, a scheduler, an interrupt handler, a resource manager and a time manager that as a whole supports 16 tasks or more, priority levels, external interrupts, semaphores, flags, watchdogs, timers for delay and periodic start of tasks.

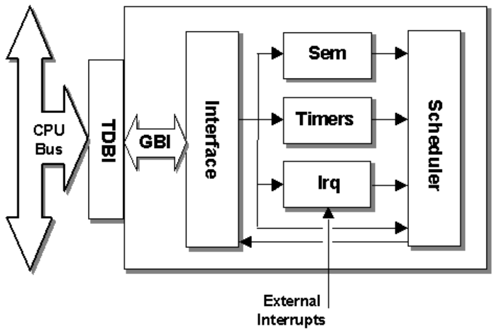

Figure 1. Example of the CAL real-time kernel architecture. [1] **[5]**

The Interrupt Service Routines (ISR) of external interrupts have priorities and are handled and scheduled like ordinary tasks.

3. PREVIOUS WORK

This section covers research-projects at the CAL where the real-time kernel has been involved. It is presented with the kernel in focus.

3.1 Realising a real-time kernel for single processor systems

Obtaining absolute timing determinism is one of the main reasons given why the kernel was implemented in hardware [4] [6]. The first prototype of the kernel, FASTCHART, was a µ-processor with an integrated hardware kernel in one chip. In the second prototype, FASTCHARD, the kernel was implemented as a separate unit on a stand-alone chip, adjusted to a CPU [7]. Further developments have been implemented based on the second approach.

The FASTCHARD was a part of a minor real-time system consisting of a CPU, main memory and I/O ports. The system bus and an interrupt line connected the CPU and FASTCHARD. Eight external interrupts were connected; it contained seven registers and task-queues implemented in Random Access Memory (RAM).

3.2 Multiprocessor real-time kernel in hardware

When the hardware kernel was introduced into multiprocessor systems it was called real-time unit (RTU). The first version with support for multiprocessor systems was called RTU94, followed by RTU95. Both RTU94 and RTU95 could handle three CPUs (see figure 2). More functionality and improvements were added. The number of function calls was increased to include semaphores, event flags and watchdogs and a real-time clock for continuous supervision of time was added. This means that all administration of resources that have any kind of time dependency has to be supported by the RTU, e.g. distribution of CPU time, semaphores, flags and sorting of queues. [8]

[1] TDBI – Technology Dependent BusInterface, GBI – Generic Bus Interface

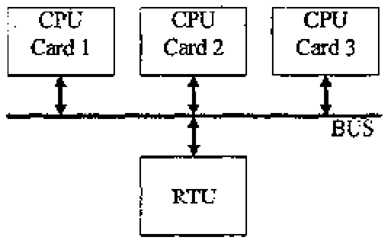

Figure 2. RTU in multiprocessor system architecture. [9]

To support a multiprocessor environment, the RTU was implemented to consist of one scheduler for each CPU and tasks could be initialised to execute on a fixed CPU (local task), or on any CPU (global task). There was one ready queue for each CPU and one queue for the global tasks. Each scheduler checks both the local and the global queues. [10] Since the RTU has knowledge of the load of each CPU it can be used for dynamic load balancing. [8] In [9] the RTU was presented as a co-processor in multiprocessor environments.

In [11], [12] and [13] the RTU was used in a research project in multiprocessor systems called Scalable Architecture for Real-Time Applications (SARA). The SARA-system is based on the idea to incorporate as many parts of a real-time operating system into hardware as possible. The scalability of the SARA-system could be used in the transition from a single processor system into a multiprocessor system. The RTU handled the scheduling of the system. In [11] the RTU based dynamic scheduling decisions on extra observability in the form of load information from bus-monitors.

3.3 Benchmarking and comparisons

In [14] and [15] the hardware kernel, the RTU, is used *together* with a pure software RTOS and is handling the scheduling activities. This results in the hardware kernel accelerating the software RTOS. Figure 3 illustrates the RTUs placement in the system.

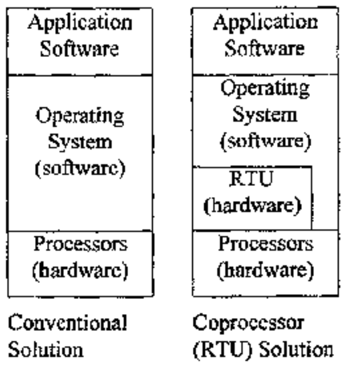

Figure 3. Overview of a software and hardware implemented real-time kernel solution. [9]

3.3.1 Benchmarking of application response time and clock tick administration

In [14], the real-time kernel is called "booster" and the functionality is reduced to merely consist of the scheduling part (not semaphores, flags etc.). It is used together with a RTOS implemented in software. Benchmark of a model of a telecommunication application running in three different systems was performed. The systems were:

1. A processor supervised by a commercial single processor RTOS.
2. A processor supervised by a RTOS with the booster.
3. Two processors supervised by a RTOS with the booster.

Application response time and RTOS overhead for clock tick administration was measured with and without data located in local or global accessed memory, or cache memory.

The conclusions were that a real-time kernel in hardware (booster) decreases the application response time, a fast memory system decreases the difference in using and not using a hardware kernel and the clock tick administration is zero when using a hardware kernel.

3.3.2 Comparison of SoC architectures with associated RTOS - in hardware, hardware/software and pure software

In [15], the real-time kernel in hardware, here referred to as the Real-Time Unit (RTU), was used in a performance comparison with two other RTOSes: a pure software Atalanta RTOS[2], and a hardware/software RTOS composed of part of Atlanta RTOS interfaced to a System-on-a-Chip Lock Cache[3] hardware (SoCLC). The SoCLC is a hardware support to accelerate software locks and semaphores in a software RTOS [16]. All systems contained three processors running a database application with many different task level synchronization scenarios. A framework to generate the three system configurations was used.

In measuring the average-case simulation time, the RTU system showed best performance, a 50% speed-up over case performing on 6 tasks and 36% speed-up performing on 30 tasks, compared to the pure software RTOS. The RTU system also had best performance when number of clock-cycles spent on communication, context switch and computation was measured.

3.3.3 Comparison of hardware RTOS and software RTOS

In [17] a performance comparison between the real-time kernel in hardware and a corresponding kernel in software, in a multiprocessor system, was done. The software kernel was especially implemented for this comparison using almost the same API as the hardware kernel uses.

The speed-up achieved with the hardware kernel was 2.6 times. Other important results were that the time for creating tasks in the software kernel increases with number of tasks while it is constant in the hardware kernel. This is because of list management that increases with number of tasks in a software kernel.

It was discovered that the software kernel was faster when tasks was created on a master node, since it can draw benefits from using system cache in this case while the hardware kernel suffers from Peripheral Component Interconnect (PCI) bus access latencies.

[2] An open source multiprocessor RTOS developed at the Georgia Institute of Technology, USA.

[3] Developed at the Georgia Institute of Technology, USA.

3.4 Realising special purpose hardware components utilising the real-time kernel in hardware

Having the kernel implemented in hardware makes it possible to create other hardware components that can benefit from the fact that they can be integrated to the kernel in different useful aspects.

3.4.1 Monitoring RTOS kernel activities

Multiprocess Application Monitor (MAMon) [2] is a non-intrusive monitor that gives observability into the execution of a single- or multiprocessor system supporting the real-time kernel in hardware. MAMon is an integrated solution to on-chip monitoring of system-level events in real-time systems. The observability comes from a probe unit, which is integrated at the rtl-level of the hardware kernel, detecting and collecting events regarding process execution, communication, synchronisation and I/O interrupt activities. Collected events are time stamped and transferred to a separate computer system hosting an event database and a set of monitoring application tools that shows the results graphically.

3.4.2 Interprocess Communication Support

[18] describes a hardware implementation of asynchronous Interprocess Communication (IPC) in an RTU based architecture. It was investigated how performance and message flow in a message intense system could be increased by adding some functionality, like message priority, to the IPC functions and implement it in an RTU architecture. This resulted in an IPC-RTU, the ordinary RTU with an augmentated instruction set. The IPC implementation supported message priority, priority inheritance on message arrival and task time-out on message send/receive. The IPC administration, sorting message queues etc. was placed in the RTU. The conclusions were that it is possible to implement IPC in hardware but that the design becomes too big to fit into one FPGA.

3.5 Hardware kernel energy consumption

In order to study the RTUs impact on system energy consumption, an energy characterisation of the RTU was performed in [19]. The results obtained showed that the power consumption is independent of what function the RTU performs and that power consumption during idle periods are approximately the same as during system calls. The conclusion was that the RTU needs to be power optimised, using techniques such as gated clocking, in order to beat a SW based RTOS. For applications that use the RTOS functions extensively, a power optimised RTOS hardware accelerator, like the RTU, would be justified.

4. RELATED WORK

The related work section is limited to research projects of hard real-time systems.

The related work of the real-time kernel includes two projects also implementing kernel activities in hardware, TRON-project and F-timer solution, and research in utilising co-processors to accelerate scheduling; special purpose RTOS co-processors or standard RTOS co-processors.

4.1 The TRON-project

The Real-time Operating System Nucleus project (TRON) with the aim of creating an ideal computer architecture, started in 1984. The Industrial TRON (μITRON) is a subproject of TRON and is an architecture for the real-time operating system for embedded computers. It is used as the real-time multitasking operating system for intelligent objects. [20]

[21] presents a high performance real-time OS using VLSI technology. The solution consists of a hardware part, called "silicon TRON", and a software part, called μITRON. The concept is illustrated in figure 4.

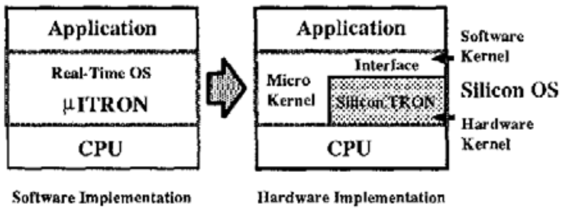

Figure 4. The concept of the Silicon OS. [21]

The hardware part implements the scheduler with the system call functions and the software part implements other system call functions and interface processes between applications and the hardware part. The Silicon TRON together with the μITRON is called a "Silicon OS". Like the CAL kernel, this solution also communicates with the CPU with register and interrupts for task switch.

The time measurements on system calls regarding flags and semaphores showed that the system call processing time in hardware can be reduced to 130 to 1880 times faster than a conventional implementation in software.

The latest article presented in 2003 on μITRON is a pure software solution and does not use the hardware part "Silicon Tron". [22]

4.2 F-Timer

[23] presents a hardware architecture for real-time operating systems support using special hardware components implemented in one FPGA.

The F-Timer is a co-processor that communicates with the microprocessor and releases the processor of the tasks time management. The F-Timer hardware architecture handles external asynchronous interrupts and scheduling of tasks with priority. All tasks are programmed and when the execution time of a certain task is reached, the microprocessor is interrupted and the correct task is available at the bus. It can handle 32 tasks.

A similar software solution, based on a microcontroller, was created for comparison. Measuring the performance, the conclusion was that the software solution was 18 times worse than the F-Timer hardware architecture solution.

4.3 The Spring Scheduling Co-processor

The Spring scheduling co-processor (SSCoP) is a VLSI (Very Large Scale Integration) accelerator for distributed multiprocessor real-time systems. It can be used for both static and on-line scheduling. The SSCoP accelerates the execution of the critical activity of the Spring scheduling algorithms used by the Spring operating system kernel. It speeds up the algorithm by three orders of magnitude [24]. The resulting reduction in scheduling latency will enable real-time systems to handle tasks with shorter deadlines compared to scheduling done in software.

The Spring kernel is for example based on the ideas of integrated CPU scheduling and resource allocation in use of the scheduler in a planning mode, enhancing the system determinism. There is a scheduler on the main system node and an application dispatcher for each application node in the system, which is responsible for the dispatching for the application tasks. The scheduler and applications dispatcher processes are designed to run in parallel. Concurrent execution of the scheduler and the multiple dispatchers are achieved by reserving a set of tasks for each dispatcher, where the scheduler is not free to reschedule the tasks reserved for the dispatchers. Each dispatcher has tasks to execute while the scheduler is trying to reschedule the remaining tasks to guarantee the new task. [25]

The Spring co-processor is designed to plan work dynamically into the future to meet the deadline of currently active tasks. Some of the differences between the CAL real-time kernel and the Spring co-processor is that it only guarantee that the tasks in the executing state are those with highest priority among all the tasks in the ready state. [9]

4.4 MARS

The Maintainable Real-Time System project (MARS) [26], is a distributed computer system that consists of a number of autonomous, fail-silent node computers. They are interconnected by a real-time network. Each node is a self-contained computer with a local real-time clock and an interface to the real-time network. It is controlled by the MARS Operating System and executes a set of application tasks. Communication among components and tasks is achieved by exchanging broadcast messages. There is no explicit synchronisation between tasks, all component activities are implicitly synchronized using the global time.

The tasks and messages are scheduled prior to the run time of the application, statically scheduled, in a way that guarantees that all deadlines will be met.

Special for MARS is a set of actively redundant components, combined to form a Fault-Tolerant Unit (FTU). The FTU handles the nodes failures.

4.5 A holistic approach to real-time system design

The research in [27] is focusing on a holistic approach to real-time system design instead of certain parts. The research project is designed in a set of layers where predictable behaviour is a focus at every layer. In the hardware layer, the parallel computer hardware architecture consists of one main processor and three co-processors in an asymmetrical multiprocessor concept:

- Main processor
- Kernel co-processor
- Arithmetic co-processor
- Data access co-processor

The main processor is the task processor and it is an independently operating control system on which the application programs are run. The operating system kernel services are migrated out to the specialized kernel co-processor. It recognises external signals, administrates time events and monitors access to shared variables and synchronisers. An arithmetic co-processor is provided to support numerically intensive computation and finally there is an external data access processor that supports external variables and peripheral device access. This co-processor also handles the saving of contents occurring during a context switch. The co-processors are connected point-to-point with each other.

4.6 HARTIK

HARTIK [28] is a hard real-time kernel for programming robot tasks with explicit time constraints and guaranteed execution. It is specifically designed to develop predictable robotics applications. To adjust to multi-sensor robotic systems and to be flexible in expressing timing constraints, HARTIK handles four types of tasks; hard tasks, sporadic tasks, soft tasks and non-real-time tasks. Also a dynamic pre-emptive scheduling with guarantee is used, the system performs a schedulability analysis to see if a critical task is to meet its timing constraints and if it is not, the programmer is notified. A particular one-to-many communications mechanism is used, the Cyclic Asynchronous Buffer (CAB), which is designed for the communication among periodic activities and eliminates unpredicted delays.

5. CONCLUSIONS

Implementing a real-time kernel in hardware makes it possible to draw benefits from hardware characteristics such as parallelism and determinism.

The execution time of real-time functions gets deterministic and task switch can be performed without any CPU time delay. [6] [23] When real-time kernels are implemented in software, one of the disadvantages is that the execution time for the service calls will have a minimum and a maximum time. The time gap can be big and the worst-case time is one of the factors that will decide the utilisation factor of the system. The scheduling time varies with the number of tasks and scheduling algorithm and must be bounded by a pessimistic worst case execution time, which decrease the determinism. In hardware, the time gap can be designed to be close to 0, which leads to predictable time behaviour, simpler timing analysis of the system and almost no overhead. It is also easier to debug tasks since different protection modes are not required. [29] [10] [15] [30]

A hardware kernel executes in parallel to the CPU, which relieves pressure from the CPU which gets almost 100% execution time for the application tasks. There is less software code in memory since the functionality is implemented in hardware instead. [7] [23]

A software OS will generate a clock tick interrupt to the CPU when it is executed. Also when the lists of tasks (queues) are worked at and new periodic delay times are calculated for the tasks. With the hardware kernel in the system, it checks all queues concurrently and only generates an interrupt to the CPU when there is to be a task switch. [29] [31]

Another advantage of having the kernel in hardware is the possibility to use complex scheduling algorithms, unlimited of different queue types without any performance loss. Also there is an improved understandability and complexity reduction when the system is divided into parts. [29] [10]

Hardware based RTOS is not energy efficient compared to software RTOS. Even during idle periods there are big amounts of power wasted due to unwanted activity triggered by the clock. [18] Power consumption of FPGAs is not ideal, when low power design is an issue. [23]

6. REFERENCES

[1] Norström, C., Sandström, K., Mäki-Turja, J., Hansson, H., Thane, H. and Gustafsson, J. *Robusta realtidssystem*. (MRTC, Mälardalen University, Västerås, Aug. 2000).

[2] Mohammed, E.S. *On-Chip Monitoring for Non-Intrusive Hardware/Software Observability*. IT Licentiate Thesis, ISSN 1404-3041, ISRN MDH-MRTC-120/204, (Department of Information Technology, Uppsala University, Sweden and Department of Computer Science and Engineering, Mälardalen University, Sweden, 2004).

[3] Lindh, L. and Sjöholm, S. *VHDL för konstruktion*. ISBN 91-44-01250-0, 3rd edition, 1999.

[4] Lindh, L. and Stanischewski, F. *FASTCHART – A Fast Time Deterministic CPU and Hardware Based Real-Time-Kernel*. In IEEE, Euromicro workshop on Real-Time Systems, (June 1991).

[5] RealFast Intellectual Property AB, Skivfilargränd 2, 721 15 Västerås, Sweden, www.realfast.se [cited 2004-10-01]

[6] Lindh, L. and Stanischewski, F. *FASTCHART – Idea and Implementation.* In IEEE International Conference on Computer Design (ICCD), Boston, USA (Oct. 1991).

[7] Lindh, L. *FASTCHARD – A Fast Time Deterministic HARDware Based Real-Time Kernel*. In IEEE International Conference on Computer Design (ICCD) (Cambrdge, USA, Oct. 1991).

[8] Lindh, L. and Vörös, P. *Applikationsanpassad Real-Tids Co processor till styrsystem för industrirobotar. (Applicationadjusted Real-Time Co-processor in controlsystems for industrial robots)*. In Proceedings in Electronic Design Automation. (April, 1996).

[9] Stärner, J., Adomat, J., Furunäs, J. and Lindh, L. *Real-Time Scheduling Co-Processor in Hardware for Single and Mulitprocessor System.* In 'Beyond 2000: Hardware and Software Design Strategies', Proceedings of the 1996 EUROMICRO Conference, p. 509-512, (Prague Czech Republic, Sep. 1996).

[10] Lindh, L., Stärner, J. and Furunäs, J. *From Single to Multiprocessor Real-Time Kernels in Hardware*. IEEE Real-Time Technology and Applications Symposium. (Chicago, May, 1995).

[11] Klevin, T. and Lindh, L. *Scalable Architecture for Real-Time Applications And Use of bus-monitoring*. In Proceedeings of Sixth International Conference on Real-Time Computing Systems and Applications, RTCSA'99, p. 208-211, (Dec. 1999).

[12] Lindh, L., Klevin, T. and Furunäs, J. *Scalable Architecture for Real-Time Applications – SARA.* Swedish National Real-Time Conference SNART99 (Linköping, Sweden, Aug. 1999).

[13] Enblom, L. and Lindh, L. *Adding Flexibility and Real-Time Performance by Adapting a Single Processor Industrial Application to a Multiprocessor Platform*. In 'Parallel and Distributed Processing', Proceedings of the 2001 EUROMICRO Workshop, p. 487-490, (Mantova, Italy, Feb. 2001).

[14] Furunäs, J. *Benchmarking of a Real-Time System that utilises a booster*. In International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA200), (June, 2000).

[15] Lee, J., Mooney, V. J., Ingström, K., Daleby, A, Klevin, T. and Lindh, L. *A Comparison of the RTU Hardware RTOS with a Hardware/Software RTOS.* In Proceedings of the ASP_DAC 2003, Design Automation Conference, p. 683-688, (Jan, 2003).

[16] Akgul, B. S., Lee, J. and Mooney, V. J. *System-on-a-Chip processor synchronization hardware unit with task preemption support.* International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '01), p 149-157. (Nov. 2001).

[17] Samuelsson, T., Åkerholm, M., Nygren, P., Stärner, J. and Lindh, L. *A Comparison of Multiprocessor Real-Time Operating Systems Implemented in Hardware and Software.* International Workshop on Advanced Real-Time Operating System Services (ARTOSS), (Porto, Portugal, 2003).

[18] Furunäs, J., Adomat, J., Lindh, L., Stärner, J. and Vörös, P. *A Prototype for Interprocess Communication Support, in Hardware.* In 'Real-Time Systems', Proceedeings in 1997 EUROMICRO Workshop, p. 18-24, (Toledo, Spain, June 1997).

[19] Haukilahti, R. *Energy Characterization of a RTOS Hardware Accelerator for SoCs.* In System-on-Chip Conference, (Falkenberg, Sweden, 2002).

[20] *Overview of the TRON Project.* In TRON Project International Symposium 1995. Proceedings of the 12th. p. 100-104. (Dec. 1995).

[21] Nakano, T., Komatsudaira, Y., Shiomi, A. and M. Imai. *VLSI Implementation of a Real-time Operating System.* Proc. of ASPDAC '97, pp. 679-680. (Jan. 1997).

[22] Srinivas, N.S., and Shankaran, S. *μITRON customization for use in Automated Test Equipment*. AUTOTEST 2003, IEEE Systems Readiness Technology Conference. Proceedings, p. 699 – 702. (Sep. 2003)

[23] Parisoto, A., Souza, A., Jr., Carro, L., Pontremoli, M., Pereira, C. and Suzim, A. *F-Timer: dedicated FPGA to real-time systems design support.* In Real-Time Systems, In Proceedings if the Ninth Euromicro Workshop, p.35 – 40. (June, 1997).

[24] Burleson, W., Ko, J., Niehaus, D., Ramamritham, K. Stankovic, J.A., Wallace, G. and Weems, C. *The spring scheduling co-processor: a scheduling accelerator.* Computer Design: VLSI in Computers and Processors, 1993. ICCD '93 in Proceedings of the 1993 IEEE International Conference p. 140 – 144. (Oct. 1993).

[25] Stankovic, J. and Ramamaritham, K. *The Spring Kernel: A new Paradigm for Real-Time Systems.* In Software IEEE Volume 8, Issue 3, 1991. p. 62-71. (May 1991).

[26] Kopetz, H., Fohler, G., Grunsteidl, G., Kantz, H., Pospischil, G., Puschner, P., Reisinger, J., Schlatterbeck, R., Schutz, W., Vrchoticky, A. and Zainlinger, R. *Real-time system development: The programming model of MARS.* In Autonomous Decentralized Systems, 1993. Proceedings. ISADS 93., International Symposium . p. 290 – 299. (March, 1993).

[27] Colnaric, M., Verber, D. and Halang W. A. *Design of Embedded Hard Real-Time Applications with Predictible Behaviour.* In Real-Time Applications, Proceedings of the IEEE Workshop. (1993).

[28] Buttazzo, G. C. *HARTIK: A Real-Time Kernel for Robotics Applications. In* Real-Time Systems Symposium, Proceedings. P. 201 – 205. (Dec. 1993).

[29] Lindh, L. *A Real-Time Kernel implemented in one chip*. In IEEE press, Real-Time Workshop (Oulu, June 1993).

[30] Adomat, J., Furunäs, J., Lindh, L. and Stärner, J. *Real-Time Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems*. In Proceedings of the 1996 Euromicro Workshop on Real-Time Systems. (L'Aquila, Italy, June, 1996).

[31] Lindh, L. *Utilization of Hardware Parallelism in Realizing Real Time Kernels*. Doctoral Thesis, TRITA – TDE 1994:1, ISSN 0280-4506, ISRN KTH/TDE/FR--94/1--SE, (Department of Electronics, Royal Institute of technology, Stockholm, Sweden, 1994).