Configurability and Real-Time Operating Systems with Hardware Support - A State-of-the-Art Report

Susanna Nordström
susanna.nordstrom@mdh.se
Department of Computer Science and Electronics
Malardalen University, Vasteras

February 6, 2008

Abstract

Configurable real-time operating systems has the ability to be customized in order to only include services used by a real-time operating system application. This report gives an overview of research performed in the area of hardware support for real-time operating systems with focus on provided functionality, the hardware/software partitioning of provided functionality and to what extent the functionality is configurable.

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 behavior of the system is only considered correct if the correct result is presented within a specified time limit [23]. 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 [8].

The most common way of implementing RTOS functionality is to do it completely in software. However, a real-time kernel in hardware, the Real-Time Unit (RTU), has been developed at Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Sweden. The RTU is implemented in VHDL and used in integrated circuits such as Field Programmable Gate Arrays (FPGA). The RTU has been a topic for research in both uni- and multiprocessor projects since 1991, when the first article was published by Lindh *et al.* [14]. Previous work on the RTU concerns for example creation and proof of concept [14, 15, 11, 12], extended implementation for inclusion in multiprocessor systems [16, 29, 13, 5, 2] and further benchmarking and performance comparisons with software solutions [4, 9, 28]. Previous work on the RTU has shown motivations for having the real-time functionality implemented in hardware. The deterministic characteristics of hardware improves the predictability of the time behavior in a hardware implemented RTOS since the time gap between the service call minimum and maximum time is decreased. Another characteristic of hardware implementations is concurrency, which can be utilized in hardware support and its internal components, enabling quick response. In software real-time kernels, the CPU has to execute code for clock-tick administration and task waiting queues. In the RTU and other hardware support, the CPU load is reduced because no CPU execution time for clock-tick administration is needed. Neither needs the CPU be involved in task queue handling since the hardware support performs this administration in parallel to the CPU. This results in reduced system overhead and faster service call response time. Further, the memory footprint is reduced since functionality is placed in hardware instead of software.

However, RTOS with hardware support, like the Real-Time Unit (RTU), has been criticized for not being adjustable to the same extent as software real-time kernel solutions are. This is one of the reasons that may prevent system designers from choosing a real-time kernel with hardware support when building a system, even though they could benefit from characteristics associated with hardware support described earlier. Another disadvantage with hardware support is the occupation of hardware resources. Hardware support occupies FPGA area and sometimes a small amount of memory, a conventional software RTOS consumes none FPGA area.

One aspect that addresses these issues is configurability. Most software RTOS for embedded systems and FPGA designs are, to some extent, configurable in order to be customizable and not use more system resources than necessary, an important matter when the RTOS is used in resource restricted environments. When the kernel is implemented in hardware, like the RTU and other hardware support, not only memory footprint is motivation for configuration, the number of logic cells occupied in the FPGA has to be considered as well. Adding configurability to hardware supported RTOS makes them more adjustable and addresses the FPGA area occupation issue when the hardware support part can be configured for minimum FPGA area resource usage.

This state-of-the-art report intends to give an overview of related work in the area of hardware support for real-time operating systems in connection with configurability. While there have been much work on performance analysis of hardware support, this report focus on provided functionality, the hardware/software partitioning of provided functionality and to what extent the functionality is configurable. Configurability of RTOS is described in Section 2 followed by Section 3 containing a an description of eight different hardware support followed by a Summary in Section4.

2 Configuration

In conventional software based RTOS, a lot of functionality is available in order to satisfy a large variety of customer requirements. For this reason, many RTOS are configurable in order to only include functionality that will be used in the actual application. Configuration possibilities make it possible not to use more system resources than absolutely necessary, motivated by decreased memory usage.

Configurability broadens the use of a particular RTOS since it can be used in a wider context when it can provide both limited and extended functionality. By being able to use as small amount of resources as possible, a cheaper target device can be used, which decreases the end product cost. Further, configurability is a possibility for a system to grow if new system requirements arise over time, extending system lifetime.

We use the term RTOS configurability to describe the ability for the system designer to enable desired functionality, and disable undesired functionality, among functionality provided by an RTOS at compile time. RTOS configuration may concern enabling/disabling of RTOS functionality regarding:

- Tasks
- Scheduling priorities
- Inter process communication (IPC) such as semaphores, flags, event management, message queues, message boxes and mutexes.
- Timing functionality
- Stack usage
- Memory allocation
- Processor endianess
- Processor datawith (8, 16, 32 or 64 bit wide)
- Error checking
- Debugging possibilities

Pre-runtime RTOS configurations are performed by defining the services needed at compile time. This can be accomplished by setting flags, or in the case of libraries, having the linker include the services used by the application [3]. The programmer can use an RTOS/OS specific GUI provided in the system development tool or use a specific configuration file.

When an RTOS is partly implemented in hardware, not only memory footprint is motivated for configuration; occupied amount of FPGA area for the hardware support has to be considered as well.

Pre-runtime configuration of hardware components is performed using generic VHDL design. Generic design makes it possible to pass information into a design description of a component by setting generic parameters, e.g. the size of an input port [1]. By doing this, the size can be varied according to system requirements. Similar to configuring software implementations, the programmer may set the generic parameters in connection with a specific GUI provided in the development tool or use a specific configuration file.

What kind of functionality that is configurable in the hardware part of a hardware/software implemented RTOS is quite distinctive as well as the implementation of the hardware part itself. Following Section 3, includes on overview of different solutions to hardware support and remarks on configurability.

3 Hardware Support for Real-Time Operating Systems

The conventional way of implementing RTOS functionality is in a software programming language, executed by a general purpose CPU. The term hardware support for RTOS describes RTOS functionality that has been implemented with a hardware description language (HDL), e.g VHDL or Verilog. To what extent the hardware support implements RTOS functionality is dependent on research project but the RTOS application, including system tasks are implemented in software and executed by the system CPU.

Hardware support may include RTOS functionality such as scheduling, inter process communication (IPC), interrupt management, timing management, clock-tick administration, context switch routine, task control block (TCB), task queues and other resource queues. The hardware support may have a specific software driver executed by a CPU, that utilizes the hardware implemented functionality. A common way for a software driver to communicate with the hardware support part is through memory mapped registers over a CPU bus.

Hardware support for RTOS can be required to be used together with a general purpose CPU or be closely integrated with a special purpose CPU. If a general purpose CPU is used, the CPU registers can only be manipulated with software programming and the context switch routine must be implemented in software as in conventional software RTOS. If a special purpose CPU is used, perhaps implemented in HDL, the CPU registers is reachable and the context switch can be performed in hardware in one or a few system clock cycles. The hardware support may be classified as a co-processor since it performs specific functionality in parallel to a CPU that executes the system application. The related work presented in the following sections have different approaches to this.

Following sections presents an overview of research performed in the area of hardware support for RTOS. Each hardware support project will be described in the aspect of chosen RTOS hardware/software partitioning, implemented functionality and remarks on configurability. Each project is summarized in a table where provided functionality is shown regarding hardware or software implementation. Task needs special attention since a tasks can be marked being implemented in both hardware and software. This is because a task is considered implemented in software when there exist application tasks executed by a main CPU and when a context switch routine is implemented in software requiring a software implemented task control block (TCB, a structure containing task information regarding task entry point, stack, timing, priority and state information). A task is considered being a part of the hardware implementation when the scheduler is implemented in hardware, containing tasks and task information in task queues.

3.1 Real-Time Unit (RTU)

As described earlier, the RTU originates from the work of Lind *et al.* [14] and has been a topic for different directions of research and has consequently had different implementations. This section will describe the latest RTU single-processor solution implemented to be configurable [22].

Table 1: Implemented functionality in the RTU [27].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks (periodic, aperiodic)	HW and SW	Number of (2 to 512)
Scheduling algorithm (Priority based)	HW	
Timers	HW	Size of argument (up to 16 bits)
External interrupts	HW	Number of (2 to 256)
RTOS clock-tick processing	HW	Resolution
Context switch routine	SW	
IPC:		
Semaphores	HW	Number of (2 to 1 024)
Flags	HW	Number of (1 to 26 flag-bits)

The hardware part of the RTU is implemented in hardware (using hardware description language, VHDL) and contains the scheduling, IPC in the form of binary semaphores and flags, external interrupt management and time management control. The hardware implementation is utilized through memory mapped registers and is used together with a small software driver, also called application programmers interface (API). It makes it possible for the programmer to utilize the hardware, i.e., transfer the service calls to the kernel. The RTU component is connected to a general purpose CPU with a bus interface and an internal interrupt to notify the CPU when a taskswitch is about to occur. Context switch routine, task and TCB is implemented in software but task queues are implemented in hardware. Functionality and configuration possibilies are described in Table 1.

3.2 The δ Hardware/Software RTOS Framework and the Configurable Hardware Scheduler

Mooney *et al.* have implemented the δ hardware/software RTOS framework, a hardware/software generation tool for multiprocessor system-on-chip designs [18, 10, 17]. The motivation is to simplify and speed up the design process of creating a hardware/software co-design system with automatic generation of a complete RTOS from predesigned hardware/software RTOS components.

The hardware components are implemented in hardware description language Verilog and the software components in C. The δ framework has a hardware RTOS library containing three components: a system-on-chip lock cache (SoCLC, a mechanism for protecting and synchronizing use of critical sections in a multiprocessor system) a system-on-chip deadlock detection unit (SoCDDU) and a system-on-chip dynamic memory management unit (SoCDMMU).

Table 2: Implemented functionality and configuration options in the δ hardware/software RTOS framework [18, 10, 17].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks	SW	Number of
Scheduling (Priority based)	SW	
RTOS clock-tick processing	SW	
Context switch routine	SW	
IPC:		
Semaphores	SW	Enable/Disable
Event groups	SW	Enable/Disable
Mailboxes	SW	Enable/Disable
Queues	SW	Enable/Disable
Mutexes (mutual exclusion objects)	SW	Enable/Disable
Special features:		
Processors (PowerPC or ARM)	HW	Number of
Deadlock detection	HW or SW	Enable/Disable
Dynamic memory management	HW or SW	Enable/Disable
SoC lock cache	HW	Enable/Disable

The δ framework software RTOS library includes their own Atlanta RTOS for multiprocessor systems (including priority-based preemptive scheduler, semaphores, mailboxes, queues and mutexes). There is also a δ framework base system library for including processor specific items such as bus arbiters, caches and I/O. Number of general purpose processors is also optional in the framework.

The configuration is performed when the user sets the configuration parameters in a GUI tool that generates the files of the components to be included in the system. The user can configure the δ framework in the options described in Table 2. The configuration focus mainly on the option to have certain parts of the RTOS implemented in hardware that are in-house developed solutions to RTOS issues. The main purpose the tool is to aid the system designer to explore which configuration is most suitable for a specific application requirement, e.g. regarding RTOS resource usage.

Further, a configurable hardware scheduler has been implemented by Mooney *et al.* [7]. The scheduler is configured similar to the δ framework with a GUI tool that after user configuration input generates the hardware files in Verilog and software driver files in C. The hardware scheduler which also includes RTOS clock-tick processing is configurable in number of tasks, external interrupts, timer resolution, and scheduling algorithm. The scheduler provides three scheduling disciplines: priority-based, rate monotonic and earliest deadline first. The scheduling mode can be changed at runtime. A software driver utilizes the hardware scheduler functionality through memory mapped registers of a general purpose CPU. Processor dependent code, such as the context switch routine, is also implemented in software. The functionality of the hardware scheduler is summarized in Table 3.

Table 3: Implemented functionality in the configurable hardware scheduler [7].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks	HW and SW	Number of (up to 64)
Scheduling algorithm	HW	Priority based, Rate monotonic or Earliest deadline first
RTOS clock-tick processing	HW	Resolution
External interrupts	HW and SW	Number of (up to 8)
Context switch routine	SW	

3.3 Co-Scheduler2

Morton *et al.* present the HW/SW partitioning of a single-processor real-time kernel in [19]. By strategic choice for speed-up purposes, only the scheduling including task

Table 4: Implemented functionality in the cs2 (Co-Scheduler2) [19].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks (periodic, aperiodic)	HW and SW	Number of (3 to 16)
Scheduling algorithm (Earliest deadline first)	HW	
Clock-tick processing	HW	
Context switch routine	SW	

queues, implemented as Earliest Deadline First (EDF) algorithm, is moved to a coprocessor, Cs2 (Co-Scheduler2). A general purpose CPU executes the application and tasks. The coprocessor is not claimed to be configurable but the coprocessor size and performance are analyzed for 3 to 16 tasks. The coprocessor grows linearly in size with number of tasks. An overview of the Cs2 is shown in Table 4.

3.4 Real-Time Task Manager (RTM)

Jacob *et al.* has implemented a real-time task manager (RTM) in hardware [6]. The RTM is a processor extension that implements scheduling, time management and event management with the purpose to minimize real-time operating system performance drawbacks. The RTM is an on-chip memory mapped peripheral. It cannot be used a stand-alone RTOS, it is meant to be integrated into software RTOS where it handles the scheduling. They claim the RTM not to application specific. The RTM is described as scalable and is reported used in configurations of 32, 64 and 256 tasks and events.

Table 5: Implemented functionality in the Real-time Task Manager (RTM) [6].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks	HW and SW	Number of (32, 64 or 256)
Scheduling algorithm (Priority based)	HW	
RTOS clock-tick processing	HW	
Context switch routine	SW	
IPC:		
Event management	HW	Number of (32, 64 or 256)

3.5 Operating System Coprocessor (OSC)

Oliviera *et al.* presents the Operating System Coprocessor (OSC) which is hardware VDHL implemented operating system functionality such as task scheduling, context switching, inter process communication and timing [24, 25].

The OSC is one of four dedicated co-processors in the Advanced Real-time Processor Architecture project (ARPA), a project with focus on investigating system-on-chip solutions optimized for real-time systems. They develop both main CPU and co-processors themselves. The close relation between the processors enables the OSC to have privileged access to registers and program counter of the main CPU which enables the OSC to perform a context switch without software intervention. The OSC supports either non real-time tasks or real-time tasks. Four real-time scheduling policies are provided as shown in the summary in Table 6. The OSC exchange task and semaphore information with the main CPU with memory mapped registers. The software driver is currently assembler implemented.

Table 6: Implemented functionality in the Operating System Coprocessor (OSC) [24, 25].

Functionality	HW/SW	Configurability
General RTOS functionality:		
Tasks	HW and SW	Number of
Scheduling algorithm	HW	Rate monotonic, Deadline monotonic, Earliest deadline first or Least slack first
RTOS clock-tick processing	HW	
Context switch routine	HW	
IPC:		
Semaphores	HW	Number of

3.6 The Silicon OS in the TRON project

The Real-time Operating System Nucleus project (TRON) is a Japanese project for research in ideal computer architectures in different areas. It has been running since 1984 and has produced several subprojects. One of the subprojects is the Industrial-TRON (ITRON) which is a software real-time OS for use in embedded systems. Nakano *et al.* presented in [21, 20] a real-time OS where parts of the ITRON RTOS functionality was implemented in hardware (HDL). The solution consists of a hardware part, called "Silicon TRON", and a software part, the remaining parts of the ITRON. The hardware part implements task scheduling, task synchronization, task communication and external interrupt management. The Silicon TRON together with the software part is called a "Silicon OS". Configurability is not described but different number of tasks, semaphores, flags and timers were reported as described in Table 7. The Silicon OS is connected to a general-purpose CPU as a peripheral with memory mapped registers. It is also connected to an interrupt input port of the CPU when a context switch is about to occur.

Table 7: Implemented functionality in the Silicon OS [21, 20].

Functionality	HW/SW	Configurability
General RTOS functionality:		
Tasks (periodic, aperiodic)	HW and SW	Number of (3 to 16)
Scheduling algorithm (Priority based)	HW	
External interrupts	HW	
RTOS clock-tick processing	HW	
Timers	HW	Bit width of timer argument (8, 16 or 32)
Context switch routine	SW	
IPC:		
Semaphores	HW	Number of (8, 16 or 32)
Flags	HW	Number of (8, 16 or 32)

3.7 F-Timer

Parisoto *et al.* presents a hardware architecture for real-time operating systems support using special hardware components implemented in one FPGA [26]. The included F-Timer is a co-processor that communicates with the main processor and releases it from 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 processor is interrupted and the correct task is available on the bus. Configurability is not discussed but the F-timer is said to be adjustable. Details of reported functionality is summarized in Table 8.

Table 8: Implemented functionality in the F-timer [26].

Functionality	**HW/SW**	**Configurability**
General RTOS functionality:		
Tasks (periodic, aperiodic)	HW and SW	
Priority levels	HW	
Scheduling algorithm (Smallest input first output, Priority based)	HW	
RTOS clock-tick processing	HW	
Timers	HW	
External interrupts	HW	
Context switch routine	SW	

4 Summary

In this report, research projects in the area of hardware support for RTOS have been described in the aspect of hardware/software partitioning of provided functionality. The combination of configurability and hardware support have been discussed. The combination of configurability and hardware support for RTOS may increase the ability to better exploit the benefits associated with hardware support such as increased performance and predictability, and decreased CPU load and memory usage.

Provided functionality among the hardware supported RTOS implementations have been summarized in Table 9 where hardware/software partitioning is shown for each feature, and configurability and enable/disable possibilities are marked. Regarding the *Features* in Table 9 the abbreviations and meaning will be explained. *Tasks* are in most cases marked being implemented in both hardware and software. This is because a task is considered implemented in software when there exist application tasks executed by a main CPU and when a context switch routine is implemented in software requiring a software implemented task control block (TCB), a structure containing task information regarding task entry point, stack, timing, priority and state information. A task is considered being a part of the hardware implementation when the scheduler is implemented in hardware, containing tasks and task information in task queues. Further features in Table 9 are *Sched.* (scheduling algorithm), *Clock* (internal RTOS clock-tick), *Irq* (external interrupts triggered by external events), *Csw* (context switch routine), *Sem* (any kind of semaphore functionality), Flags (any kind of flag functionality), *Events* (event groups or event management in the area of *IPC*). *Mailbox*, *Queues* and *Mutex* are not abbreviations and are common RTOS features for IPC. The abbreviations of the hardware support in the table are each presented in previous sections of this report.

Table 9: A summary of provided functionality in different hardware support implementations. The table shows if a feature is implemented in both hardware and software (H/S), only hardware (HW) or only software (SW). Configurable features are marked (†).

	Hardware Support							
Feature	RTU	δ	Sch	Co2	RTM	OSC	TRON	F-timer
General RTOS functionality:								
Tasks	H/S†	SW†	H/S†	H/S†	H/S†	H/S†	H/S†	H/S
Sched.	HW	SW	HW†	HW	HW	HW†	HW	HW
Clock	HW†	SW	HW†	HW	HW	HW	HW†	HW
Irq	HW†	n/a	H/S†	n/a	n/a	n/a	HW	HW
Csw	SW	SW	SW	SW	SW	HW	SW	SW
IPC:								
Sem.	HW†	SW‡	n/a	n/a	n/a	HW†	HW†	n/a
Flags	HW†	n/a	n/a	n/a	n/a	n/a	HW†	n/a
Events	n/a	SW‡	n/a	n/a	HW†	n/a	n/a	n/a
Mailbox	n/a	SW‡	n/a	n/a	n/a	n/a	n/a	n/a
Queues	n/a	SW‡	n/a	n/a	n/a	n/a	n/a	n/a
Mutex	n/a	SW‡	n/a	n/a	n/a	n/a	n/a	n/a
Specific features:								
DDU[1]	n/a	H/S‡	n/a	n/a	n/a	n/a	n/a	n/a
DMMU[2]	n/a	H/S‡	n/a	n/a	n/a	n/a	n/a	n/a
LC[3]	n/a	HW‡	n/a	n/a	n/a	n/a	n/a	n/a

† Configurable

‡ Enable/Disable

[1] Deadlock detection unit.

[2] Dynamic memory management unit.

[3] System-on-chip lock cache.

Apparent in Table 9 is that the δ Framework (δ in Table 9) is different from the other hardware support. Instead of implementing hardware support of RTOS functionality closely related to the real-time kernel functionality, the δ Framework project proposes special purpose hardware components providing solutions to known RTOS issues such as deadlock detection and dynamic memory management. The RTOS provided in the framework is software implemented. This project is the only project where options to enable or disable functionality is provided clearly.

Even though the hardware support included in this report are implemented differently and more or less independent of each other, there are several similarities. All but one hardware support is connected to a general purpose CPU as a memory mapped peripheral. When a general purpose CPU is used, the system designer must manipulate the CPU registers with software programming and hence the context switch routine (*Csw* in Table 9), is implemented in software. This means that the TCBs that are stored and re-stored during a context switch, is implemented in software as well. The advantage with this solution is that the hardware support can be ported to another general purpose CPU. However, in the project behind Operating System Coprocessor (*OSC* in Table 9), they develop both the main CPU and the OSC themselves meaning they have access to CPU registers in hardware and can perform a context switch without software intervention. This increases performance of the context switch. In early implementations of the RTU, in the FASTCHART project, a similar solution was presented. The hardware real-time kernel was integrated with an in-house developed CPU and context switch was possible to perform in only one clock-cycle.

Besides being connected to a general purpose CPU, other similarities among presented hardware support are that, except for the δ Framework, they all have the scheduler and RTOS clock-tick processing implemented in hardware. Implementing the RTOS clock-tick in hardware relieves the main CPU from handling timing calculation for periodic tasks and delayed tasks. In software implemented RTOS, the RTOS has to check the task delay queues, decrease each task's timer and re-scheduled a task if a timer has expired. This procedure is performed in certain time intervals and each time the main CPU is interrupted and has to execute software code for handling this. In a hardware implemented scheduler, this is performed in parallel to the CPU executing the running application task, which leads to increased performance.

Regarding scheduling, the configurable hardware scheduler and the Operating System Coprocessor (*Sch* and *OSC* in Table 9) are providing configurability regarding scheduling while the system designer has several options regarding scheduling algorithms in hardware. The configurable hardware scheduler is re-configurable at run time while the OSC provides the designer to choose scheduling algorithm at compile time. The other hardware support only provide one scheduling algorithm.

Half of the hardware support in Table 9 provide some form of IPC, and when they do, provided IPC is configurable. Configurability in general is discussed to most extent in the hardware support projects that provide complete tools for generating whole systems: the δ Framework, OSC and to some extent the configurable hardware scheduler. Here, configurability is accomplished when setting component parameters in connection with using the tool. Among the other hardware support that does not provide a tool environment, the Silicon OS in the TRON project and the RTU (*TRON* and *RTU* in Table 9), reports configurability regarding tasks, timer resolution, semaphores and flags. The variations in configurability is different but provided configurable functionality is very similar in these two hardware support.

References

[1] J. Bhasker. *A VHDL Primer*. Prentice Hall PTR, Revised edition, 1995.

[2] L. Enblom and L. Lindh. Adding Flexibility and Real-Time Performance by Adapting a Single Processor Industrial Application to a Multiprocessor Platform. In *Parallel and Distributed Processing EUROMICRO Workshop*, Mantova, Italy, February 2001.

[3] F. Engel, G. Heiser, I. KuZ, S. M. Petters, and S. Ruocco. Operating Systems on SoCs: A Good Idea? In *ERTSI in conjunction with 25th IEEE RTSS*, Lisbon, Portugal, December 2004.

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

[5] T. Klevin and L. Lindh. Scalable Architecture for Real-Time Applications And Use of bus-monitoring. In *International Conference on Real-Time Computing Systems and Applications*, December 1999.

[6] P. Kohout, B. Ganesh, and B. Jacob. Hardware Support for Real-time Operating Systems. In *IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Newport Beach, USA, 2003.

[7] P. Kuacharoen, M. A. Shalan, and V. J. Mooney III. A Configurable Hardware Scheduler for Real-Time Systems. In *International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, USA, June 2003.

[8] J. J. Labrosse. *MicroC/OS-II The Real-Time Kernel*. CMP Books, second edition, 2002.

[9] J. Lee, V. J. Mooney III, K. Ingström, A. Daleby, T. Klevin, and L. Lindh. Comparison of the RTU Hardware RTOS with a Hardware/Software RTOS. In *Design Automation Conference*, January 2003.

[10] J. Lee, K. Ryu, and V. J. Mooney III. A Framework for Automatic Generation of Configuration Files for a Custom Hardware/Software RTOS. In *International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 2002.

[11] L. Lindh. FASTHARD - A Fast Time Deterministic Hardware Based Real-Time Kernel. In *IEEE press, Real-Time Workshop*, Athens, January 1992.

[12] L. Lindh. *Utilization of Hardware Parallelism in Realizing Real Time Kernels*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1994.

[13] L. Lindh, T. Klevin, and J. Furunäs. Scalable Architecture for Real-Time Applications - SARA. In *Swedish National Real-Time Conference (SNART)*, Linköping, Sweden, August 1999.

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

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

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

[17] V. J. Mooney III. Hardware/Software Partitioning of Operating Systems. In *Design Automation and Test in Europe Conference*, March 2003.

[18] V. J. Mooney III and D. M. Blough. A Hardware-Software Real-Time Operating System Framework for SoCs. *IEEE Design and Test of Computers*, 19:44–51, 2002.

[19] A. Morton and W. M. Loucks. A Hardware/Software Kernel for System on Chip Designs. In *ACM Symposium on Applied Computing*, Nicosia, Cyprus, 2004.

[20] T. Nakano, Y. Komatsudaira, A. Shiomi, and M. Imai. VLSI Implementation of a Real-time Operating System. In *Design Automation Conference (ASP-DAC)*, Chiba, Japan, January 1997.

[21] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai. Hardware Implementation of a Real-time Operating System. In *TRON Project International Symposium*, Tokyo, Japan, November 1995.

[22] S. Nordström and L. Asplund. Configurable Hardware/Software Support for Single Processor Real-Time Kernels. In *International Symposium on System-On-Chip Conference*, Tampere, Finland, November 2007.

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

[24] A. Oliviera, V. Sklyarov, and A. Ferrari. ARPA - An Open Source System-on-Chip for Real-Time Applications. In *Embedded Real-Time Systems Implementation Workshop*, Lisbon, Portugal, December 2004.

[25] A. Oliviera, V. Sklyarov, and A. Ferrari. ARPA - An Technology Independent and Synthetizable System-on-Chip Model for Real-Time Applications. In *Digital System Design in Euromicro Conference*, Porto, Portugal, August 2005.

[26] A. Parisoto, A. J. Souza, L. Carro, M. Pontremoli, C. Pereira, and A. Suzim. F-Timer: dedicated FPGA to real-time systems design support. In *Real-Time Systems, 9th Euromicro Workshop*, 1997.

[27] Prevas AB, Västerås, Sweden, www.prevas.se, 2007.

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

[29] J. Stärner, J. Adomat, J. Furunäs, and L. Lindh. Real-Time Scheduling Co-Processor in Hardware for Single and Mulitprocessor System. In *EUROMICRO Conference*, Prague, Czech Republic, September 1996.