Latin American Applied Research 37:35-40 (2007)

uRT51: AN EMBEDDED REAL-TIME PROCESSOR IMPLEMENTED ON FPGA DEVICES

R. CAYSSIALS[†], M. DUVAL[‡], E. FERRO[†] and O. ALIMENTI[†]

[†]*Dpto. Ing. Elect., Univ. Nac. del Sur –CONICET, 8000 Bahía Blanca, Argentina*
{iecayss, ieferro, iealimen}@criba.edu.ar
[‡] *Dpto. Ing. Elect., Univ. Nac. del Sur –CIC, 8000 Bahía Blanca, Argentina*
duval@criba.edu.ar

***Abstract*— In this paper we describe and evaluate the main features of the uRT51 processor. The uRT51 processor was designed for embedded real-time control applications. It is a processor architecture that incorporates the specific functions of a real-time system in hardware. It was described using synthesizable VHDL and it was implemented on FPGA devices. We describe how the uRT51 processor supports time, events, task and priorities. The performance of the uRT51 processor is evaluated using a control application as a case study. The experiments show that the uRT51 processor scheduling features outperform the ones obtained using a traditional RTOS-based real-time system.**

***Keywords*— Scheduling, Real-Time, Embedded Processor**

I. INTRODUCTION

Real-time systems are ubiquitous and diverse, and include mobile phones, entertainment devices, automobiles, toys, smartcards, medical devices, network switching equipment, sensors and industrial robots. They are modeled as a set of tasks that has to be executed by a processor before a certain deadline.

Embedded real-time systems are usually built using either Real-Time Operating Systems (RTOS) or designing the whole system as a monolithic program based on specific hardware resources. Real-time operating systems allow designer to achieve a higher abstraction level, higher portability and better verification and maintenance features of the system. Several RTOSs were proposed and designed in order to give real-time support to applications (e.g. POSIX.1003 (IEEE1003. 1d, 1999), Spring kernel (Stankovic and Krithi, 1991), QNX (Hildebrand, 1992)). The RTOS includes a special system task, named *the scheduler*, which shares the processor among the tasks that require execution. The scheduler implements a priority discipline to determine the next task to be executed. The scheduler should be executed periodically and its execution produces overhead on the system.

Some applications may require stricter temporal features or may not tolerate the runtime overhead produced by the operating system. In these cases, a direct use of the hardware resources is mandatory. Timers, counters and interrupts have to be directly programmed avoiding an easy maintenance of the system. In addition, programming directly the hardware resources of the system may generate failures hard to debug.

On the other hand, several approaches have been proposed to implement processors with real-time features. In Halang and Colnaric (1993), Halang *et al.* (1995) and Glavinic *et al.* (1999), a co-processor architecture is proposed to support real-time applications. In Adomat *et al.* (1996), a real-time kernel is implemented in hardware. In Kohout *et al.* (2003), a Real-Time Task Manager (RTM) is proposed as a parallel unit to carry out the real-time behaviour of the processor. These architectures are very restrictive on both the priority discipline implemented and the number of tasks supported.

In Matthew *et al.* (1999), it is described the AAMP processor developed by Rockwell-Collins to be used in critical avionics applications. The AAMP was verified to have a predictable behaviour, but it does not have special real-time features but a multitasking support. None of these approaches are available for commercial, industrial or educational developments.

The uRT51 is a processor for embedded real-time applications. Its architecture was designed from a real-time point of view and it may support a great deal of priority disciplines over a set of up to 65,000 tasks. The uRT51 architecture is scalable and consequently the number of tasks and events supported depends on the amount of physical memory of the system. It is described in VHDL and it can be used to build system-on-chip (SoC) architectures over FPGA devices. With the uRT51 Real-Time Suite, temporal parameters can be defined independently of the functionality of each task. Hence, task's code does not include real-time programming which permits a higher abstraction level of implementation and a better flexibility in the design (uRT51, 2005).

In this paper, we describe the main features of the uRT51 processor. We evaluate the performance of the uRT51 processor in a control application. We propose a speed control of a DC motor to measure the performance of the uRT51 processor. The experiments show that the uRT51 is much more efficient than an RTOS-based embedded real-time system.

This work was supported by the Technological Modernization Program under Grant BID1201/OC-AR

The paper is organised as follows: Section 2 describes the implementation of a traditional RTOS. Section 3 describes the main features of the uRT51 architecture. The uRT51 Real-Time Suite facilities are sketched in Section 4. In Section 5, the speed control of a DC motor is proposed as a case study. Results are analysed in Section 6. Conclusions are drawn in Section 7.

II. REAL-TIME PROCESS MODEL AND RTOS SCHEDULING

A real-time system is modelled as a set $\mathbf{\Pi}$ of n periodic tasks to be executed on a processor. Each task i performs a certain function and its real-time behaviour is characterised by its period, T_i, its deadline, D_i, and its execution time, C_i.

Tasks must be invoked periodically according to its period. Most of RTOSs run a time-based task invoked by a periodic timer interrupt that goes off at regular intervals and checks whether a real-time task has to be invoked or not (Gallmeister, 1995). Because the timer task is invoked at discrete time intervals, denoted T_{timer}, the time in a real-time system is considered slotted and either tasks' periods and tasks' deadlines should be expressed in slots. Therefore, task i will have to be invoked either every $\lfloor T_i/T_{timer} \rfloor$ or every $\lceil T_i/T_{timer} \rceil$ *slots*, and T_{timer} defines the precision with which the real-time parameters can be expressed.

For instance, if the timer task is set to go off every T_{timer} = 500μs, a real-time task with a period of T = 1.6ms, should be invoked either every 3 slots (1.5 ms) or 4 slots (2 ms). In both cases there exist an error in the task invocation that may force to modify the task's code.

T_{timer} has influence either on the time resolution of the real-time system as well as the overhead that the RTOS produces.

III. THE uRT51 ARCHITECTURE

In this section we describe the main components of the uRT51 architecture. A full description of the uRT51 features can be found in uRT51 (2005).

Figure 1 shows a layout of a system based on an uRT51 processor consisting of:

- The 8051 core: it implements a subset of the instruction set of the 8051 processor and it can recognise some special real-time instructions to configure the Real-Time Manager. Input/Output devices can be driven directly by the 8051 core.
- The Real-Time Manager: it controls the real-time behaviour of the uRT51 processor. It continuously checks if there exists a real-time action to be performed. When no task is requiring to be executed, the real-time manager halts all the uRT51 activities and consequently it reduces the power consumption of the system.
- The Debugging Unit: it allows an easy integration with the uRT51 Programming Suite. All the activities of the uRT51 can be controlled and supervised by the uRT51 Programming Suite through the Debugging Unit.
- The Memory Controller: it implements all the control functionality to connect the uRT51 processor to external memories. It supports different bus widths.
- The Interrupt Manager: it gives support to asynchronous real-time events that can release real-time tasks.

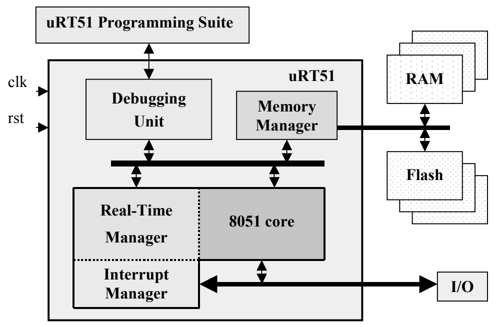

Figure 1. uRT51's architecture layout.

The RAM memory is shared between the 8051 core and the Real-Time Manager. Sharing is synchronized by the Memory Controller to avoid waiting states. The Real-Time Manager stores all the real-time information that it requires to execute the real-time tasks into the RAM of the system. This information includes release time, deadline, priority and state of each real-time task. The Real-Time Manager checks if there exists an event that forces to change the state of the 8051 core.

The Real-Time Manager is not a processor unit but a digital circuit optimised to carry out the real-time functions of the uRT51 processor. The architecture was designed to work with a 10MHz external clock. With this frequency, both tasks' period and tasks' deadline can be expressed with a time resolution of 100ns. The 8051 core does not execute any timer subroutine and consequently there is not RTOS overhead. Hence, both the schedulability of the system and the response time of the real-time tasks are improved.

A. Time and Events in the uRT51

A real-time processor should support time adequately because it is when events take place. We define *event* as an occurrence or happening, usually significant to the performance of a function, operation, or task. Consequently, the evolution of a real-time system may be adequately defined by expressing the time when events happen. Release, deadline, end of execution, priority promotion, abortion are some examples of events in a real-time system.

The uRT51 processor adequately supports time and events. It keeps the time of the system into internal registers of the Real-Time Manager. The Real-Time Manager continually checks if there exists an event that requires to be activated or a task whose state should be modified. The performance handling real-time information is high and predictable because events and tasks are handled separately from the execution of the real-time tasks. Moreover, jitters and real-time performance of the uRT51 processor can be measured in periods of the system's clock instead of the interval of time defined by the scheduler's period (T_{timer}), as it is the case in RTOS-based real-time systems. Therefore, the performance is improved in several orders of magnitude. The maximum number of events supported depends on the physical amount of memory of the Real-Time Memory assigned to events structures.

B. Tasks and Priorities

Because of most of real-time process models consider a set of tasks to be executed into a processor, a real-time architecture should support multitasking. Consequently, a real-time architecture has to manage all the task's parameters needed to provide a multitasking environment. The uRT51 processor utilises *task structures* to store the parameters of the real-time tasks. The task structures are stored into the RAM memory of the system. The maximum number of tasks supported depends on the physical amount of memory of the Real-Time Memory assigned to task structures.

In a multitasking system, a priority is assigned to each task in order to schedule the set of tasks that are ready to be executed. The priority of each task may be modified as well as remained fixed during runtime according to the scheduling discipline implemented. Previous real-time processor approaches implemented a predefined scheduling discipline over a reduce set of priorities (Halang and Colnaric, 1993; Halang *et al.*, 1995; Glaviníc *et al.*, 1999; Adomat *et al.*, 1996; Matthew *et al.*, 1999). The uRT51 processor does not execute a predetermined priority discipline but chooses for execution the highest priority task instead. The priority of each task is held in the respective task structure and it can be modified during runtime according to the real-time configuration performed.

The scheduling discipline depends on how the priority of each task is assigned. Task's priority may be changed configuring the action taken on timed events or interrupts. On the contrary, if actions do not modify the tasks' priority, then a fixed priority discipline is implemented. Therefore, a great deal of priority disciplines may be implemented.

C. Real-Time Configuration

The Real-Time Manager should be configured in order to set the real-time parameters of the real-time application. The configuration is carried out by the execution of special real-time instructions. The set of real-time instructions expands the original instruction set of the 8051 core. When the Real-Time Manager is configured, it is rarely necessary to execute additional real-time instructions. However, real-time tasks can execute real-time instructions during runtime.

Once the Real-Time Manager is configured it carries out all the real-time functions at the same time that the 8051core executes the code of the real-time tasks.

Some of the real-time instructions are: Set Period Task *i* in *r* ns, Enable Event *i* as Release of Task *j,* Change State of Task *i* to Ready.

The full set of real-time instruction can be found in uRT51 (2005). However, the uRT51 Programming Suite shows to the user an easy interface to configure the uRT51 processor without any knowledge of the real-time instructions.

IV. THE URT51 REAL-TIME SUITE

The uRT51 Real-Time Suite is a full featured, high-performance, interactive and easy to use tool collection that supports programming, debugging and analysis of real-time systems implemented on uRT51 processors. It includes all the facilities for runtime analysis as well as support for data logging. We used the uRT51 Real-Time Suite to implement the case study of this paper.

The uRT51 Real-Time Suite describes a real-time system in terms of tasks, real-time properties and priority discipline. You may include as many tasks as you like within the memory constraints of your uRT51 processor system (uRT51, 2005).

The main object of a real-time system in the uRT51 Real-Time Suite is a real-time task. A task carries out a certain function of the application. A task is defined by:

- its code: it is programmed according to the function that the task must perform, and
- its real-time properties: they set the temporal properties of the task such as periodical invocation, priority, deadline, etc. Some of the real-time properties depend on the priority discipline selected.

The task code defines the way that the task performs its function and the real-time properties establish the runtime behaviour of the task. The uRT51 Real-Time Suite includes a Task Editor for a concurrent programming of both the functionality and the real-time features of the real-time application.

V. EXPERIMENTAL SETUP

We implemented the uRT51 processor system on an Excalibur FPGA development board based on an APEX EP20KE200 device from Altera. The uRT51 architecture was described using VHDL and synthesized with Leonardo Spectrum from Exemplar. The memory required for the uRT51 processor was implemented using the EAB included in the APEX architecture. The whole architecture requires 4500 LEs approximately.

A scheduler for four tasks was implemented to be executed on the 8051 core of the uRT51 to get a fair comparison. It was implemented as a timer routine and it does not execute any real-time instruction of the Real-Time Manager unit of the uRT51. When a real-time system based on a RTOS is considered, the timer interval should be defined and tasks' period expressed in slots. Timer intervals were selected in order to get tasks periods between 2 slots and 255 slots (otherwise a 16-

bit arithmetic should be implemented on a 8-bit processor).

Experiences were performed considering system clock frequencies equal to 10, 20, 40, 50, 80 and 100MHz. The uRT51 Programming Suite was used to program the system and to analyse its runtime behaviour.

A. Case Study: DC motor speed control.

In this section we describe the speed control of a DC motor that we used to evaluate the performance of the uRT51 processor.

The speed of the DC motor is controlled varying the energy transferred to the motor using a pulse width modulation (PWM) technique. The speed is measured indirectly using an optical sensor and a slotted disc attached to the shaft of the motor (Fig.2). The speed of the motor is then measured counting the number of toggles occurred in a certain interval I_s.

We can make the system more predictable by implementing the sensor task as a periodic real-time task instead of using special counters in interrupts routines (Mackall, 1988). In this way, we can be sure that we can include this system as part of a larger application and, if the system is schedulable, then the control application will work properly. On the other way, we should assure that the interrupt routines do not interfere with the other real-time tasks of the system.

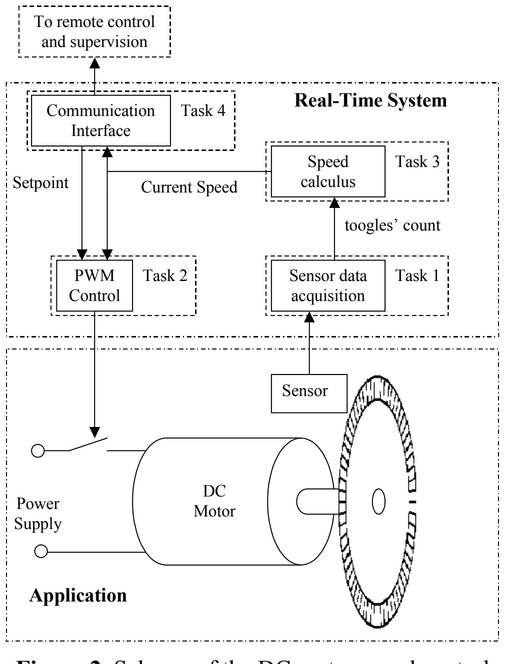

Figure 2. Scheme of the DC motor speed control

Four real-time tasks can be defined from the control functions that the system must perform (Fig.2):

- **Task 1**: Data Acquisition. This task is in charge of monitoring the sensor and it counts the number of toggles that the sensor produces on its output.
- **Task 2**: PWM control. It modifies the duty cycle of the PWM wave according to the difference between the setpoint and the speed of the motor.
- **Task 3**: Speed calculus. This task calculates the speed of the motor through the angle elapsed in the interval I_s.
- **Task 4**: Communication interface. This task sends to and receives from a host control and supervision data, through an asynchronous serial link working at 9600 bps.

B. Implementation and Evaluation

Whilst periods and deadlines depend on the functions that the control tasks must perform, the execution time depends on the computational characteristics of the processor. The four real-time tasks were implemented to be executed on the 8051 core of the uRT51 processor. The worst-case execution time of each task is given in system clock period units on the Table 1.

Table 1. Worst case execution time of the tasks of the system expressed in clock periods

Task	C [clock periods]
1	401
2	436
3	506
4	366

The priority assignment is done according to a rate monotonic policy (RM). In RM, tasks with shorter periods are assigned to higher priorities. Table 2 shows the period, the deadline and the priority of each one of the real-time tasks. Tasks' period were chosen according to the control properties of the application. Higher priorities are represented by lower priority indexes.

Table 2. Period, deadline and priority of the real-time tasks

Task	T	D	Priority
1	347.22μs	347.22μs	0 (highest)
2	1ms	1ms	1
3	23ms	23ms	3
4	1ms	1ms	2

We compare both systems measuring the jitter of each one of them. The jitter is the maximum difference between the exact intervals of two consecutive invocations of the same task. When control tasks are affected by jitter, then undesirable output frequencies are applied to the control application. These undesirable frequencies are transmitted to the application through the actuators and may produce unwanted effects such as: vibration, heat, instability, noise and so on. Figures 3, 4, 5 and 6 show the average jitter of task 1, 2, 3 and 4, respectively.

From the experiences, we got that the uRT51 processor could schedule the system with a 10MHz clock frequency whilst the RTOS-based system needed at least a 50MHz clock frequency to schedule it. Moreover, the control performance of the uRT51 processor with a 10MHz clock could not be improved using a RTOS with a 100MHz clock.

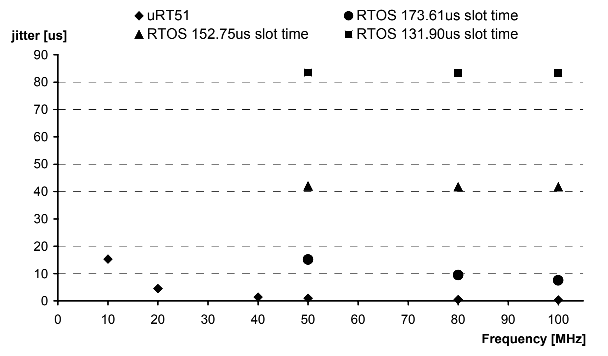

Figure 3. Average Jitter of task 1

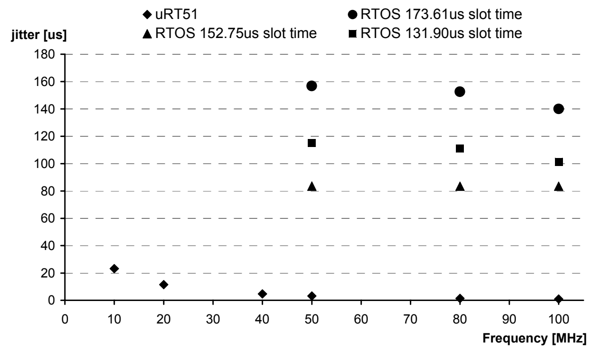

Figure 4. Average Jitter of task 2

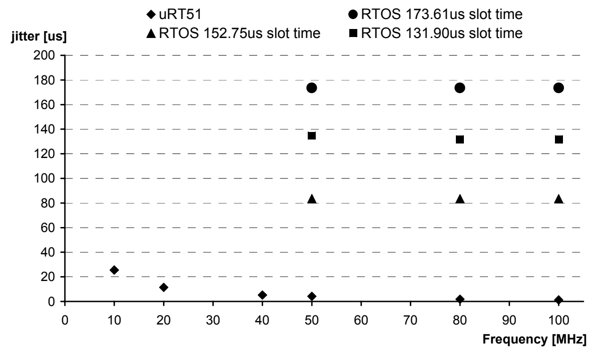

Figure 5. Average Jitter of task 3

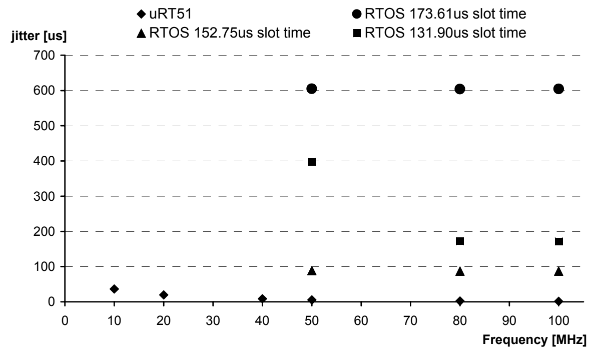

Figure 6. Average Jitter of task 4

Figures show that the performance of a RTOS-based system is very sensitive to the slot time chosen to execute the scheduler. This makes the design complicated when the system is built from a set of real-time subsystems.

On the other hand, the uRT51 processor does not introduce any slot time and consequently its performance depends on just the frequency of the system clock.

VI. DISCUSSION

From the real-time scheduling theory point, the overhead of the system is reduced in an uRT51-based implementation and the system is schedulable at lower clock frequencies with the following advantages:

- The power consumption of the system is reduced. Because the system can be scheduled at a lower clock frequency, the voltage and consequently the power consumption of the system may be dramatically reduced. Moreover, the uRT51 avoids the overhead introduced by the RTOS and as result of it the power consumption is also reduced.
- Lower cost technology can be used. When lower clock frequencies are applied, cheaper technologies can be used in the processor implementation, debugging and interfacing. Hence, a more sophisticated device should be needed if a system based on RTOS is implemented.

On the other hand, from the application point of view, the performance is improved when an uRT51 processor is utilised because the average jitter of the tasks is much less. This feature may be very important when the real-time system is applied to control applications.

In a real-time system based on a RTOS, the slot time may have a greater influence on the control performance than the frequency of the system clock. Increasing the clock frequency in a RTOS-based system may not decrease proportionally the jitter introduced. It could not be an optimum slot time for all tasks: in our case study, 173.61μs is optimal for task 1, whilst a slot time of 152,75μs is optimal for tasks 2, 3 and 4. On the other hand, we can observe that because the Real-Time Manager is a digital circuit (not a processor unit), the jitter in an uRT51 processor is just inversely proportional to the clock frequency.

VII. CONCLUSIONS

In this paper we described the main features of the uRT51 processor and we introduced the uRT51 Real-Time Suite that allows to program, to debug and to analyze real-time applications implemented on the uRT51 processor. We implemented the speed control of a DC motor to evaluate experimentally the performance of the uRT51 processor in a control application. All the tools and programs used in this paper are available through authors or uRT51 (2005).

The efficiency of the uRT51 processor allows scheduling the real-time system at a very low frequency. Whilst the uRT51 processor scheduled the DC motor speed control application with a 10 MHz clock, the RTOS-based system needed at least a 50Mhz clock.

The control properties of the real-time system are improved because the jitter of the control tasks is reduced. Consequently, the perturbations that the real-time system produces on the control application are reduced as well. A higher precision can be achieved on the temporal features of the system.

The design of a real-time application with the uRT51 processor can be done in a higher abstraction level because the functionality of each task can be defined in a concurrent way with its real-time features. Hence, the temporal characteristics of each task can be modified without reprogramming its code, as it is the case when a RTOS is utilised.

We can conclude that the uRT51 processor architecture is suitable to be applied to low-cost, low-power embedded real-time control applications.

REFERENCES

Adomat, J., J. Furuns, L. Lindh and J. Strner, “Real-Time Kernel in Hardware RTU: A Step Towards De-terministic and High-Performance Real-Time Systems,” *Proc. 8th Euromicro Workshop on Real Time Systems*, L'Aquila, Italy, 164-168 (1996).

Gallmeister, B. O., *POSIX.4: Programming for the Real World*, O’Reilly & Associates, CA (1995)

Glaviníc, V., S. Gros and M. Colnaric, “Vhdl-based modeling of a hard real-time task processor,” *Proc. 1999 IEEE International Symposium on Industrial Electronics*, Bled, Slovenia, 49-54 (1999).

Halang, W., M. Colnaric and D. Verber, "Supporting high integrity and behavioural predictability of hard real-time systems." *Informatica, Special Issue on Parallel and Distributed Real-Time Systems*, **19**, 59–69 (1995).

Halang, W.A. and M. Colnaric, "Architectural support for predictability in hard real time systems." *Control Engineering Practice*, **1**, 281–285 (1993).

Hildebrand, D., “An Architectural Overview of QNX,” *Proc. 1992 Usenix Workshop on Micro-Kernels & Other Kernel Architectures*, Seattle, USA, 113-126 (1992).

IEEE1003.1d. *Draft Information Technology - Portable Operating System Interface (POSIX): Part 1: System Application program interface; Addi-tional Real-Time Extensions*, IEEE Standards (1999)

Kohout, P., B. Ganesh and B. Jacob, “Hardware support for real-time operating systems,” *Proc. First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Newport Beach, USA, 45-51 (2003).

Mackall, D. A., “Development and flight test experiences with a flight-crucial digital control system”, *Internal Report, Technical Paper 2857*, NASA AMES Research Center, NASA (1988).

Matthew, M., S. Wilding, P. Miller, D. A. Greve and M. Srivas., “Formal verification of the AAMP-FV microcode,” *Internal Report MD21076-1320*, NASA (1999)

Stankovic, J.A. and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software*, **8**, 62-72 (1991).

uRT51, The uRT51 Site, http://www.uRT51.com.ar. (2005)

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.