International Review on *Computers and Software* *(IRECOS)*

Contents

(continued)

PWP Praise Worthy Prize

International Review on Computers and Software (I.RE.CO.S.), Vol. 8, N. 8
ISSN 1828-6003 *August 2013*

Real-Time Scheduling Architecture for Embedded Systems

Ricardo Cayssials[1, 2, 3], Edgardo Ferro[1], José M. Urriza[4], Eduardo Boemo[5]

Abstract – *Industrial applications require meeting real-time specifications. Real-time systems are implemented using processors in order to execute real-time tasks. Temporal constraints must be supported by real-time operating systems or designing the application based on specific hardware resources. Previous approaches to real-time processors have implemented operating system functions in hardware and consequently they are designed to manage tasks' periods rather than real-time. They cannot be used in a great deal of applications because they are based on restrictive models. This paper proposes the Hardware Real-Time Scheduling Architecture (HRTSA) that introduces an innovative methodology with which to efficiently manage time, events, priorities and tasks in an embedded hardware implementation. The HRTSA is described and real-time performance is evaluated.* ***Copyright © 2013 Praise Worthy Prize S.r.l. - All rights reserved.***

***Keywords**: Real-Time Scheduling, FPGA, Soft-Processor, Microprocessor Design*

I. Introduction

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. RTOSs allow designers to achieve a higher abstraction level, higher portability and finally, 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 [1], Spring kernel [2], QNX [3], uC/OS II [4], [5], among others).

The RTOS includes a special system task, known as the scheduler, which shares the processor among the tasks which require execution. The scheduler carries out a scheduling policy to determine the next task to be executed. The real-time behaviour of the system depends on the scheduling policy implemented.

Each RTOS applies a scheduling policy and it is difficult to change it in order to modify the real-time characteristics of the application. The scheduler should be executed periodically producing an overhead on the system. Some applications may require stricter temporal features or may not tolerate the runtime overhead produced by the RTOS or the system memory requirements. In these cases, a direct use of the hardware resources is mandatory. Timers, counters and interrupts must be directly programmed making easy maintenance of the system difficult.

The improvement of the computational capacity has allowed the utilisation of microprocessor-based systems within a great deal of equipment and many embedded real-world applications, such as telecommunications, transportation, industrial automation, surveillance, and so on. These classes of applications have real-time operation constraints which must be fulfilled to avoid serious human injuries or material damages [6].

Several papers have proposed the implementation of scheduler functions in hardware in order to improve the real-time efficiency of the system ([6], [7], [8], [9], [10], [11], [12], [13], [14], [15]). Most of the ideas are based on transferring the scheduling functions of the RTOS to hardware. However, while the runtime efficiency increases ([16]), the real-time restrictions originating from the software version implemented in RTOSs remain in the hardware. Hardware schedulers which have migrated from RTOS usually implement a certain scheduling policy over a reduced number of tasks which cannot be easily modified.

Nowadays, Field Programmable Gate Arrays (FPGAs) offer a high capacity to implement Systems-on-Programmable-Chip (SOPC). The architecture of the processors implemented in FPGAs, known as soft-processors, may be modified in order to add new features to the processor prior to its synthesis and implementation. This may include new instructions, often called custom instructions, and inter-processor interfaces. These features make SOPCs and FPGA devices suitable for industrial and embedded applications ([17], [18], [19], [20], [21], [22]) since they allow a high flexibility for Hardware/Software Co-design.

In this paper, an architecture referred to as Hardware Real-Time Scheduling Architecture (HRTSA), and which is able to incorporate real-time scheduling properties into a soft-processor, is proposed. The HRTSA was designed from a real-time point of view and may support any real-time scheduling mechanism over a set of up to 65,000 tasks.

The HRTSA 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 can be used to build SOPC over FPGA devices from different vendors.

Manuscript received and revised July 2013, accepted August 2013

The architecture may be adapted to different soft-processors (e.g. Altera NIOS II, Xilinx Microblaze, 8051 core). With the HRTSA, temporal parameters can be defined independently of the functionality of each task. Hence, a task’s code may not include real-time programming which permits a higher abstraction level of implementation and a better flexibility in design.

The paper is organised as follows: Section 2 describes the typical model used in real-time theory. The HRTSA is explained in Section 3. Section 4 describes how time and events are supported by the HRTSA. The support for tasks and priorities are explained in Section 5. Section 6 details the architecture of the Real-Time Manager Unit of the HRTSA. An example of configuration of the HRTSA is explained in Section 7. The performance of the HRTSA is compared with a traditional scheduler in Section 8. Finally, results are analysed in Section 9 whilst conclusions are drawn in Section 10.

II. Real-Time Model

A real-time system is modelled as a set Π of n periodic tasks to be executed on a processor. Each task i performs a certain function and it is characterised by its period or minimum time between invocations, T_i, its deadline, D_i, and its execution time, C_i. The *utilisation factor* of each task i is defined as C_i/T_i and the *total utilisation factor* of the system is the summation of the utilisation factor of all tasks.

If each real-time task of the system always meets its deadline, then it is said that the system is *schedulable*. Real-time advance techniques compute the worst case of response of any real-time task of the system in order to analyse its schedulability ([23], [24], [25]).

Scheduler assigns the processor according to the scheduling policy which it implements. Fixed Priority (FP) and Earliest Deadline First (EDF) are two of the most important scheduling policies in real-time ([23], [26]). In an FP policy, each real-time task is assigned with a priority in the design time and it remains fixed during runtime. In an EDF policy, the priority of a task increases as its deadline draws closer. The EDF policy is optimal in the sense that if the system is not schedulable under an EDF policy, then the system is not schedulable under any other scheduling policy. However, the complexity of the EDF policy may make it difficult to achieve an efficient implementation, compared with a Fixed Priority one, in an RTOS for embedded applications.

The time that a task has to wait to be executed/completed depends on the computation time required by higher priority tasks that are ready to be executed. Because the pattern of releases is not fixed, and execution times of a task may vary from invocation to invocation, which results in a variable interference and therefore a variable response time of the task. These time variations produce a jitter on both the starting and completion times of task execution. Despite the fact that the jitter is not usually taken as a real-time parameter, it has been proven that it could produce undesirable effects on control and signal processing applications ([27]). In this paper, the jitter is used to compare the real-time performance of the different scheduling mechanisms evaluated. In the following section, the proposed Hardware Real-Time Scheduling Architecture is described. It is designed considering the real-time model and consequently it supports the different scheduling mechanism proposed in the real-time literature.

III. The Hardware Real-Time Scheduling Architecture

The basis of the HRTSA relies on the control of the real-time behaviour of the system without hindering the execution of the system’s tasks. The architecture includes two main processing units (Fig. 1): the Processing Unit (PU) and the HRTSA unit.

The processing unit (PU) is a soft-processor core which fetches the instructions of the tasks from the memory and executes it accordingly. In this paper, the Intel 8051 instruction set was chosen as a case study to validate ideas. However, the HRTSA may be adapted to others soft-processors (e.g NIOS II, MicroBlaze).

The instructions set of the Intel 8051 processor is recognised in addition to a number of special real-time instructions which are included to configure the real-time behaviour of the system. On the other hand, the Real-Time Manager Unit (RTMU) controls all of the system’s real-time behaviour It may produce a task switching of the PU, whilst at the same time managing each event which takes place during runtime according to the information stored in the Event as well as the Task’s structures. This information may be modified by executing special real-time instructions in the task code. Each time that the PU recognises one of the real-time instructions, the RTMU takes control and executes it. One of these instructions is, for instance, the exit one, which ends the execution of the task that the PU is executing.

When the PU recognises an exit operation code, the RTMU takes control and changes the status of the current executing task to waiting state whilst also modifying the internal registers of the PU in order to execute the next task which is ready to be executed.

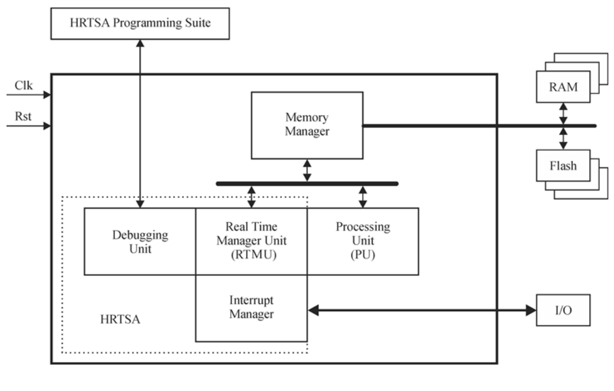


Fig. 1. The HRTSA-based processor architecture

The interrelation and communication between the RTMU and the PU is carried out through a Communication Interface. Soft-processors offer different alternatives to implement the Communication Interface such as: custom instruction, co-processing or source code modification.

The task which the PU executes is selected by the RTMU according to the real-time information. Each time that the RTMU is required to perform a task switching, it performs the following actions: (1) waits until the completion of the instruction that is being executed by the PU (PU is not preempted during an instruction cycle), (2) stops the PU before the next fetch cycle, (3) saves all the registers of the PU into the Task Structure and, (4) restores the registers of the PU with the ones of the next executing task. Therefore, task switching is performed by modifying the registers of the PU (program counter included). Consequently, when the RTMU releases the PU, it will be executing the new task. The task switching may be implemented, for instance, by: (1) the RTMU directly accesses to the PU´s registers (as it was implemented in the case study in this paper) or (2) the RTMU forces to the PU to execute a task switching routine. However, alternative mechanisms may be adapted according to the features of the soft-processor chosen.

The Interrupt Manager unit deals with external events that may invoke real-time tasks. Each time an enabled external event takes place, the Real-Time Manager Unit executes the Associated Action. Therefore, sporadic real-time tasks invoked by external events are supported in a way similar to that of real-time periodic tasks.

The Memory Manager implements memory access arbitration in order to allow for sharing the same external memory between the HRTSA and the PU.

The performance, adaptability and scalability of the architecture depend on how the real-time information is both stored and handled by the RTMU. This information includes: task structures, event structures and the action code associated with each event. The following sections describe the real-time information in detail.

IV. Time and Events in a Real-Time System

Time is important in a real-time system 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, and abortion are some examples of events in a real-time system.

According to the nature of the event, we can differentiate between *synchronous* and *asynchronous* events. We say that an event is synchronous when its occurrence can be expressed as a function of the clock of the system.

On the contrary, we say that an event is asynchronous when its occurrence depends on an external signal which is not a function of the clock of the system.

While external interrupts are asynchronous events, timers produce synchronous ones. Depending on the base of time considered, synchronous events may be classified in two categories:

1. *absolute* time events: this category includes the events whose occurrence is a function of the system time. Release and deadline of periodic tasks are examples of absolute time events.
2. *relative* time events: this category includes the events whose occurrence is a function of the executed time of a task. These events are related to the time at which the task has been executed. Usually, fault-tolerance strategies suggest the implementation of relative time events.

For example, in an EDF policy the priority of each task is determined according to the time remaining until its deadline. Let us consider a real-time system with two periodic tasks whose periods (*T*), deadlines (*D*) and maximum execution times (*C*) are detailed in Table I.

TABLE I
EXAMPLE OF REAL-TIME PARAMETERS

	T [ms]	*D* [ms]	*C* [ms]
Task 1	6	6	3
Task 2	8	7	4

Fig. 2 shows the evolution of the system scheduled under an EDF policy after a simultaneous release of both tasks. Each down-arrow represents the release of a task.

The priority of each task is determined according its deadline: the closer to its deadline, higher its priority.

The use of the processor is granted to the highest priority task. The priority of each task can be determined by configuring absolute events each time the task is released: the deadline of task 1 is set to 6 each time task 1 is released and, similarly, it is set to 7 for task 2.

Fig. 2 shows the value of the deadline at 1ms intervals, but it is a value that modifies in a time-continuous way.

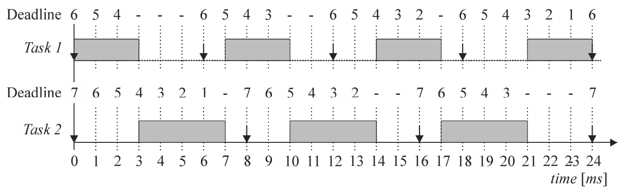

Deadline 6 5 4 - - - 6 5 4 3 - - 6 5 4 3 2 - 6 5 4 3 2 1 6
Task 1
Deadline 7 6 5 4 3 2 1 - 7 6 5 4 3 2 - - 7 6 5 4 3 - - - 7
Task 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
time [ms]

Fig. 2. Example of synchronous events

Time and events should be adequately supported by the real-time processor. It is mandatory to keep the absolute time of the system in an internal register of the RTMU. Such a register in the HRTSA architecture is known as *System Time Register* (STR). The RTMU increases this register one unit per system's clock period.

The range of the STR should be long enough to avoid an overrun during the lifetime of the system.

For instance, an STR of 64 bits takes 58,494 years to overrun with a system's clock equal to 10MHz. Similarly, the *Relative Time Register* (RTR) counts how long the current executing task has been executed. Of course, the RTR is accordingly updated when tasks are switched.

On the other hand, events should be adequately defined in order to describe the real-time behaviour of the system.

The next time an event takes place is called *Occurrence Time*. While the occurrence times of absolute time events are related to the value of the STR, the occurrence times of relative time events are related to the value of the RTR of the respective task. Therefore, an event is, in essence, defined by its occurrence time and its *Associated Action*. The action associated with an event is a code program stored in the memory of the system and it is executed by the RTMU when the event takes place.

IV.1. *Event Structure*

The *Event Structure* stores all of the event information (Fig. 3). The occurrence time is stored in the *Occurrence Time* field (OcF) of the respective event structure. The RTMU continuously checks whether: (1) the STR reaches the value of some of the Occurrence Fields of the absolute events and (2) the RTR reaches the OcF of the relative events of the executing task.

The starting address of the associated action is held in the *Code_Address* field of the event structure.

63 … 48	47 … 32	31 … 16	15 … 0
STATUS	CODE_ADDRESS	PREVIOUS_EVENT	NEXT_EVENT
OCCURRENCE_TIME			
PERIOD			
DATA_1			
DATA_2			
DATA_3			
DATA_4			

Fig. 3. Event Structure

In order to efficiently manage the events of the system, the event structures are organised in a linked list sorted by its occurrence times. An RTMU register, referred to as *Next Absolute Event* (NAE), and a field named *Next_Event* in each event structure, are in charge of maintaining this organisation. Each time an OcF is modified, the Next Event register and fields are accordingly updated by the RTMU.

Fig. 4 demonstrates an example of the linked list organisation of the event structures. This linked list is dynamically updated accordingly each time that an event takes places (the event is deleted from the list and the associated action executed) or an event is enabled (the event is inserted into the linked list sorted by its *Occurrence Time* field).

Relative time events are organised in a similar way for each task. The maximum number of events supported depends on the physical amount of memory assigned to events structures.

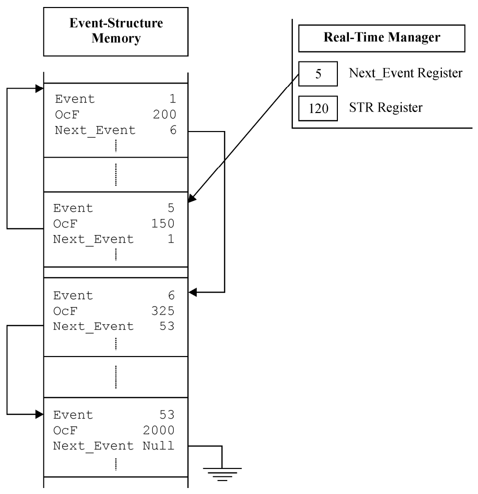


Fig. 4. Example of the Absolute time event structure organisation

V. Tasks and Priorities

Because most 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 must manage all the task's parameters needed to provide a multitasking environment. A *Task Structure* is defined to store each real-time parameter of the task. The task structures are stored into the memory of the system. The maximum number of tasks supported depends on the physical amount of memory assigned to task structures.

In a multitasking system, a priority is assigned to each task in order to schedule the set of tasks which are ready to be executed. The priority of each task may be modified or remain fixed during runtime according to the scheduling policy implemented.

Previous real-time processor approaches implemented a predefined scheduling policy over a reduced set of priorities ([28], [29], [30], [31], [32]).

V.1. *Task Structure*

The HRTSA does not execute a predefined scheduling policy but instead chooses to execute the highest priority task. The priority of each task is held in the *Ready_Priority* field of its respective task structure (Fig. 5). In order to schedule tasks efficiently, the RTMU keeps an updated linked list of task structures sorted by task priority.

The *Highest Priority Task register* (HPT) of the RTMU points to the highest priority task (smallest number). There is a field in each task structure, named *Next_Task*, which points to the immediate next priority task structure of the linked list.

The scheduling policy depends on how the priority of each task is assigned. Task's priorities may be changed by programming the Associated Action of either absolute or relative events, interrupts or exceptions.

Therefore, any arbitrary scheduling policy may be implemented.

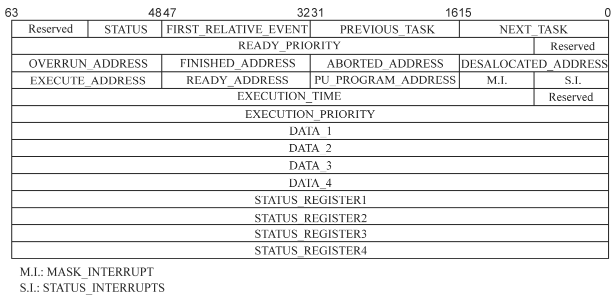


Fig. 5. Task Structure

For example, if actions associated with task release events modify the priority of their respective tasks to $OcF + D$ (priority is set to absolute deadline of the task), then an EDF policy is implemented. On the contrary, if actions do not modify task priority, then a Fixed Priority policy is implemented.

In the example shown by Fig. 2, it would be sufficient to set the priority of the task equal to its absolute deadline each time that a release event takes place in order to implement an EDF scheduling policy. As priority may be a function of the system time, the range of the priority of each task should be long enough in order to avoid an overflow.

Therefore, the length of task priority is the same number of bits as the STR register.

Because many events may be defined and each action may modify the task's priorities, a great many priority mechanisms may be implemented. In addition, as RTMU manages events and tasks, the execution of the tasks is not disturbed by scheduling. Scheduling overhead is almost neglected (just the task switching time) whatever the scheduling policy is. Additionally, a priority level may be assigned to each task when it is in executing state (setting the respective *Execution_Priority* field of the task structure).

In this way, it is easy to implement, for example, a non-preempted scheduling mechanism. It is only necessary to assign a higher priority to the executing state than any other priority of the tasks in ready state.

Each time that a task changes its status, an *Associated Action* takes place. Task structure stores the address of the program code of the action associated to each exception. Three states are supported:

- Wait state: the task was not invoked. The task structure is not inserted in the linked-list.
- Ready state: the task was invoked and requires executing. The task structure is inserted in the linked-list according to its Ready Priority.
- Execution state: the processor is granted to the task for execution. The task structure is removed from the linked list while the task is executing.

In this way, an action may be associated when a task changes from either wait to ready, ready to wait, ready to execution, execution to wait or execution to ready. The RTMU executes the action associated with each exception in the same way that it executes actions associated with events.

The STATUS_REGISTERS fields are reserved to save the PU's internal registers, preserving the status of the tasks when the task is preempted in order to restore it when the task switches to execution state again. Some soft-processors allow custom instruction or co-processing options for accessing the internal registers directly.

These alternatives, as well as modifying the source code, allow achieve a high switching performance since the saving and restoring operations of the PU's registers are done through a Direct Memory Access (DMA). If there is not a direct access from the HRTSA to the internal register of the PU, then the task switching should be performed through executing storing and restoring procedures by the PU.

VI. The Real-Time Manager Unit

The RTMU manages events, tasks and executes the action code associated with each event and exception which takes places. To do this, the RTMU must:

1. Update the STR and the respective RTR with a rate of one per time unit.
2. Execute the real-time instructions included in the task code. When the PU decodes an operation code for the RTMU, such as modifying certain real-time parameters or a task exit code, then the RTMU takes control and executes it.
3. Check if a certain event is taking place. The absolute time events and the relative time events are continuously monitored in order to establish whether the respective action code has to be executed.
4. Check if the executing task is the highest priority one. On the contrary, a task switching is performed in order to choose the highest priority task for execution.
5. Keep both the events and the tasks' structures organised. Each time that either an event or a task structure is modified, the event list and the priority list are accordingly updated.
6. Execute the action codes when a task changes its state. Each time that a task changes its state, the action code of the respective exception is executed according to the information stored in its task structure.

All these functions are carried out at the same time that the PU executes the system tasks. The performance handling real-time information is high and predictable because of the way in which events and tasks are organised.

Moreover, jitters and real-time performance of the processor are measured in periods of the system's clock instead of timer's ticks as is the case in real-time operating systems. Therefore, the performance is improved in several orders of magnitude.

VII. Memory Organization

A certain amount of system memory must be assigned to hold the following real-time information:

- The event structures.
- The task structures.
- The action codes.
- General-purpose data.
- Stack for next associated action codes.

The general-purpose data area may be accessed directly by the instructions that the RTMU executes. Important run-time information of the action codes may be stored in this area.

The stack for next action codes stores the addresses of the associated action codes that the RTMU has to execute when the current action code is completed. Execution of action codes is non-preempted. As an action code may call further actions codes, then the respective addresses must be saved in order to execute them later.

The maximum number of events and tasks as well as the amount of memory for data, code and stack depends on the physical amount of memory assigned to each one of them. To do this, real-time memory access is segmented.

Five internal segment registers of the RTMU point to the beginning of each area. Therefore, the amount of physical memory assigned to each area of memory is configured according to the value of the respective segment registers.

VII.1. Real-Time Configuration

The RTMU should be configured in order to produce the real-time behaviour that the application requires. 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 PU. When the RTMU is configured, it is rarely necessary to execute additional real-time instructions. However, real-time tasks can execute real-time instructions during runtime.

TABLE II
RTMU'S REGISTERS

Register	Description	Width [bits]
A	General Purpose Register	64
B	General Purpose Register	64
CET	Current Executing Task	16
CPri	Current Priority	64
CSR	Code Segment Register	16
DSR	Data Segment Register	16
ESR	Event Segment Register	16
ETP	Event to Process	16
Flags	Flag Register	8
HPT	Highest Priority Task	16
NAE	Next Absolute Event	16
NAOT	Next Absolute Occurrence Time	64
NRE	Next Relative Event	16
NROT	Next Relative Occurrence Time	64
PC	Program Counter	16
PriHPT	Priority Highest Priority Task	64
RTR	Relative Time Register	48
SP	Stack Pointer	16
SSR	Stack Segment Register	16
STR	System Time Register	64
TSR	Task Segment Register	16

TABLE III
HRTSA INSTRUCTIONS

Instruction	Description
add reg_d, reg_s	Adds the source register to the destination register, leaving the result in the destination register.
add reg_d, field_d, reg_s, field_s	Adds the specified field of the event pointed by the source register into the field pointed by destination register.
chstatus task reg, status	Changes the status of the pointed task by register to status.
chstatus task task, status	Changes the status of the task task to status.
cmp reg_d, field_d, reg_s, field_s	Compares the specified field of the event pointed by the source register into the field pointed by destination register.
dec reg_d	Decrements the destination register.
dec reg_d, field_d	Decrements the field pointed by the destination register.
disable event event	Disables the event event. When an event is disabled, it will not take place.
disable event reg	Disables the event pointed by register. When an event is disabled, it will not take place.
enable event event	Enables the event event and consequently it will take place according to its configuration.
enable event reg	Enables the event pointed by register and consequently it will take place according to its configuration.
end	Finishes the execution of code in the Real-Time Manager Unit.
exit	Finishes the execution of the current executing task.
inc reg_d	Increments the destination register.
inc reg_d, field_d	Increments the field pointed by the destination register
init event event	Configures the structure of event to be considered by the Real-Time Manager Unit.
init task task	Configures the structure of task to be considered by the Real-Time Manager Unit.
jc address	Jumps if carry.
je address	Jumps to address address if zero flag is set.
jg address	Jumps if it is greater.
jl address	Jumps if it is less.
jmp address	Jumps unconditionally to address.
jnc address	Jumps if not carry.
jne address	Jumps if no zero.
load upu_pc, a	Sets the PC of task with the register A of the Real-Time Manager Unit.
mov event event, field, value	Copies the value in the specified event_field of the event structure.
mov event reg, field, reg	Copies the source register in the specified field of the event pointed by the source register.
mov reg, event reg, field	Copies the specified field of the event pointed by the source register into the destination register.
mov reg, reg	Copies the value in the source register to the destination register.
mov reg, task reg, field	Copies the specified field of the task pointed by the source register into the destination register.
mov reg, value	Copies the value value to the destination register.
mov reg_d, field_d, reg_s, field_s	Copies the specified field of the event pointed by the source register into the field pointed by destination register.
mov task task, field, value	Copies the value in the specified field of the task structure.
mov task reg, field, reg	Copies the source register in the specified field of the task pointed by the source register.
not reg	Performs the not bit-wise of register.
sub reg_d, reg_s	Subtracts the source register from the destination register, leaving the result in the destination register.
sub reg_d, field_d, reg_s, field_s	Subtracts the specified field of the event pointed by the source register from the field pointed by the destination register.

Once the RTMU is configured, it carries out all of the real-time functions at the same time that the PU executes the code of the real-time tasks.

The RTMU is a processing unit with the set of internal registers. The instructions executed by the RTMU are reported in Table III.

VIII. Example: Programming a Real-Time Application

Real-Time instructions should be executed to configure the real-time behaviour of the system. This configuration is very flexible and does not depend on the code of the real-time task. For instance, an EDF policy can be reduced to the correct configuration of the associated actions of the real-time events.

As an example, the configuration of the RTMU, in order to execute the real-time system in Table I, under an EDF policy, is described in the following sections.

VIII.1. Initialisation (Included at the Beginning of the Code Executed by the PU)

The initialisation of the Event Structure of EVENT1 should be included at the beginning of the code executed by the PU and could take the following form:

```
; Creates event 1 Structure.
INIT EVENT 1
; Set First Occurrence Time
MOV EVENT 1, OCCURRENCE TIME, 600000
; Store Period of Task 1.
MOV EVENT 1, DATA1, 600000
; Store Task 1 as task associated to event 1.
MOV EVENT 1, DATA2, 1
; Set the address of the associated action.
MOV EVENT 1, CODE ADDRESS, RELEASE_START
; Enable event 1.
ENABLE EVENT 1
```

This code initialises Event 1, configures the variable DATA1 with the period of the task (600 ms), configures the variable DATA2 with the task associated with the event (Task 1) and configures the Associated Action at address RELEASE_START. Finally, the initialisation code enables Event 1 (the event structure of event 1 is inserted in the linked list of absolute events). The initialisation of event 2 is similar.

The initialisation of the Task Structure of Task 1 could take the following form:

```
; Creates Event 1 Structure.
INIT TASK 1
;Sets the beginning of Task 1's code.
MOV TASK 1, Task_Program_Code_Address, Start_Addr_Task1
; No action when task is aborted.
MOV TASK 1, Task_Code_Address_Aborted, NO_CODE
; No action when task's
; status changes to Execution.
MOV TASK 1, Task_Code_Address_Execution, NO_CODE
; No action when task is pre-empted.
MOV TASK 1, Task_Code_Address_Desallocated, NO_CODE
; No action when task's status changes to Ready
MOV TASK 1, Task_Code_Address_Ready, NO_CODE
; No action when task finishes.
MOV TASK 1, Task_Code_Address_Finished, NO_CODE
; Set task's priority to 60000
; when task is in execution state.
MOV TASK 1, Task_Ready_Priority, 60000
; Store the deadline of the task (600ms).
MOV TASK 1, DATA2, 60000
```

The initialisation of the task structure configures the associated actions for each event that the task produces.

These events are: Aborted, Execution, Desallocated, Ready and Finished. In this example, only the Ready event is configured. Initialisation of Task 2 is similar.

VIII.2. Associated Actions

The real-time policy is performed executing the associated action of the different events during runtime.

The associated actions may be programmed taking into account the ETP register of the RTMU. The ETP register holds the index of the event which has taken place. In this way, the associated action can be parameterised using this register.

Each time that Event 1 or Event 2 takes place (the occurrence time is reached) the event is configured (the occurrence time is set to the current occurrence time plus the period of the task). The status of the task is changed to Ready and consequently is inserted into the linked list of ready tasks. The Ready Priority of the task is set equal to the absolute deadline of the task to implement an EDF policy among the task of the system. The Execution Priority of the task is set equal to the Ready Priority to implement a preemptable scheduling policy. Otherwise, if a non-preemptable policy is desired, then the Execution Priority might be set equal to zero.

Therefore, the associated action with Events 1 and 2 could be as follows:

```
RELEASE_START:
; The occurrence time is added to the period
; of the task stored in the variable DATA1
; and the event is enabled.
ADD ETP, OCCURRENCE_TIME, ETP, DATA1
INSERT ETP IN ABSOLUTE LIST
; The ETP register is loaded with the
; task associated with event (stored
; in variable DATA2.)
MOV A, EVENT ETP, DATA2
MOV ETP, A
; Register B of the RTMU is added to
; the deadline of the task (stored
    ; in variable DATA1 of the task structure).
    MOV B, TASK ETP, DATA1
    ADD A, B
    ; The Ready Priority is set equal to the
    ; absolute deadline to perform an EDF policy
    ; as well as the Execution Priority
    ; to make it preemptive.
    MOV TASK ETP,TASK_READY_PRIORITY, A
    ; The status of the task is changed to ready
    MOV TASK ETP, TASK_EXECUTION_PRIORITY,
A
    CHSTATUS TASK ETP, ready
    ; The associated action is ended
    END
```

IX. Performance Evaluation

The HRTSA was described in VHDL, synthesised using Quartus II v11.0 and implemented in a Cyclone III FPGA device from Altera. The architecture required approximately 3500 LEs. The system includes a 16Mb Mobile SDRAM memory to store the tasks' codes and the real-time information.

A set of 560 real-time systems was randomly generated in order to evaluate the performance of the HRTSA. Each real-time system contains 10 real-time tasks. The period of each task (T) was randomly generated between 800μs and 8000μs, whilst deadlines (D) were set equal to the periods and the worst case execution times (C) were generated to produce a total utilisation factor equal to 0.7, considering a system clock frequency equal to 10MHz. While the period and deadline depends on the specification of the real-time application, the execution time of a task is a function of the time required to execute such a task. Therefore, when the system clock frequency is increased, the total utilisation factor decreases.

For comparison purposes, a real-time scheduler was implemented to execute each one of the real-time systems generated. It was implemented as a timer routine (TR), as it is in a RTOS, and it does not execute any real-time instruction of the RTMU. When a real-time system based on TR is considered, the interval in which the scheduler must be invoked to execute the scheduling policy should be defined. Different invocation intervals were implemented to analyse its influence on the execution of the real-time system.

The invocation interval of the scheduler was set to 130μs, 150μs and 175μs. When a RTOS is utilised, a task's period should be expressed in units of the invocation interval of the scheduler.

Consequently, the periods of the tasks were rounded to the upper multiple of the invocation interval of the scheduler. This choice reduced the total utilisation factor of the systems implemented in RTOS but allowed a conservative comparison with the performance of the HRTSA.

On the other hand, the HRTSA can achieve a precision of 100ns with a 10MHz clock and consequently the implementation of the real-time system based on a HRTSA is more accurate.

The evaluation of both systems, TR and HRTSA, was performed by modifying the system clock frequencies to 10, 20, 40, 50, 80 and 100MHz. The jitter was used as a measure to compare the performance of the system. The jitter is defined as the difference between the period of the task and the interval between the two consecutive starting times of the task during runtime.

As the execution pattern of the tasks is not fixed, the starting time of the tasks is not periodic. The maximum jitter of each task was stored and the average among the 560 systems for task 1, 2, 6 and 10 are shown in Figs. 6.

The HRTSA-based processor can schedule the systems with a 10MHz clock frequency whilst the TR-based systems needed at least a 50MHz clock frequency to schedule them.

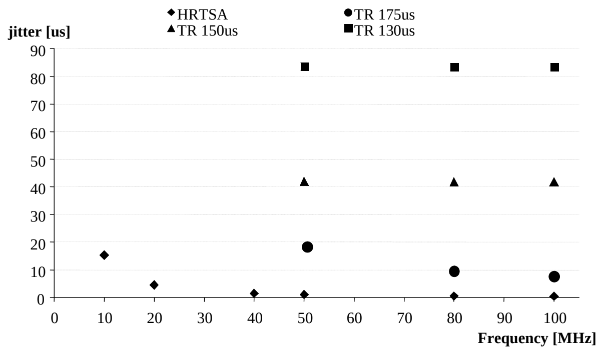
X. Result Analysis

From the results, the overhead of the system produced by the scheduler is reduced with a HRTSA-based implementation and the system is schedulable at lower clock frequencies with the following advantages:

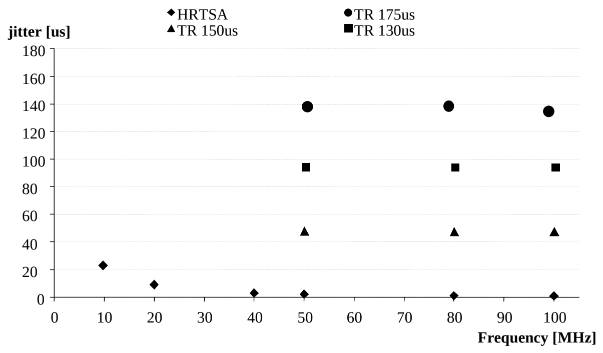
- The power consumption of the system may be reduced. The power dissipation of a digital circuit is proportional to the clock frequency of the system. Because the system can be scheduled at a lower clock frequency, the voltage and consequently the power consumption of the system may be dramatically reduced [33]. Moreover, the HRTSA avoids the overhead introduced by the RTOS and as a result the power consumption is also reduced.
- The cost of the technology required may be reduced. When lower clock frequencies are applied, lower-cost technologies can be used in processor implementation, debugging and interfacing. Hence, a more sophisticated device should be needed if a system based on RTOS is implemented.
- The electromagnetic interference is reduced when lower clock frequencies are applied and consequently systems can easily meet electromagnetic standards.
- The accuracy of the implementation of the real-time system is improved. With a 10MHz clock, the real-time parameters can be expressed with a precision of 100ns. When a RTOS is utilised, the real-time parameters should be modified to be a multiple of the invocation interval of the scheduler.

On the other hand, from the application point of view, the performance is improved when a HRTSA is utilised because the maximum jitter of the tasks is reduced. This feature may be very important when the real-time system is applied to control or digital signal processing applications ([34], [35], [36]).

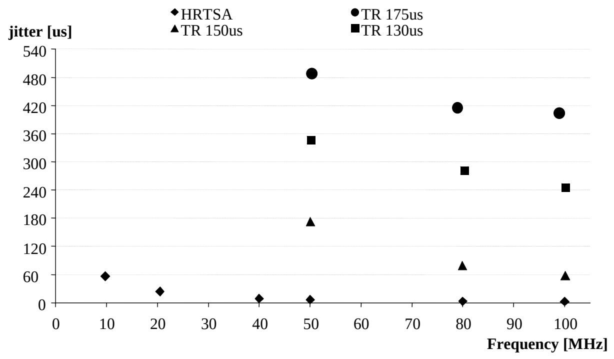
In a real-time system based on an RTOS, the interval time of the scheduler invocation may have a greater influence on the control performance than the clock frequency.



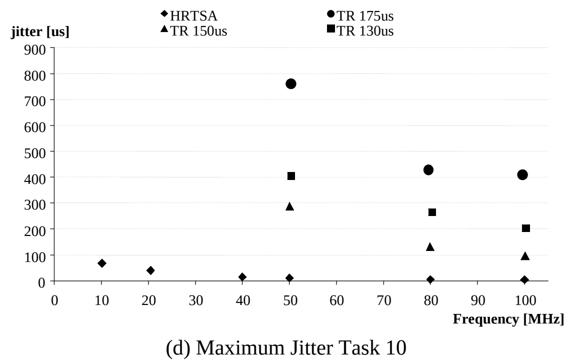
(a) Maximum Jitter Task 1



(b) Maximum Jitter Task 2



(c) Maximum Jitter Task 6



(d) Maximum Jitter Task 10

Figs. 6. Maximum jitter of real-time tasks

Increasing the clock frequency in an RTOS-based system may not proportionally decrease the jitter introduced. There may not be an optimal interval time of scheduler invocation for all tasks: whilst 175μs is optimal for task 1, a slot time of 150μs is optimal for tasks 2, 6 and 10. On the other hand, we can observe that the jitter in an HRTSA-based processor is just proportional to the clock frequency.

XI. Conclusion

Embedded and industrial applications require digital processing architectures in order to meet real-time constraints. Real-time theory offers diverse scheduling mechanisms with varied real-time features, each of which is suitable for different applications. Most of these mechanisms consider a scheduler as part of the RTOS.

However, when the scheduling mechanisms are complex, the chances of an efficient implementation in an embedded system are reduced. Consequently, most RTOSs implement a Fixed Priority policy or a Cyclic Executive that offers a deterministic performance with a low overhead cost. RTOSs with different scheduling policies are restricted to systems whose capacity enables the implementation of more complex scheduling mechanisms. Several papers have proposed the implementation of scheduler functions in hardware in order to improve the real-time efficiency of the system. Most of these papers are based on transferring the scheduling functions of the RTOS to hardware. However, while the runtime efficiency increases ([16]), the real-time restrictions originating from the software version implemented in RTOSs remain in hardware. Usually, hardware schedulers migrated from RTOS implement a certain scheduling policy which cannot be easily modified. On the other hand, real-time processors with scheduling features in hardware are based on the possibility of executing a reduced set of real-time tasks under a predefined scheduling policy chosen by the designer of the processor.

Modern Field Programmable Gate Arrays (FPGAs) offer the possibility to implement Systems-on-Programmable-Chip based on soft-processors.

The architecture of these processors may be modified to add new features in order to make it more suitable for the target application. Some of these features may include custom instructions and inter-processor interfaces, useful for Hardware/Software Co-design methods. In this paper, the Hardware Real-Time Scheduling Architecture designed to incorporate real-time properties with soft-processors is described. The HRTSA can be configured to implement any scheduling policy and support a large number of real-time tasks.

The HRTSA is described including the events and task structures to configure the real-time features of the application. A simple example is sketched to show the easiness with which an EDF scheduler can be implemented. Experience shows that the efficiency of the HRTSA allows for scheduling of the real-time system at a very low frequency. While the HRTSA-based processor schedules a real-time application with a 10MHz clock, a system scheduled with a timer routine needed at least a 50MHz clock. The HRTSA is easily adaptable to soft-processors in order to improve the real-time performance of the system implementing the more adequate real-time methodology.

Acknowledgements

This work was supported in part by the Program 2012 UAM-Santander Bank University Cooperation Projects with Latin America. Additional funds have been obtained from *Convenio Marco* UNS-UAM 2010-2015.

References

[1] IEEE1003.1d-1999, IEEE Standard for Information Technology-Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API)- Amendment D: Additional Real time Extensions [C Language], 1999.

[2] John A. Stankovic and Krithi Ramamrithan, "The Spring Kernel: a new paradigm for real-time systems," IEEE Software, Vol.3, No. 3, pp.62-72, 1991.

[3] Dan Hildebrand, "An Architectural Overview of QNX", Proceeding of the Workshop on Micro-Kernels & Other Kernel Architectures, pp. 113-126, Seattle, 1992.

[4] Jean J. Labrosse, “MicroC/OS-II: The Real Time Kernel”, CMPBooks, 2002.

[5] Jean J. Labrosse, “uC/OS-III, The Real-Time Kernel”, Micrium, 2009.

[6] Arnaldo Oliveira, Luís Almeida, and António de Brito Ferrari, "The ARPA-MT Embedded SMT Processor and its RTOS Hardware Accelerator," IEEE Transactions on Industrial Electronics, vol. 58, No. 3, pp. 890-904, March 2011, 2011.

[7] Hamdaoui, F., Ladgham, A., Sakly, A., Mtibaa, A., Real time implementation of medical images segmentation using Xilinx System Generator, (2012) *International Review on Computers and Software (IRECOS)*, 7 (6), pp. 2861-2867.

[8] Jason Agron, Wesley Peck, Erik Anderson, David Andrews, Ed Komp, Ron Sass, Fabrice Baijot, and Jim Stevens, "Run-Time Services for Hybrid CPU/FPGA Systems on Chip," Proceeding of the 27th IEEE International Real-Time Systems Symposium, pp. 3-12, Dec. 2006.

[9] Li Yan, Li Xian-yao, Gu Ping-ping, Zhao Hong-jie, and Cheng Ping, "Hardware Implementation of uC/OS-II based on FPGA,", Proceeding of the 2nd International Workshop on Education Technology and Computer Science (ETCS), pp. 825-828, March 2010.

[10] Elhamzi, W., Saidani, T., Said, Y., Atri, M., FPGA based Real Time wavelet Video coding, (2013) *International Review on Computers and Software (IRECOS)*, 8 (1), pp. 243-249.

[11] Melissa Vetromille, Luciano Ost, César Marcon, Carlos Reif, and Fabiano Hessel, "RTOS Scheduler Implementation in Hardware and Software for Real Time Applications", Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping, pp. 163-168, June 2006.

[12] Pramote Kuacharoen, Mohamed Shalan, and Vincent Mooney III, "A Configurable Hardware Scheduler for Real-Time Systems", Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, pp. 96-101, 2003.

[13] Ondrej Krejcar, Petr Tucnik, Ondrej Adamec, “Evaluation of aJile aJ-80 Real-Time embedded platform for RT-Java parameters”, Measurement, Elsevier, Vol. 44, Issue 7, pp. 1253-1260, August 2011.

[14] Tiago Muck, Antonio Frohlich, Michael Gernoth, Wolfgang Friedrich, “Implementing OS components in hardware using AOP”, ACM SIGOPS Operating Systems Review, Vol. 46, Issue 1, pp.:64-72, Jan. 2012.

[15] Rajeswari, P., Nagarajan, N., Real time network traffic monitoring using FPGA, (2013) *International Review on Computers and Software (IRECOS)*, 8 (7), pp. 1658-1662.

[16] Jaehwan Lee, Vincent John Mooney III, Anders Daleby, Karl Ingström, Tommy Klevin and Lennart Lindh, "A Comparison of the RTU Hardware RTOS with a Hardware/Software RTOS", Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2003), pp. 683-688, Jan. 2003.

[17] Octavian Cheng, Waleed. Abdulla and Zoran Salcic, "Hardware-Software Codesign of Automatic Speech Recognition System for Embedded Real-Time Applications," IEEE Trans. on Industrial Electronics, Vol. 58, No.3, pp. 850-859, March 2011.

[18] Alfredo Rosado-Muñoz, Manuel Bataller-Mompeán, Emilio Soria-Olivas, Claudio Scarante and Juan F. Guerrero-Martínez, "FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches", IEEE Trans. on Industrial Electronics, vol. 58, No. 3, pp. 860-870, March 2011.

[19] Joshua Weber, Erdal Oruklu and Jafar Snaiie, "FPGA-based Configurable Frequency-Diverse Ultrasonic Target-Detection System", IEEE Trans. on Industrial Electronics, vol. 58, No. 3, pp. 871-879, March 2011.

[20] M.A. Aguirre, J.N. Tombs, V. Baena-Lecuyer, J.L. Mora, J.M. Carrasco, A. Torralba, L.G. Franquelo, “Microprocessor and FPGA interfaces for in-system co-debugging in field programmable hybrid systems”, Microprocessors and Microsystems, Elsevier, Vol. 29, Issue 2, pp. 75-85, April 2005.

[21] Mahmoud Hamouda, Handy Fortin Blanchette, Kamal Al-Haddad and Farhat Fnaiech, "An Efficient DSP-FPGA-Based Real-Time Implementation Method of SVM Algorithms for an Indirect Matrix Converter", IEEE Trans. on Industrial Electronics, vol. 58, No. 11, pp. 5024-5031, Nov. 2011.

[22] Nagarajan, V., Waran R., Srinivasan, V., Kannan, R., Thinakaran, P., Hariharan, R., Vasudevan, B., Nachiappan, N.C., Saravanan, K.P., Sridharan, A., Sankaran, V., Adhinarayanan, V., Vignesh, V.S., Mukundrajan, R., “Compilation Accelerator on Silicon”, Proceedings IEEE Computer Society Annual Symposium on VLSI, pp.:267-272, Aug. 2012.

[23] Chang. L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM (JACM), vol. 20, No. 1, pp. 46-61, Jan. 1973.

[24] Houssine Chetto and Maryline Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, Vol. 15, No. 10, pp. 1261-1269, Oct. 1989.

[25] J. Urriza, L. Schorb, J. Orozco, R. Cayssials, “Reduced Computational cost in the Calculation of Worst Case Response Time for Real-Time Systems”, Journal of Computer Science & Technology, Vol. 9, Octuber 2009.

[26] Joseph Y. Leung and Jennifer Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks", Performance Evalaluation, Elsevier, Vol. 2, No. 4, pp. 237-250, Dec. 1982.

[27] Richard J Vaccaro, “Digital Control: A State-Space Approach”, McGraw-Hill College, 1995.

[28] Matjaz Colnaric and Wolfgang A. Halang, "Architectural support for predictability in hard real time systems", Control Engineering Practice, Elsevier, Vol. 1, No. 1, pp. 51-57, Feb. 1993.

[29] Vlado Glaviníc, Stjepan Gros, and Matjaz Colnaric, "VHDL-based modeling of a hard real-time task processor", Proceeding of the IEEE International Symposium on Industrial Electronics (ISIE’99), Vol. 1, pp.49-54, Jul. 1999.

[30] Matjaz Colnaric, Domen Verber and Wolfgang A. Halang, "Supporting High Integrity and Behavioural Predictability of Hard Real-Time Systems," Informatica (Slovenia), Special Issue on Parallel and Distributed Real-Time Systems, Vol. 19, No.1, pp. 59-69, February 1995.

[31] Joakim Adomat, Johan Furunäs, Lennart Lindh, and Johan Stärner, "RealTime Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems", Proceedings of the in 8th Euromicro Workshop on Real Time Systems, pp. 164-168, June 1996.

[32] Steven Miller, David Greve, Matthew Wilding and Mandayan Srivas, "Formal Verification of the AAMP-FV microcode", NASA Langley Technical Report, MD21076-1320, 1999.

[33] Clive Watts and Ravi Ambatipudi, "Dynamic Energy Management in Embedded Systems", Computing and Control Engineering, IEE, Vol. 14, No. 5, pp.36-40, Oct. 2003.

[34] Manuel Lluesma, Anton Cervin, Patricia Balbastre, Ismael Ripoll and Alfons Crespo, "Jitter Evaluation of Real-Time Control Systems", Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 257-260, Sep. 2006.

[35] Daniele Fontanelli, Luigi Palopoli and Luca Greco, "Deterministic and Stochastic QoS Provision for Real-Time Control Systems", Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 103-112, April 2011.

[36] Frederick M. Proctor and William P. Shackleford, "Real-time Operating System Timing Jitter and its Impact on Motor Control", Proceedings of the SPIE Sensors and Controls for Intelligent Manufacturing II, Volume 4563, pp. 10-16, October 28, 2001.

Authors' information

[1]Universidad Nacional del Sur -DIEC / Av. Alem 1253 – Bahía Blanca – Argentina.

[2]Universidad Tecnológica Nacional – FRBB / 11 de Abril 464 – Bahía Blanca – Argentina.

[3]CONICET / Av. Alem 1253 – Bahía Blanca – Argentina.

[4]Universidad Nacional de la Patagonia San Juan Bosco / Brown 3025 – Puerto Madryn – Argentina.

[5]Universidad Autónoma de Madrid – Madrid- España.

Ricardo Cayssials received the Engineer degree in electronics in 1993 and the PhD degree in engineering in 1999 from the National Southern University, Bahía Blanca, Argentina. Since 1994, he has been with the National Southern University, where he is currently Adjunct Professor and Researcher at CONICET. He also is professor at the National Technological University in Bahia Blanca (UTN-FRBB). In 2001 and 2003 he granted two postdoctoral stays at the University of York, York, England. His research interests include real-time system, programmable logic devices and system on programmable chip. He is senior member of IEEE.

Edgardo Ferro received the Engineer degree in electronics in 1990 and the PhD degree in engineering in 1999 from the National Southern University, Bahía Blanca, Argentina.
Since 1990, he has been with the National Southern University, where he is currently Adjunct Professor. His current research interests include real-time system, Industrial Automation, fieldbus communications and system on programmable chip.

José Urriza is an Adjunct Professor at the Universidad Nacional de la Patagonia San Juan Bosco in Puerto Madryn, Argentina. He received the Engineer degree in electronics in 1998 and the PhD degree in engineering in 2007 from the National Southern University, Bahía Blanca, Argentina. His research interests include schedulability of real-time systems and dynamic voltage scheduling.

Eduardo Boemo is a titular professor of ASIC design at the School of Computer Engineering of the Universidad Autónoma de Madrid. His current research interests include the design of FPGA-based systems, low-power techniques, computer arithmetic, self-timed circuits and electrical engineering education. He received the electrical engineering degree from the Universidad Nacional del Mar del Plata (Argentina) and the PhD degree in telecommunications engineering from the Universidad Politécnica de Madrid (Spain) in 1985 and 1998, respectively.