Real-Time Scheduling Coprocessor for NIOS II Processor

M. Varela, R. Cayssials[1], E. Ferro
Universidad Nacional del Sur – [1]CONICET
Department of Electrical Engineering
[1]UTN - FRBB
Bahía Blanca - Argentina
ricardo.cayssials@uns.edu.ar

E. Boemo
Universidad Autónoma de Madrid
Departamento de Tecnología Electrónica y
de las Comunicaciones
Madrid – España

Abstract— **In this paper we describe and analyze the main features of the Hardware Real-Time Scheduler Coprocessor unit (HRTC) for NIOS II processor. We describe how the HRTSC supports time, events, task and priorities. The HRTSC was designed as a SOPC component to incorporate real-time features for embedded real-time applications. The hardware architecture has an easy integration with the IDE programming environment. The Avalon interface showed to be an efficient specification to share memory and data communication among memory, processor and HRTSC. The performance of the HRTSC architecture is analyzed considering real-time flexibility, programmability and power consumption reduction.**

Keywords; NIOS II Processor, Real-Time, SOPC Component

I. INTRODUCTION

Real-time systems are ubiquitous and diverse, and include mobile phones, entertainment devices, automobiles, toys, smartcards, medical devices, network switching equipment, sensors, industrial robots, among others. 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. RTOSs allow designer 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. Spring kernel [1], QNX [2], uC/OSII [3], among others). The RTOS includes a special system task, named the scheduler, which shares the processor among the tasks that require execution. The scheduler carries out a priority discipline to determine the next task to be executed. The real-time behavior of the system depends on the priority discipline implemented. Each RTOS applies a priority discipline 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.

This work was supported in part by the Program 2011 UAM-Santander Bank University Cooperation Projects with Latin America. Additional funds have been obtained from *Convenio Marco* UNS-UAM 2010-2015.

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 usually generates hard to debug failures.

Several papers have proposed the implementation of scheduler functions in hardware in order to improve the real-time efficiency of the system ([4], [5], [6], [7], [8], [9], [10],[11]). Most of the ideas are based on transferring the scheduling functions of the RTOS to hardware. However, while the runtime efficiency increases ([12]), the real-time restrictions originated from the software version of the RTOSs remain in hardware. Usually, hardware schedulers migrated from RTOS implement a certain priority discipline over a reduced number of tasks that cannot be easily modified. Moreover, real-time features cannot be included to others processors since the whole architecture of the processor is designed to include those real-time characteristics.

In this paper, an architecture, called Hardware Real-Time Scheduling Coprocessor unit (HRTSC), able to incorporate real-time scheduling properties to Altera´s NIOS II processor, is proposed. We describe and analyze the main features of the HRTSC for NIOS II processor to support time, events, task and priorities. The HRTSC may support any real-time priority mechanism over a set of up to 65,000 tasks. The HRTSC is scalable and consequently the number of tasks and events supported depends on the amount of physical memory of the system. When the HRTSC is utilized, temporal parameters can be defined independently of the functionality of each task. Hence, task's code may not include real-time programming which permits a higher abstraction level of implementation and a better reusability and flexibility in the design.

The paper is organized as follows: Section 2 describes the typical model used in real-time theory and the implementation of scheduling mechanisms in RTOS. Section 3 describes how time and events are supported by the HRTSC. The interface with the user application is presented in Section 4. Section 5 details the implementation of the HRTSC and the analysis the first experiences. Conclusions and future works are drawn in Section 6.

978-1-4673-0186-2/12/$31.00 ©2012 IEEE

II. REAL-TIME PROCESS MODEL AND RTOS SCHEDULING

A real-time system is modeled as a set **Π** of n periodic tasks to be executed on a processor. Each task i performs a certain function and its real-time behavior is characterized 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 RTOS 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 ([13]). Because the timer task is invoked at discrete 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 would 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 differences in the task invocation time 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.

Scheduler assigns the processor according to the priority discipline that it implements. Fixed Priorty (FP) and Earliest Deadline First (EDF) are two of the most important disciplines in real-time ([14]). In a FP discipline, each real-time task is assigned with a priority in the design time and it remains fixed during runtime. In an EDF discipline, the priority of a task increases as closer it is to its deadline. The EDF discipline is optimal in the sense that if the system is not schedulable under an EDF discipline, then the system is not schedulable under any other priority discipline. However, the complexity of the EDF discipline may turn difficult to achieve an efficient implementation, compared with a Fixed Priority one, in a RTOS for embedded systems. The HRTSC proposed can be configured to implement any arbitrary priority discipline with a very low overhead on the system processor.

III. THE HRTSC ARCHITECTURE

The HRTSC unit was designed as a component of the Altera's SOPC Builder. The interface with the system was defined according to the Avalon-MM specification ([15]) and consequently the SOPC Builder generates the system interconnect fabric, producing an easy integration with the NIOS II Processor. Figure 1 shows a layout of a system-on-chip including a HRTSC unit:

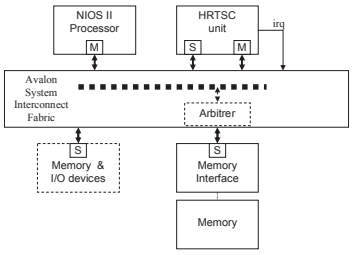

Figure 1: HRTSC component in a SOPC.

The HRTSC unit is memory-mapped to the NIOS II system fabric. The HRTSC has two Avalon-MM ports:

- a slave port: used by the NIOS II to configure the real-time features of the HRTSC and to read its status, and
- a master port: used by the HRTSC to access the real-time data structures stored into the memory of the system specially assigned to it.

The HRTSC has an interrupt to request service from the NIOS II processor.

The HRTSC stores all the real-time information that it requires to execute the real-time tasks into the RAM memory of the system. This information includes release time, deadline, priority and state of each real-time task. The HRTSC checks if there exists an event that forces to change the state of the real-time tasks in order to produce an interrupt request to the NIOS II processor.

The HRTSC holds the system time in an internal 64-bit register named *System Time Register* (STR). The HRTSC 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. The STR register increments with a 10MHz external clock (it takes 58,494 years to overrun the STR). With this frequency, both tasks' period and tasks' deadline can be expressed with a time resolution of 100ns. The NIOS II processor does not execute any timer subroutine and consequently there is not scheduler overhead. Hence, both the schedulability of the system and the response time of the real-time tasks are improved.

A. Time and Events

Time in a real-time system is important 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.

According to the nature of the event we can differentiate between *synchronous* and *asynchronous* events. We said that an event is synchronous when its occurrence can be expressed as a function of the clock of the system. On the contrary, we said that an event is asynchronous when its occurrence depends on an external or internal signal that it is not function of the clock of the system.

While external interrupts or processor exceptions are asynchronous events, timers produce synchronous ones. Asynchronous events are handled by both interrupts and exceptions. 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 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 function of the executed time of a task.

These events are related to the time that the task has been executed.

The HRTSC architecture adequately supports time and events. The HRTSC continually checks if there exists an event that requires to be activated or a task whose state should be modified.

The Event Structure stores all the event information (Fig. 2). The occurrence time is stored in the Occurrence Field (OcF) of the respective event structure. The HRTSC continuously checks if the STR reaches the value of some of the Occurrence Fields of the absolute events. The starting address of the associated action is hold in the Code_Address field of the event structure. The maximum number of events supported depends on the physical amount of memory of the system assigned to events structures. The fields of an event structure are showed in figure 2.

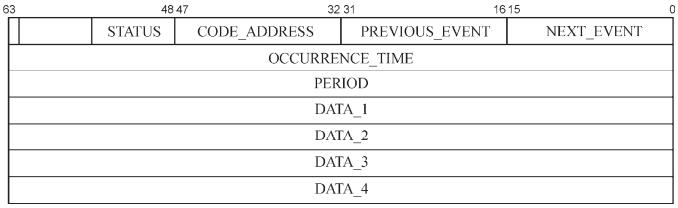

Figure 2: Event Structure.

In order to manage efficiently the events of the system, the event structures are organized in a linked list sorted by occurrence times. An HRTSC register, named *Next Absolute Event (*NAE), and a field with the same name in each event structure, are in charge of keeping this organization. Each time that an OcF is modified, the Next Event register and fields are accordingly updated by the RTU.

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 real-time multitasking environment. The HRTSC utilizes 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 system assigned to task structures.

Previous real-time processor approaches implemented a predefined scheduling discipline over a reduce set of priorities ([16], [17], [18]).

The HRTSC does not execute a predefined priority discipline but chooses for execution the highest priority task instead. The HRTSC may change the task's priority by configuring the action taken on every event. Consequently, a great deal of priority disciplines may be implemented. The priority of each task is hold in the *Ready_Priority* field of its respective task structure (Fig. 3). In order to schedule tasks efficiently, the HRTSC keeps updated a linked list of task structures sorted by task priority. The Highest Priority Task register (HPT) of the HRTSC 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, building a linked list similar to the event one.

The scheduling discipline 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. Therefore, any arbitrary priority discipline may be implemented. For example, if actions associated to task release events modify the priority of their respectively tasks to OcF + *D* (priority is set to absolute deadline of the task), then an EDF discipline is implemented. On the contrary, if actions do not modify task priority, then a FP discipline is implemented.

The maximum number of tasks supported depends on the physical amount of memory of the system assigned to task structures. The fields of a task structure are showed in figure 3.

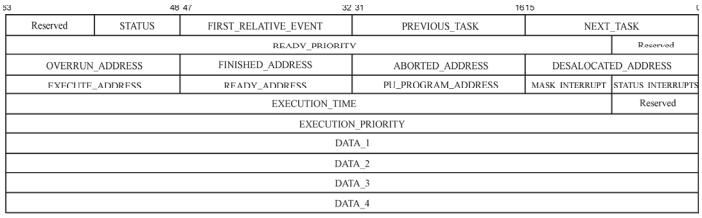

Figure 3: Task Structure.

Each time that a task changes its status, an *Associated Exception* takes place. Task structure stores the address of the program code of the action associated to each exception. In this way, an action may be associated when task changes from either wait to ready, ready to wait, ready to execution, execution to wait or execution to ready. The HRTSC executes the action associated to each task event in the same way that actions associated to events.

C. Real-Time Configuration

The HRTSC should be configured executing special HRTSC commands. Once the HRTSC is configured, it is rarely necessary to execute additional real-time commands. However, real-time tasks may execute real-time commands during runtime.

Once the HRTSC is configured, it carries out all the real-time functions at the same time that the NIOS II processor executes the real-time tasks.

The HRTSC is a co-processing unit with the set of internal registers described in Table 1.

TABLE 1: HRTSC REGISTERS

Register	Description	Width [bits]
STR	System Time Register	64
CET	Current Executing Task	16
CPri	Current Priority	64
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
PriHPT	Priority of the Highest Priority Task	64
RTR	Relative Time Register	48
PC	HRTSC Program Counter	16
SP	HRTSC Stack Pointer	16
A	General Purpose Register	64
B	General Purpose Register	64

The commands executed by the HRTSC may be categorized in four groups: arithmetic (addition, subtraction, decrement and increment), data transfer (mov), flow control (comparison and conditional jumps) and real-time. The Table 2 lists the real-time commands defined to manage the real-time structures of the system.

TABLE 2: HRTSC REAL-TIME COMMANDS

Command	Description
chstatus *task, status*	Changes the status of the task *task* to *status*.
disable event *event*	Disables the event *event*. When an event is disabled, it will not take place.
enable event *event*	Enables the event *event* and consequently it will take place accordingly to its configuration.
end	Finishes the execution of commands in the HRTSC.
exit	Finishes the execution of the current executing task.
init event *event*	Configures the structure of *event* to be considered by the HRTSC.
init task *task*	Configures the structure of *task* to be considered by the HRTSC.

The HRTSC is continuously checking if an event takes place in order to interrupt the processor to change the status of the real-time tasks. Each event is associated with an action that the HRTSC executes when the event takes place. The real-time behavior of the system will be defined according the commands programmed for each associated action of the events. Since there could exist several events that the HRTSC should execute at a certain time, the HRTSC pushes each action into a stack memory for further execution. The Figure 4 shows a layout of the HRTSC datapath.

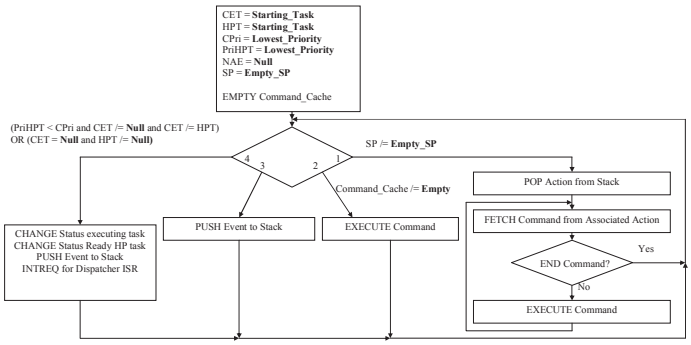

Figure 4: HRTSC behavior.

There exist four conditions that the HRTSC continuously checks:

1. The Stack is not empty: then the associated action is executed. The execution of associated actions may produce further actions that should be pushed into the stack.
2. The NIOS II requires the execution of a command: then the command is executed when the command cache of the HRTSC is not empty.
3. The STR reached the Occurrence Time of the next event: then the associated action of the event is pushed into the stack.
4. The priority of a ready task is higher than the Current Executing Task or the processor is idle and there is at least a task in ready state: then the Current Executing Task is preempted and the processor is granted to the Highest Priority Task in ready state.

In this way, the HRTSC covers all the real-time behavior of the system, allowing the NIOS II processor to execute only real-time tasks avoiding the overhead of the real-time scheduler.

IV. THE HRTSC PROGRAMMING ENVIRONMENT

Altera provides NIOS II Software Built Tools for Eclipse as the IDE programming environment for NIOS II-based system on chip. Most of the applications utilize a Board Support Package (BSP) with the definition of the hardware resources for an easy interface with the software application.

The software interface between the NIOS II processor and the HRTSC was implemented as an Interrupt Software Routine (for task switching) and system calls (for HRTSC configuration) included in the Board Support Package.

Task switching is implemented as an Interrupt Service Routine that stores the NIOS II internal registers in memory to preserve the executing task status and restores the internal register for the highest priority task. Memory addressing for real-time storage are read from status registers of the HRTSC.

V. IMPLEMENTATION AND ANALYSIS

The first version of the HRTSC was designed for SOPC Builder 10.0 and implemented in a Cyclone IV FPGA in a BeMicro SDK development kit. The architecture required nearly 4000 LEs. The interrupt service routines and system calls were implemented in the BSP of the applications.

The HRTSC was tested with a 10MHz clock for simple applications containing up to four real-time tasks with different temporal requirements. From first experiences it could be noted the HRTSC efficiently interrupt the NIOS II processor when a task switching has to be performed. This reduces the power consumption of the system without any special strategy.

Two priority disciplines were implemented: Fixed Priority and Earliest Deadline First. The selection of the priority discipline is performed by changing the associated action of the events, without modifying the real-time tasks. In this way, the real-time constraints can be satisfied easily since it does not require any change of the application code that could be error-prone.

The overhead of the system (NIOS II´s processor time that is not utilized to execute real-time task code) for the applications tested was less than 2%. Most of the overhead was produced by the task switching routine.

A. EDF implemented

In this section we describe the associated actions programmed to implement an EDF discipline. This configuration is very flexible and it does not depend on the code of the real-time task. For instance, an EDF discipline can be reduced to the correct configuration of the associated actions of the real-time events. It can be noted, in spite of EDF is an optimal real-time discipline, it is not widely implemented because its complexity of implementation on traditional RTOSs.

1) Initialization (included at the beginning of the code executed by NIOS II processor):

TABLE 3: INITIALIZATION CODE FOR EDF DISCIPLINE

Code	Description
init event 1	Creates the Event 1 Structure
mov event 1, occurrence time, 600000	Set First Occurrence Time
mov event 1, data1, 600000	Store Period of Task 1
mov event 1, data2, 1	Store Task 1 as task associated to event 1
mov event 1, code address, release_start	Set the address of the associated action
enable event 1	Enable event 1

This code initializes the EVENT1, configures the variable DATA1 with the period of the task (600 ms), configures the variable DATA2 with the task associated to the event (Task 1) and configures the Associated Action at address RELEASE_START. At the end, the initialization code enables the Event 1 (the event structure of event 1 is inserted in the sorted list of absolute events). The initialization of the rest of release events is similar.

The initialization of the Task Structure of Task 1 could be as follows:

TABLE 4: INITIALIZATION THE TASK STRUCTURE

Code	Description
init task 1	Creates the Event 1 Structure
mov task 1, task_program_code_address, start_addr_task1	Sets the beginning of Task 1's code
mov task 1, task_code_address_aborted, no_code	No action when task is aborted
mov task 1, task_code_address_execution, no_code	No action when task's status changes to Execution
mov task 1, task_code_address_desallocated, no_code	No action when task is preempted
mov task 1, task_code_address_ready, no_code	No action when task's status changes to Ready
mov task 1, task_code_address_finished, no_code	No action when task finishes
mov task 1, task_execution_priority, 60000	Set task's priority to 60000 when task is in execution state
mov task 1, task_ready_priority, 60000	Set task's priority to 60000 when task is in ready state
mov task 1, data2, 60000	Store the deadline of the task (600ms)

The initialization 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. Initialization of every task is similar.

2) Associated actions

The real-time discipline 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 HRTSC. The ETP register hold the index of the event that took place. In this way, the associated action can be parameterized using this register

Each time that an event 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 it is inserted in 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 priority discipline among the task of the system. The Execution Priority of the task is set equal to the Ready Priority to implement a preemptable priority discipline. Otherwise, if a non-preemptable discipline is desired, then the Execution Priority might be set equal to zero. Therefore, the associated action to the events could be as follows:

TABLE 5: ASSOCIATED ACTION FOR RELEASE EVENT

Code	Description
release_start:	Address Label
add etp, occurrence_time, etp, data1	the occurrence time is added to the period of the task stored into the variable DATA1 and the event is enabled again
insert etp in absolute list	
mov a, event etp, data2	The ETP register is loaded with the task associated with the event (stored into variable DATA2)
mov etp, a	
mov b, task etp, data1	Register B of the RTU is added to the deadline of the task (stored into variable DATA1 of the task structure)
add a, b	
mov task etp, task_ready_priority, a	The Ready Priority is set equal to the absolute deadline to performed an EDF discipline as well as the Execution Priority to make it preemptive
mov task etp, task_execution_priority, a	
chstatus task etp, ready	The status of the task is changed to ready
end	The associated action is ended

B. Analysis

From the real-time scheduling theory point of view, the first experiences showed that the overhead of the system is reduced when task are schedule with a HRTSC unit. Moreover, the processor is halted each time that there is not task in ready state and consequently, the power consumption of the system is reduced. The temporal constraints are more flexibly configured since it is not necessary to modify the real-time task code when different priority disciplines are wanted to be implemented.

On the other hand, from the application point of view, the performance is improved when the HRTSC is utilized because the system time should not be considered slotted. In a real-time system based on a RTOS, the slot time may have a greater influence on the control performance than the clock frequency. Moreover, it may be necessary to re-program the real-time tasks when the slot time is changed.

With the HRTSC a high precision can be achieved. With a 10MHz clock, it could be defined every event with an accuracy

of 100ns, which is suitable for most of real-time systems ([19]). When more precision is desired, then the HRTSC clock can be increase which does not interfere with the execution of the NIOS II processor.

VI. CONCLUSIONS

Real-time theory offers diverse scheduling mechanisms with varied real-time features, each on of them 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 a real system are reduced.

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 ([12]), the real-time restrictions originated from the software version implemented in RTOSs remain in hardware. Usually, hardware schedulers migrated from RTOS implement a certain priority discipline 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 priority discipline chosen by the designer of the processor.

In this paper, a Hardware Real-Time Scheduling Coprocessor Unit designed to incorporate real-time properties to NIOS II processor is described. The HRTSC can be configured to implement any priority discipline and support a large number of real-time tasks. The HRTSC is described including the events and task structures to configure the real-time features of the application. The utilization of the Avalon Specification and the BSP interface makes the use of the HRTSC easy and flexible for the SOPC Builder user.

Future works will include the evaluation of the performance for different real-applications and the construction of a SOPC Builder component fully compatible with SOPC Builder specification and able to be incorporated to the NIOS II library. It also has to be implemented an efficient mechanism to halt the NIOS II processor when no task is required to be executed.

REFERENCES

[1] J. A. Stankovic and K. Ramamrithan, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software,* 1991.

[2] D. Hildebrand, "An Architectural Overview of QNX," in *Usenix Workshop on Micro-Kernels & Other Kernel Architectures*, Seattle, 1992, pp. 113-126.

[3] J. J. Labrosse, Ed., *uC/OS-III, The Real-Time Kernel*. Micrium Press, 2009, p.^pp. Pages.

[4] M. Zhou*, et al.*, "Adaptive Hardware Real-Time Task Scheduler of Multi-Core ATPA Environment," in *2009 NASA/ESA CAnference on Adaptive Hardware and Systems*, 2009.

[5] J. Agron*, et al.*, "Run-Time Services for Hybrid CPU/FPGA Systems on Chip," in *27th IEEE International Real-Time Systems Symposium*, 2006.

[6] L. Yan*, et al.*, "Hardware Implementation of uC/OS-II based on FPGA," in *2010 Second International Workshop on Education Technology and Computer Science*, 2010, pp. 825-828.

[7] T. Nakano*, et al.*, "Hardware Implementation of a Real-Time Operating System," in *Twelfth TRON Project International Symposium*, 1995, pp. 34-42.

[8] M. Vetromille*, et al.*, "RTOS Scheduler Implementation in Hardware and Software for Real Time Applications," in *Seventeenth IEEE International Workshop on Rapid System Prototyping*, 2006, pp. 163-168.

[9] A. Aliveira*, et al.*, "The ARPA-MT Embedded SMT Processor and its RTOS Hardware Accelerator," *IEEE Transactions on Industrial Electronics,* vol. 58, pp. 890-905, March 2011 2011.

[10] P. Kuacharoen*, et al.*, "A Configurable Hardware Scheduler for Real-Time Systems," in *International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2003.

[11] Q. Deng*, et al.*, "A Reconfigurable RTOS with HW/SW Co-Scheduling for SOPC," in *2nd International Conference on Embedded Software and Systems*, 2005.

[12] J. Lee*, et al.*, "A Comparison of the RTU Hardware RTOS with a Hardware/Software RTOS," in *ASP-DAC 2003. Design Automation Conference*, 2003, pp. 683-688.

[13] B. O. Gallmeister, *POSIX.4: Programming for the Real World*: O'Reilly & Associates, 1995.

[14] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* vol. 20, pp. 46-61, 1973.

[15] Altera, "Avalon Interface Specifications," Altera, San Jose2011.

[16] V. Glaviníc*, et al.*, "Vhdl-based modeling of a hard real-time task processor," in *IEEE International Symposium on Industrial Electronics*, Bled, Slovenia, 1999, pp. 49–54.

[17] D. V. W.A. Halang M. Colnaric, "Supporting high integrity and behavioural predictability of hard real-time systems," *Informatica, Special Issue on Parallel and Distributed Real-Time Systems,* vol. 19, pp. 59–69, February 1995.

[18] J. Adomat*, et al.*, "RealTime Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems," in *8th Euromicro Workshop on Real Time Systems*, L'Aquila, Italy, 1996, pp. 164-168.

[19] IEEE1003.1d. (1999). *Draft Information Technology - Portable Operating System Interface (POSIX): Part 1 : System Application program interface; Additional Real-Time Extensions. [C Language]*.