Architectural solutions for enhancing the real-time behavior of distributed embedded systems

Carlos E. Pereira
Federal University of Rio Grande do Sul (UFRGS)
Electrical Engineering Department
cpereira@eletro.ufrgs.br

Marcelo Götz
University of Paderborn
Heinz Nixdorf Institut
mgoetz@uni-paderborn.de

Abstract

The paper presents a low cost embedded hardware and software architecture that makes use of COTS components to support distributed real-time embedded systems. The proposed architecture addresses a common problem in conventional architectures: undesirable timing variations in application's temporal behavior due to overload caused by operating system activities when dealing with concurrent and time-triggered processes. The proposed architecture makes use of a 32bits high performance microcontroller and the open source code operating system for embedded applications uClinux and it enhances these with extensions to better cope with distributed real-time systems development.

1. Introduction

The correctness of a real-time system depends not only on the logical results of computation, but also on the timeliness of the sampled inputs and produced results [10], therefore adding complexity to its development. Due to advances in areas such as microcontrollers and software, distributed real-time embedded systems have become very attractive and are widely used in several application domains, such as industrial automation, telecommunication and personal communication (cellular phones, PDAs), process control and home appliances. There is a clear trend in moving the development from a centralized architecture, consisting of few nodes which concentrate all decision making and control processes, to network centric embedded systems, that means, embedded systems that have to present the ability of coordinate their activities with other systems by means of communication and synchronization.

Real-time operating systems (RTOS) are a key component of real-time distributed embedded systems. By handling aspects such as task scheduling, asynchronous event handling, I/O interfacing, the existence of RTOS considerably eases the task of programming concurrent applications, since programmer does not have to explicitly take into account concurrent processing in their programs. However, since RTOS activities usually compete for the same CPUs with other application tasks, they can (and usually do) influence the overall system's performance, leading to a non deterministic temporal behavior.

This article presents a hardware and software architecture for real-time distributed embedded systems, which aims to overcome the problems mentioned above. At hardware level it mainly attacks the problem of temporal determinism through the use of dedicated hardware artifacts using COTS components. At software level, open source and widely adopted operating system for embedded systems, uClinux [2], is adopted and some real-time extensions are proposed. The RTOS scheduling software is partitioned and allocated to a special processor, in order to minimize the overhead caused by execution of RTOS administrative and to achieve a deterministic temporal behavior. This paper is divided as follows: section 2 discusses some related work previously described in the literature. In section 3, the proposed architecture is depicted and its mains components and functionality are presented in section 4. Section 5 presents preliminary results and performance measurements. Finally, section 6 draws some conclusions and signals directions of future research

2. Related Works

Architectural concepts for supporting the goal of achieving a deterministic temporal behavior are described in the literature in ([9, 6, 11]). All have in common the conclusion that a deterministic temporal behavior can only be supported by providing new architectural concepts, since conventional hardware and software architectures are inherently non deterministic with regard to their temporal behavior. In the following sub-sections a brief discussion on the most relevant related work is presented

2.1. Multi-processor Architecture for Embedded Real-time Systems

In a previous work developed at UFRGS by one of the authors and described in [9], a low cost multi-processor architecture based on 8 bits microcontrollers was introduced. Its goal was to provide a processing unit to be used in distributed real-time systems with high level of temporal determinism.

The proposed architecture incorporates three microcontrollers with distinct responsibilities:

- the *Main Processor* is responsible for the management of all time-based activities, such as time-activated dispatch of concurrent processes and timer management. It is also responsible for the handling interrupts and asynchronous signals as well as for the scheduling of concurrent processes;
- the *Communication Processor* deals with all message-based communication processes, coordination and synchronization among tasks. It includes all communication network related activities, such as sending and receiving messages, addressing, routing, etc.
- the *Execution Processor* is responsible for running applications tasks.

The proposed architecture has been successfully applied to some industrial applications in the shoe industry and presented a deterministic temporal behavior. One of the major drawbacks of this work was that it was based on a rudimental embedded operating system developed by the authors, which only included the basic functionalities of task management and communication. The main focus of that work was on an evaluation of the enhancements obtained from the proposed hardware architecture.

2.2. Spring Architecture for Real-Time Systems

The Spring architecture presented in [8] is a set of hardware and software, highly integrated, to build complex real-time systems. One of its more important features is the functional distribution of the system, on that the predictability is enhanced due to isolation of application from the external interrupts. This is reached using distinct processors for the application and for the rest of the system. It was also designed to be used in distributed systems.

The proposed architecture was aimed for high-end applications and suggested the use of optic communication channels and a VLSI device to perform the scheduling activities. The major drawback of the Spring architecture is that most of its tools and APIs were *proprietary* what negatively impacts portability and programming aspects.

2.3. Architectural Support for Predictability in Hard Real Time Systems

The work by Halang and Colnaric, which is described in [6], is a multi-processor architecture consisting of one or more general-purpose microprocessors and one *co-processor* for running operating system tasks. The main goal of the approach was to enhance system's predictability and improve system's performance by minimizing the number of interrupts to operating systems task execution. A programming language was specifically developed for the target hardware.

3. Proposed Architecture

The architecture to be proposed in this paper combines the advantages of an open source and stable operating system and a multi-processor microcontroller-based hardware platform. By combining public available, open source operating system code and low cost microcontrollers, a cost-effective solution can be obtained. The proposed architecture represents an extension of the work presented in [9], aiming a better support in software by adopting COTS and widely used Linux software. Similar to the work in [9], three distinct functional modules are identified - application tasks execution, task management, and communication - but differently those are not mapped to three individual microcontrollers. In the current approach only two processors are necessary, one of them containing a hardware-based communication support.

The *Application processor* has a similar functionality to the *Execution block* presented in [9], and the main activities are: (i) Execution of application tasks; (ii) management of memory associated with each task; (iii) task context switch; (iv) communication with Secondary Block. The *Manager processor* handles file-system, I/O, and timing management (asynchronous signals), as well as scheduling algorithms execution, monitoring of execution time and blocking time (information used by some scheduling algorithms).

To ensure a deterministic temporal behaviour of the uClinux operating system, it is suggested to split its functionality into the two processors. Functions that are frequently used by application programs and do not tend to cause non-deterministic behaviour are co-located in the application processor with application tasks. Management and communication functions, on the other hand, should run in the manager processor. After a carefully study on operating system internals, such as memory location and variables, the proposed separation of the operating system is the physical division between the user-level and kernel-level in the Main Block and Secondary Block. The logical connection between them is provided by a communication channel.

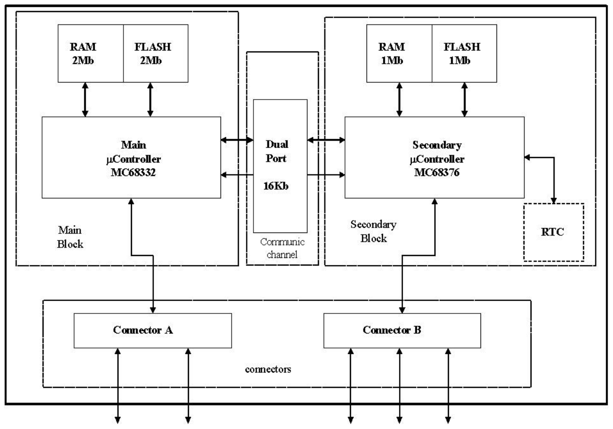

Figure 1. Block diagram from proposed hardware architecture.

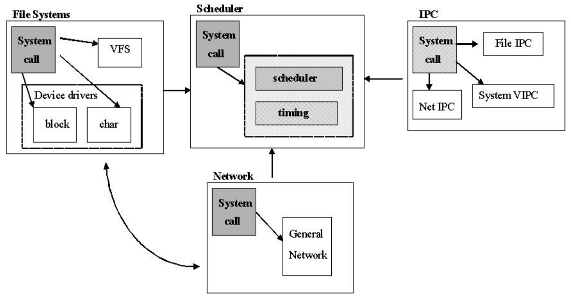

Figure 2. Conceptual architecture for uClinux

4. Implementation (Hardware and Software)

4.1. Hardware

The microcontrollers MC68332 and MC68376 from Motorola were selected to implement the application and manager processor respectively, due to the following reasons:

- Both have powerful programming environments and runtime available in the internet, with free license of use (GPL);
- There are uClinux ports for both microcontrollers, whose source code is also open;
- They are low cost, since it is for general purpose use (COTS components);
- There are versions of the selected microcontrollers which include embedded hardware support for communication protocols such as CAN;
- They have a 32bits architecture;

Figure 1 shows a block diagram of the implemented hardware architecture.

The hardware includes both FLASH and RAM memory. The first holds the binary executable code, and the second one stores variables, stack, data structures and the tasks contexts. Based on the results obtained in [5], following memory size was selected: (i) for the main block: 2Mbytes of RAM and 2Mbytes of FLASH, (ii) for the Secondary Block: 1 Mbytes of RAM and 1Mbytes of FLASH. Both processors can exchange data through a dual-port memory.

For the developed prototype, a microcontroller with a embedded CAN communication module was chosen. The CAN protocol was selected due to its timing characteristics and the possibility of implementing real-time publisher-subscriber concepts [7] on top of it.

4.2. Software

Version 2.0.38 of uClinux was selected in the described implementation due to two main reasons: it was a stable version at the time of the implementation and it included a port for a MC68332 microcontroller successfully used in [5]. The complete operating system code was available on the Internet. The uClinux provides a logical separation between the user level and kernel level using system calls. This leaded to a natural partition of operating system functionality into two blocks, one dealing with kernel related tasks and another one handling application related tasks. Figure 2 depicts the the operating system parts that must be modified.

While the proposed partition should decrease the overhead imposed by operating system management activities on the temporal behavior of application tasks, in order to obtain a fully deterministic temporal behavior, additional real-time extensions to uClinux were proposed: a scheduling strategy similar to the one proposed in RED-Linux [12]. The main advantage of the proposed architecture including two processors is that one of the main overheads imposed by RED-Linux, mainly the insertion of *interruptible points* is avoided, since such code blocks run on the manager processor and therefore do not interfere with the application tasks. RED-Linux was selected after an evaluation among others extensions to Linux [13, 1].

5. Preliminary Results

The hardware and software development occurred simultaneously. Therefore, in order to allow the testing of software modules a MEGA332 board, composed by a MC68332 CPU (described in [5]), was used to execute the Primary Block. The Secondary Block was executed using

	Case A	Case B
Period of T1	$2ms$	$2ms$
Jitter of T1	$140\mu s$ - $280\mu s$	$130\mu s$ - $270\mu s$
Period of T2	$4ms$	$4ms$
Jitter of T2	$120\mu s$ - $200\mu s$	$120\mu s$ - $210\mu s$

Table 1. Measurements of periods and jitters

an emulation software running on a PC machine. This software (called xcopilot) makes the emulation of a board composed by a MC68328 CPU used in the PalmPilot product. As it is open source code, it was possible to implement the Secondary Block on it. The communication channel between the blocks was implemented using the USART in MEGA332 and the USART of PC (mapped by xcopilot).

Timing measurements were done using available output pins of the MEGA332 board (application tasks sent a signal to the output which was captured with a digital oscilloscope). The runtime environment using during the test included two real-time tasks, which were statically created and were ready to start just after system boot. Both tasks were cyclic with a period of 2 and 4 ms respectively.

The Linux scheduler was modified in such a way that real-time tasks had a higher execution priority than non real-time ones (it would run other non real time tasks, only when real-time tasks have finished their execution within a cycle). A Rate Monotonic Algorithm (RMA) was adopted as scheduling strategy. Table 1 shows the time measurements (jitter and cycles) obtained in two different configurations: in Case A only the two real-time tasks are active (additional to the usual operating system management tasks) and in Case B an overhead was created by starting several non real-time tasks. The measurements were obtained with the MEGA332 running at 16MHz and 38400bps speed of USART channel (maximum possible frequency for xcopilot).

The results shows that even with a considerable increase of non critical tasks, the jitter of the real-time tasks were approximately the same. The maximum jitter obtained in the experiments (14%) can be considered good as preliminary result when compared with RTLinux port for the uClinux (referenced in[2]). Additional time measurements performed were related to the OS management tasks running in the Secondary Block. The time interval from the receipt of a scheduler message requesting a context switch until the execution start of the OS function responsible for searching the RT-task to be activated (`Search_for_RT_task`) was measured (row labeled as *latency* in Table 2) as well as the total task context switching. Both measurements are presented in Table 2 for the same scenarios depicted previously (*Case A* and *Case B*). Table 2 also presents the execution time for searching the real-time task to be activated.

	min	max
Latency	$8,5\mu s$	$11,6\mu s$
`Search_for_RT_task`	$41,5\mu s$	$41,5\mu s$
Context switch	$90\mu s$	$90\mu s$

Table 2. Measurements of periods and jitters

6. Conclusions

The work described here is an alternative hardware/software platform which aims to ensure the temporal determinism in distributed embedded real-time systems.The approach makes use of COTS software and hardware components, such as uLinux and low cost microcontrollers. This article also describes the necessity to have a platform that can support more complexity tools for modeling and design real time systems [3, 4].

The obtained results were quite encouraging, since even in a prototype form, the proposed architecture presented a deterministic temporal behavior for different configurations (with regard to an increasing number of application tasks, both real-time as non real-time). Moreover, the proposed platform allows the use of available and well known development tools, like the gcc compiler (with GNU General Public License).

The experiments performed indicated that the performance of the proposed architecture using two microcontrollers for running operating system tasks relies very strongly on the communication channel between the processors. The use of a dual-port memory presents the best trade-off regarding access time and synchronization capabilities. Therefore, the serial communication channel is not suitable to implement the channel for the proposed architecture. It limits the capacity of the system, even more when, for instance, read/write operations on a device driver have to send/receive data to/from user space.

In the proposed architecture the implementation of different and more complicated scheduler algorithms is facilitated, since the impact of its computation load on the application is minimized, since it runs on a separate processor. Additionally, the maneger processor may be used to determine the actual runtime execution and blocking times for all tasks, providing a valuable information to be used by scheduling algorithms.

As a next step in our research, a more complex case study is being developed, on which a robotic system containing two manipulators with 7 degrees of freedom each is going to be controlled by a network of active objects running on the architecture proposed on this paper.

References

[1] Rtai - real-time application interface. URL: http://www.rtai.org, 2001.

[2] Uclinux. the linux/microcontroller project. URL: http://www.uClinux.org, 2001.

[3] L. B. Becker and C. E. Pereira. From design to implementation: Tool support for the development of object-oriented distributed real-time systems. In *12th Euromicro Conference on Real-Time Systems*, Stokholm, Sweeden, June 2000.

[4] L. B. Becker, C. E. Pereira, E. Nett, and M. Gergeleit. An integrated environment for the complete development cycle of an object-oriented distributed real-time system. *Computer Systems Science Engineering*, 1999.

[5] C. Brudna. Desenvolvimento de sistemas de automacao industrial baseado em objetos distribuidos e barramentos can. Master's thesis, CPGEE - UFRGS, 2000.

[6] W. A. Halang and M. Colnaric. Architectural support for predictability in hard real time systems. *Control Engineering Practice*, 1, 1993.

[7] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the cobntroller area network (can). *2nd International Symposium on Object Oriented Real-Time Distributed Computing - ISORC99*, May 1999.

[8] D. Niehaus, E. M. Nahum, J. Stankovic, and K. Ramamritham. Architecture and os support for predictable real-time systems. *5ht International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1992.

[9] C. E. Pereira and M. Pontremoli. Hardware support for distributed real-time operating systems. *IFAC Control Engineering Practice*, 5(10):1435–1442, October 1997.

[10] J. Stankovic. *Misconceptions about real-time computing: a serious problem for next generation systems*, volume 21. October 1988.

[11] J. Stankovic, D. Niehaus, and K. Ramamritham. Springnet: "a scalable architecture for high performance predictable, and distributed real-time computing. *Workshop on Architectural Aspects of Real-Time Systems*, December 1992.

[12] Y. Wang and K. Lin. Enchancing the real-time capability of the linux kernel. *RTSS - Real-Time Systems Symposium*, 1998.

[13] V. Yodaiken. The rt-linux approach to hard real-time. URL: http://www.rtlinux.org/documents/papers/whitepaper.html, October 1997.