IP-SOC 2010

XLUNA: A REAL-TIME, DEPENDABLE KERNEL FOR EMBEDDED SYSTEMS

Giovanni Beltrame[1], Luca Fossati[2], Marco Zulianello[2], Pedro Braga[3], Luis Henriques[3]

[1]**École Polytechnique de Montréal, 900 Boulevard Edouard-Montpetit Montreal, Canada**
giovanni.beltrame@polymtl.ca

[2]**European Space Agency, Keplerlaan 1 - Noordwijk, The Netherlands**
{Luca.Fossati,Marco.Zulianello}@esa.int

[3]**Critical Software, S.A., Parque Industrial de Taveiro, Lote 48, 3045-504 Coimbra, Portugal**
{pbraga,lhenriques}@criticalsoftware.com

Abstract:

While the GNU/Linux Operating System is gaining popularity in research and student communities as well as in the business world, its impact is still limited for all those application areas requiring hard real-time capabilities, extreme robustness, and a minimal trusted computing base. Usually these systems employ hardware that does not yet provide a hypervisor mode, necessary for proper system virtualization. In this paper we present a system, called *xLuna*, composed of three parts: RTEMS, a user-mode version of Linux and a small software layer exposing virtual resources for Linux. We show that, without requiring a processor with paravirtualization support, xLuna is able to provide real-time capabilities, predictability and reliability properties while maintaining the versatility, easy of use and large application base of Linux.

1 Introduction

There is a strong trend from "classical" embedded systems towards more sophisticated and flexible architectures. Considering the tight timing constraints imposed by today's market, such complexity can only be achieved by implementing a consistent part of the system's functionalities as software components.

As a consequence, such devices have peculiar requirements, quite different from those of classical embedded architectures: high demand for processing capabilities, and for a standardized and well-known application programming interface (API). These requirements are well supported by contemporary desktop and server operating systems such as Linux, and it is not surprising that there is a strong trend towards their use also in embedded systems. The disadvantage of these kernels is that they cannot usually guarantee response times for their tasks. Unfortunately, there are application areas (such as robotic devices, healthcare and military computers) where a hard real-time response is of critical importance. Hybrid operating systems, with both real-time and time-sharing components, are starting to appear on the market as a solution to these problems. They have many important benefits such as a comfortable, well-known interface compatible with many existing applications, and the ability to run real-time as well as non real-time applications on the same machine, without impairing the real-time predictability.

Another issue, which limits the use of standard desktop OSes in embedded systems, is that they are often employed in life-critical or mission-critical scenarios. While the reliability of Linux on desktops and servers is generally very high, this typically applies to systems with widely adopted configurations. Massive changes to the system configuration, as it is often necessary for an embedded system, inherently reduce stability and require a significant maturation process. At the same time, the critical parts of the system should not be affected by the other components which are only employed to support non-critical functions. Hybrid operating systems with separate addressing spaces address this problem. The most widely available commercial solution is represented by real-time hypervisors [2]. However, these systems require special virtualization instructions like Intel VTx [7] or special processor modes. Such characteristics are not always available for the kind of embedded processors used in critical (i.e. medical, aerospace) applications.

In this paper we present *xLuna*, a reliable RTEMS/Linux kernel. This work was motivated by the need for a runtime environment targeting real-time and non-real-time applications and providing high reliability, real time behavior and the support for a large application base on hardware not directly supporting paravirtualization. Next to these features, standard programming interfaces (e.g. POSIX), an extensive set of libraries, and a familiar environment, requiring a minimal learning curve for either creating new applications or migrating/integrating existing components, are needed. xLuna requires only a dual mode (user/superuser) processor with MMU: by running the Linux kernel in unprivileged mode, this solution gathers the benefits of using a well known

desktop OS while assuring fault isolation and real-time properties on simple hardware.

Other attempts of combining Linux with a real-time kernel include MaRTE-OS [6], a real-time kernel for embedded applications providing both C and Ada interfaces, coupled with Linux in two different ports. In the first, MaRTE-OS executes as part of the Linux kernel, while in the second it is executed as a user process. Our work goes in the opposite direction in that we integrate Linux inside a RTOS and not viceversa.

In addition to maintaning real-time properties, as shown in the following, our system features complete independence and isolation between the Linux and RTEMS subsystems, something not completely true for the existing approaches (e.g. [5, 6]). This has been achieved by executing Linux as an RTEMS tasks and by implementing tight memory protection mechanisms.

This paper is organized as follows: the xLuna kernel is described in detail in Section 2 and results about its performance are contained in Section 3. Finally, Section 4 concludes the paper and describes possible future developments.

2 xLuna

The solution introduced by xLuna (shown in Figure 1) is based on two sub-systems. The first one provides support for Hard Real-Time (HRT) tasks, the second one provides a well-known programming interface with an extensive set of libraries. The advantage in having two isolated systems consists of the possibility to have applications with different criticality levels running on the same system. Thus, non-critical components are quickly developed or simply ported from a previous implementation, resulting in a reduction of time-to-market and budget. Furthermore, because the HRT system is protected by the Memory Management Unit (MMU) from the Non-Real-Time (NRT) system, a heavy validation and verification test campaign on non-critical components could be reduced or even avoided. This separation allows an approach where the Linux sub-system can be activated and deactivated at any time; moreover, protection from erroneous behaviour of the Linux kernel is ensured. xLuna is currently built to support the LEON 2 processor and it uses Snapgear Embedded Linux [1] as the Linux subsystem.

xLuna, as described in the Section 2.3.1, organizes and manages the memory for the whole system. The 4GB space addressable by the LEON processor is allocated to different devices, including RAM, PROM, bus I/O, on-chip registers, etc. The available RAM space is, in turn, divided into two parts, one for RTEMS kernel and its tasks and the other for the use by Linux kernel and Linux processes. RTEMS occupies the first chunk of memory with a fixed configurable size and the rest is allocated to Linux.

The communication and synchronization of HRT tasks with NRT processes takes place through the Inter-Systems Communication (ISC) module, as described in Section 2.3.4. This module provides

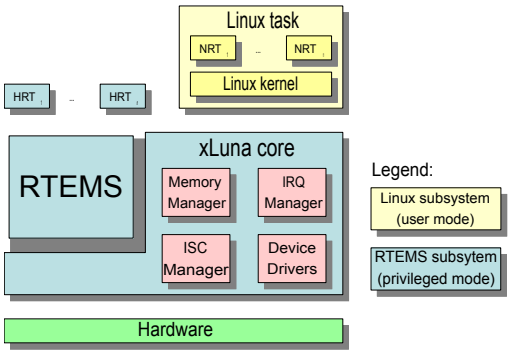

Figure 1: Overview of the xLuna architecture: xLuna and RTEMS are run in the same privileged environment, while Linux is executed in user-mode as the RTEMS idle thread

synchronous/asynchronous bi-directional communication by using message queues. RTEMS tasks interact with it through direct calls to xLuna primitives, while Linux processes read/write to a character device.

2.1 Linux Subsystem

The modifications that have been performed to the Linux kernel source code are reduced to a minimum to ensure portability among different versions. All the implemented patches are mainly due to the fact that, in our system, Linux is not executed on bare hardware and with supervisor privileges. Instead, it runs inside a software shell (xLuna) as a user-mode process. The main challenge of this approach is to maintain, through the soft-virtualization of the underlying hardware, the low-priority Linux system unaware of its execution environment.

In particular, to guarantee the real time properties of RTEMS and to ensure that failures in the Linux kernel do not affect the rest of the system, all direct hardware accesses was replaced with communication events triggering the hardware management primitives of the xLuna and RTEMS subsystems. This way, it is only necessary to make sure that xLuna routines behave as expected in order to guarantee that all operations on the hardware are properly performed. In addition, memory is not directly managed by the Linux kernel but, to achieve memory protection, Linux only accesses the RAM portion which is owned by Linux itself. This can be achieved using the mechanisms described in Section 2.3.1.

As previously introduced, communication among the two worlds takes place using the Inter-Systems Communication module. To allow Linux processes to access to the ISC, a new kernel device was created: user processes can access the interface exported by this driver and easily perform inter-systems communication through regular file operations.

At system startup, the Linux subsystem is not started automatically but it is launched by calling an xLuna primitive inside RTEMS applications. This approach was adopted in order activate Linux only when needed. Furthermore, in case of Linux kernel panic, a previously registered handler can simply

restart the NRT environment.

Apart from these modifications, all the usual functionality and libraries of Linux remain untouched and, from the point of view of the normal application developer, the modifications are unnoticeable. In particular the standard glibc library and the POSIX-Thread interface are available for writing applications. Additional libraries can also be added, as in a normal Linux distribution.

2.1.1 Scheduling Policies

Linux tasks are scheduled by the Linux kernel using its original unmodified scheduler. This because Linux is executed as an RTEMS idle thread: the whole kernel and its processes are phisically executed on the processor only when no HRT tasks are running, as shown by Figure 2.

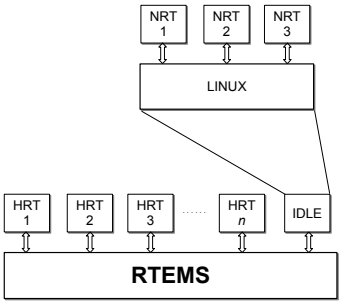

Figure 2: Task scheduling in xLuna

2.2 RTEMS Subsystem

RTEMS (Real-Time Executive for Multiprocessor Systems) [3] is a free open source real-time operating system designed for real-time embedded architectures. RTEMS does not provide any form of memory management or support for processes; in POSIX terminology, it implements a single process in a multithreaded environment. This is reflected in the fact that RTEMS provides nearly all POSIX services other than those which are related to memory mapping, process forking, or shared memory.

RTEMS was chosen as the basic block on top of which the whole xLuna infrastructure is built because it is a reliable operating system with well known efficiency, quality and standard compliance. Its maintainability and robustness as well as its high performance, provide the necessary characteristics for the utilization in critical embedded systems.

In the context of this paper, the development on the RTEMS side focuses on the implementation of new modules that support the execution of the underlying low-priority Linux kernel (more details are contained in Section 2.3). To minimize RTEMS version dependency, the xLuna project avoids modifications to the RTEMS source code whenever possible. Indeed, most of the xLuna features are made available by replacing the interrupt/trap handling routines of the RTEMS kernel, only when explicitly required by a user task. The original RTEMS handlers can also be restored when necessary. This is possible since all the code in the RTEMS subsystem is running in privileged mode. This way, the original RTEMS features are kept intact when Linux is not needed or when the xLuna kernel is switched to a safe mode, where no NRT functionality is available. Furthermore, with the adopted design, the xLuna kernel can support future RTEMS releases with minimum effort.

All RTEMS real-time characteristics are still available, as it kernel was not modified.

2.3 xLuna Subsystem

Figure 3 shows the xLuna subsystem: it is mainly composed of four submodules, managing the interactions between RTEMS and Linux and between these sub-systems and the hardware. These are the *Memory Manager*, the *IRQ Manager*, the *Inter-Systems Communication* (ISC) Manager and the *Device Drivers*. These four modules are the heart of xLuna and they run in privileged mode along with RTEMS.

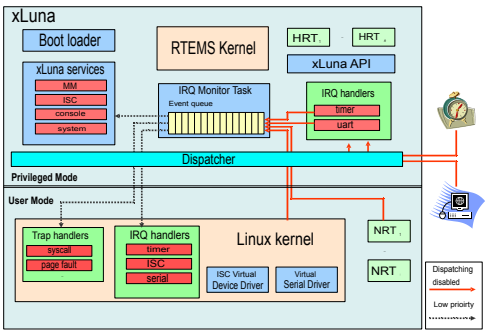

Figure 3: Internal structure of the xLuna sub-system

2.3.1 Memory management

Modern operating systems use, among other architecture-specific approaches, the paging mechanism in order to achieve memory sharing and memory protection. Paging maps virtual memory into physical memory on a page-by-page basis with different access rights for each page. This mechanism usually requires support from the underlying hardware, which in the xLuna case, is the *Memory Management Unit* (MMU) module. In particular xLuna implements a technique called *Supervised Memory Management* using the MMU. This technique allows Linux to manage virtual memory, with every MMU access being monitored by xLuna. This way, memory protection of RTEMS from Linux is guaranteed even in the case of corruption of the Linux kernel. To achieve memory protection, Linux can only access memory which is owned by Linux itself and, in the same way, the RTEMS kernel and its tasks can only access their own RAM portion. The xLuna extensions, running in protected mode, have access to both RTEMS-owned and Linux-owned memory. To this end, the virtual address space is partitioned into three sections:

- An RTEMS window for RTEMS and xLuna extensions to access all the physical RAM, ROM, and I/O addresses.
- A Linux kernel window for the Linux-owned physical RAM and ROM address spaces.
- A Linux process window to be used by Linux processes.

Different access rights for the different memory regions are used to enforce memory protection. The RTEMS window is always marked for kernel mode access, so that neither Linux processes nor the Linux kernel can have access to it, while the Linux kernel window is marked as invalid when a Linux process is running, protecting the Linux kernel from its own processes. The invalid bit is restored when a trap/interrupt occurs. This way the Linux kernel, even though running with user mode privileges, still has the usual protection from user processes. Note how the Linux process window is always accessible from all the sub-systems.

Figure 4 shows the memory map and the ownership of each memory region. The memory region visible in the RTEMS-owned window contains all of the memory-mapped physical resources. The RTEMS kernel and its tasks use this address space to access the RTEMS-owned RAM portion and all the other hardware resources, as if they were running in an environment without MMU (RTEMS is designed for such an environment). The RTEMS window is also used by the xLuna extensions to access the Linux-owned memory portion. The Linux kernel window is mapped to all the Linux-owned memory address space but it does not allow access to any other memory region. This design protects the global address space from Linux so that this sub-system is unable to access physical devices directly. The Linux process window is used by the Linux kernel to allocate virtual address space for its processes, and each of them sees a different memory window as normal.

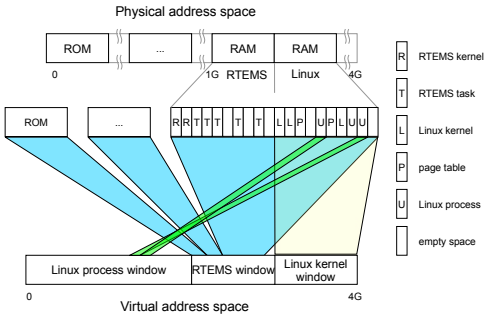

Figure 4: Detailed mapping from virtual address space to physical address space

An important factor in the memory management design is the placement and management of the page tables, which implements the above address space division and access rights management. Allowing Linux to manage all page tables might compromise system integrity as uncontrolled changes could make any physical memory space present in the hardware platform visible. This means that without employing any privileged CPU instruction, failures in the Linux kernel can lead to memory corruption in the RTEMS space. However, it is also not desirable to port the whole Linux memory management code into RTEMS, as the page table management logic in Linux is very complex and it is highly coupled with other Linux kernel functions like file system and task scheduling, The approach proposed here is to let Linux manage page tables, but under the supervision of RTEMS. In this approach, Linux memory management logic remains unchanged, but the memory containing page tables is made read-only to Linux. Low-level page table interfaces, that change the virtual to physical memory mapping, are reimplemented to invoke xLuna services that verify the validity of the modifications before applying them.

2.3.2 Interrupt handling

As previously described, Linux runs as an RTEMS regular task with low privileges and no direct connection to the underlying hardware. This situation is contrary to the basis with which that OS is designed and it introduces some difficulties, mainly catching hardware interrupts and traps generated by user processeses.

The xLuna kernel provides *bridging services* in order to connect the Linux interrupt handlers with the physical interrupt/trap sources. Since during interrupt and trap handling the dispatch of Hard Real-Time Tasks is disabled, the amount of time spent for this process is kept to a minimum. As a result, only a minimal amount of work is done inside the interrupt service routine of xLuna. The actual work is deferred and carried out later by a low priority task called Dispatcher.

More in detail, the main features provided by the *bridging service* module are: (a) managing of the incoming hardware interrupts by intercepting them and by pipelining them to the corresponding RTEMS handlers; (b) catching of all traps from the Linux subsystem and redirecting to the correct handlers; (c) allowing Linux kernel to virtually disable interrupts during its execution (in this case xLuna will not deliver the disabled interrupt to the Linux subsystem); (d) managing time synchronization between RTEMS and the Linux subsystem; and (e) providing an RTEMS API enabling the management of the Linux subsystem from RTEMS tasks.

All illegal traps from the NRT system (Linux kernel and applications) are handled by xLuna kernel so that HRT system remains protected. Hardware interrupts are first processed by xLuna and all interrupts for the Linux kernel are queued and passed to it later on.

2.3.3 Device drivers and I/O

The device drivers module allows Linux to access additional hardware that is not managed neither by the memory manager, nor by the IRQ manager. For now,

they only provide access to timer, UART, and ethernet. These devices are shared between RTEMS and Linux, which go through xLuna in order to access them.

Hardware devices can also be exclusively assigned to Linux using one of two mechanisms: (a) direct access to the device address space in case they are passive devices and (b) management through xLuna for active devices. This means that xLuna can control the access to specific devices, preventing Linux from seizing resources needed for RT tasks, or preempting it when RT requests arrive.

2.3.4 Inter-Systems Communication

In xLuna, communication between HRT and NRT tasks is possible through a message queue, as depicted in Figure 5. This message queue is implemented by the Inter-Systems Communication (ISC) manager through an RTEMS message queue. It provides synchronous and asynchronous bi-directional communication. Communication on the RTEMS side is implemented directly using message queue primitives. On the Linux side, a kernel device driver is provided for inter-systems communication, which uses system calls to access the xLuna kernel services, that in turn, call the RTEMS message queue management routines. No direct access to the queue is allowed from Linux to avoid corruption of the RTEMS side of the system. Linux user processes can access the interface exported by the kernel device driver and perform inter-systems communication through regular file operations.

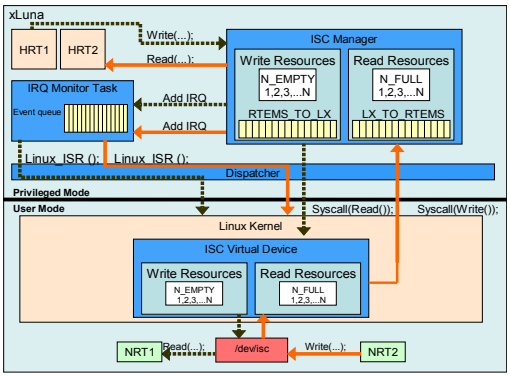

Figure 5: Detailed architecture of the Inter-System Communication mechanism

As Linux is regarded as a task of RTEMS, blocking a Linux process waiting for some message from RTEMS could naively block the whole Linux kernel from rescheduling. This is certainly not a desired behaviour since there might be other Linux processes ready to run. In xLuna, system calls from the Linux subsystem to read/write messages are only issued when resources for these operations are actually available.

To enforce isolation between sub-systems, a specific message queue system was developed and implemented on the ISC module. The communication resources are placed in the RTEMS memory space, the Linux kernel can only access them indirectly through system calls, guaranteeing higher reliability and clear separation among the HRT and NRT subsystems.

2.4 Putting It All Together

Deploying such a complex system on the target hardware architecture is not a simple task and it requires particular steps to be taken, as shown in Figure 6. The RTEMS and Linux kernel images are created separately and loaded in different physical memory areas. Then, the binary image of the final xLuna system is generated by linking these two kernel images together with a small bootstrapper. To simplify the implementation of the memory protection mechanisms, RTEMS and Linux run in separate physical address spaces. Nevertheless this separation is not mandatory, since such protection, as explained in Section 2.3.1, could be achieved by only using MMU. The implication of this strategy is that the memory areas for Linux and RTEMS have to be defined at compile time, and cannot be subsequently modified. This limitation is not of great impact on embedded real-time applications, where resources are known and distributed in advance, during the design of the system.

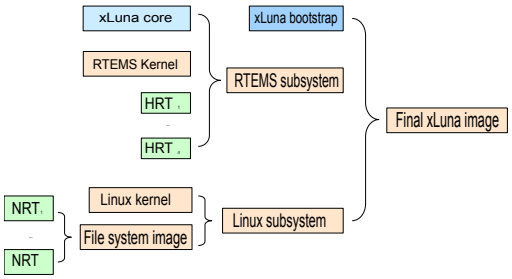

Figure 6: Linking order of the different sub-systems in order to produce the executable image of the whole architecture

The bootstrapper is the entry point to the entire xLuna system. Its purpose is mainly to define the initial MMU mapping, i.e. the RTEMS window and the Linux kernel window. After switching on the MMU, it jumps to the virtual memory space of the RTEMS entry point and gives control to the RTEMS kernel. The raw physical addresses are used only at the very beginning of the booting process, then the virtual addresses of the RTEMS window are considered.

The bootstrap process depends on how the final xLuna image is linked together. Figure 6 gives an idea of the different stages that are used to produce the final image. During linking, RTEMS is relocated to the virtual address in the RTEMS window, and similarly, the Linux subsystem is relocated to the virtual address of the Linux window. After bootstrapping, the two operating systems are able to run a virtually unlimited number of tasks: new RTEMS (RT) or Linux (NRT) tasks can be created at runtime using the services provided by the two operating systems according to constraints and available resources.

3 Experimental Results

In this Section we show some performance measures on the xLuna system. In particular we demonstrate how the real-time responsiveness of RTEMS is not affected by the Linux subsystem. Since memory is a precious resource in embedded systems, we also present the system's memory requirements. The system's performance was measured using TSIM [4] configured as a LEON 2 processor running at 50 MHz, 4-way set associative cache with 16 KB per way, UART, Timer, and an AHB bus connecting all the components.

Figure 7 shows the interrupt response time of RTEMS as the number of either Linux or RTEMS tasks running in the system increases. In our experiments, Linux executes a benchmark (the `queens` problem) and some mathematical tasks, while RTEMS performs some floating point operations interleaved with random length wait states. The measurements were taken with an increasing number of RTEMS tasks and a constant the number of Linux tasks and viceversa. Interrupts were generated at random instants during simulation. As shown, xLuna

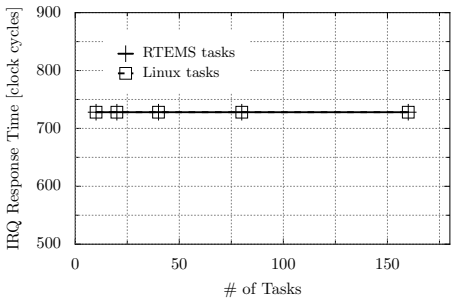

Figure 7: Interrupt response latency of the RTEMS sub-system with respect to the number of tasks in the system

does not affect the real-time behavior of RTEMS: the interrupt response time remains constant and independent of the number of tasks running on the system.

Figure 8 shows the runtime of the `queens` Linux benchmark, as measured while varying the number of both NRT and HRT tasks. As expected, the execution time scales linearly with the number of Linux tasks with the same priority: the more the tasks, the less the processing power allocated to each of them (so less cpu-time allocated to the `queens` benchmark). On the other hand, when increasing the number of RTEMS tasks, the execution time tends to saturate. This is due to the fact that each RTEMS task was designed, for this particular experiment, in a way to interleave idle time with its execution (as normal in a real-time system): and exceeding a certain number of tasks, the cumulative idle time is sufficient to allow termintating the queens benchmark. This proves that the performance of Linux tasks is not excessively affected even when running a large number of real-time tasks.

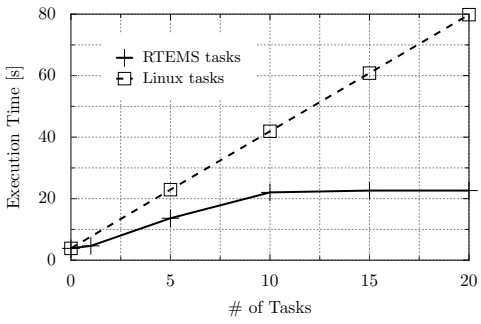

Figure 8: Execution time of the NRT queens benchmark with respect to the number of tasks in the system

The minimum space that must be allocated to hold the bootstrapper, xLuna core, RTEMS kernel and HRT tasks amounts to 256 KB. For this reason, to maximize the TLB performance, we used 256KB pages. Adding any HRT task to the "empty" system requires at least a new page, i.e. additional 256KB. Besides the Linux kernel and global data, the Linux image holds a file-system containing the NRT tasks, along with any other programs and libraries needed by the tasks themselves. The plain image without any NRT task is 1.1 MB , 1 MB being occupied by the kernel code and 87 KB by global data.

4 Conclusion

In this paper we presented xLuna, a software architecture featuring a portable user-mode version of the Linux OS, running on top of the Real-Time Operating System (RTOS) RTEMS. xLuna can run on systems without paravirtualization support, providing a seamless integration between non-real-time and real-time tasks.

Results show that xLuna does not affect RTEMS real time behaviour and interrupt response time, and that Linux tasks are not excessively slowed down even when running concurrently with a large number of RT tasks.

Future work include porting xLuna to other architectures, and the development of the increase of more flexible and efficient communication among the two subsystems.

References

[1] SnapGear Embedded Linux Distribution, http://www.snapgear.org/.

[2] D. Chisnall. *The definitive guide to the xen hypervisor*. Prentice Hall Press, 2007.

[3] A. Colin and I. Puaut. Worst-case execution time analysis of the RTEMS real-time operating system. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 191–198, 2001.

[4] Gaisler Research. TSIM, http://www.gaisler.com/.

[5] B. Leslie, C. Van Schaik, and G. Heiser. Wombat: A portable user-mode linux for embedded systems. 2005.

[6] Masmano, Real, Ripoll, and Crespo. *Extending the Capabilities of Real-Time Applications by Combining MaRTE-OS and Linux*, pages 144–155. 2004.

[7] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38:48–56, 2005.