ARINC-653 APEX based on XtratuM

M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll and A. Crespo

Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
{mmasmano, yvaliente, pbalbastre, iripoll, alfons}@aii.upv.es

Abstract. The ARINC-653 standard defines a general-purpose APEX (APplication/EXecutive) interface between the Operating System (O/S) of an avionics computer resource and the application software. This interface provides the services to build partitioned systems and has been used successfully in avionic systems. In recent years, the space domain is considering the incorporation of Time and Space partitioning (TSP) based upon the Integrated Modular Avionics (IMA) concept. This concept is fully supported by ARINC-653 and it is taken as the reference standard. XtratuM is a hypervisor for real-time systems that has been designed following the safety and security criteria for partitioned systems. XtratuM provides a virtual machine to execute partitions that have been para-virtualised. In this paper, we present LithOS, a real-time guest operating system, which executes as a XtratuM partition, that provides the ARINC-653 API. We describe the services provided by LithOS and some additional features that have been included in the API. Finally, an evaluation of LithOS is shown, comparing the performances of LithOS with other execution environments.

Keywords: ARINC-653, RTOS, partitioned systems, para-virtualisation.

1 Introduction

The technological innovation in the computer industry has driven to the availability of new high-speed processors, an increasing computing power, memory sizes and low cost on-chip memory. Accelerated computing is a framework that drives to use these resources in the integration of many applications, hence the interest to enable multiple applications, to share a single processor and memory, protected in time and space. In order to fully exploit the performance improvements of modern processors in safety-critical applications, it is advantageous to enable the integration of applications at multiple levels of critically and security on the same processing resource.

Partitioned software architectures represent the future of secure systems. They have evolved to fulfill security and avionics requirements where predictability is extremely important. The separation kernel proposed in [9] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference.

The MILS (Multiple Independent Levels of Security and Safety) initiative is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The technical foundation

adopted for the so-called MILS architecture is a separation kernel. Also, the ARINC-653 [1] standard uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture. The IMA partitioning concept emerges for protection and separation among applications from the spatial and temporal point of views. Spatial isolation protects the memory of a partition. A partition cannot acces memory out of the scope of its own memory specified on partition configuration. Temporal isolation means only one application at a point of time has acces to the system resources, whereas is not possible to an application run when another application is running.

The closed requirements in aeronautical industry with the space industry has done the ARINC-653 standard a good candidate to solve the problems present in the aerospace sector. It provides a standard environment independent from the implementation and the underlying hardware and offers to the developers a clean and portable interface to develop secure applications [10]. The importance of virtualisation is growing every day, particularly it is gaining considerable interest in the embedded domain. Virtual machine (or hypervisor) technology can be consider the most secure and efficient way to build partitioned systems. XtratuM is a hypervisor which allows to execute several applications according to the Integrated Modular Avionics(IMA) concept. XtratuM implements a partitioned based architecture which provides protection to safety critical applications. The increased importance of the ARINC-653 and the IMA concept prompts for a replacement of the non-standard XtratuM's API with a standard specification.

This paper presents the architecture of LithOS, a new real-time operating system for developing partition applications on top of XtratuM, which provides an Application Interface(API) compliant with the ARINC-653 specification.

2 The ARINC-653 specification

ARINC-653 specification[1] provides a standarized interface between the OS within IMA and the application software which specifies the interface and the behavior of the API services but leaves implementation details to OS developers. In this way, the Application Execution(APEX) not only standarizes the definition of services, but also the interface of the underlying OS. Therefore, the ARINC-653 specification defines an independent interface of hardware and Operating System(OS) and provides significant benefits: portability, reusability, modularity and integration of software building blocks.

One key aspect is that all resources have to be clearly specified at the building time by means of a configuration file (CF). This CF specifies the number of partitions, memory allocation of partitions, partition schedule, ports and channels, etc.

2.1 Services

The ARINC-653 specification describes the complete set of services. This list of services identifies the minimum functionality provided by the OS to the applications and they are defined in the ARINC-653 Part 1:

- Partition management: The main concept of the ARINC-653 is partitioning. A partition is an execution environment with separate memory space and strictly protected time, without affecting another partitions on any way, according to the IMA architecture. Partitions are scheduled according to a cyclic scheduler which is specified in the configuration file. All resources used by a partition (processes, blackboards, semaphores, ports, ...) have to be defined at system configuration time and created and initialised during the initialisation phase of the partition.
- Process management: A partition comprises one or more processes that interact dinamically to provide the partition functionality. Processes are the execution unit within a partition of ARINC-653. The scheduler works according a fixed-priority preemptive policy. A process with an higher current priority can preempt the running process. These services permit to manage the processes in the partition in a way that satisfies the requirements of the application. Processes are only visible inside its partition.
- Time management: is the basic module to manage time in the OS and ensures that hard real-time requirements are met. Time management module uses the hardware timers to read the current time and provide the time requests. An application may request a time-out, delay, periodicity, process scheduling, etc.
- Inter-partition communication: This module defines the communication mechanism between two or more partitions via messages. A port allows a specific partition to write and read messages from a channel, between a source and a destination port, specified in the configuration data. Channels, ports, maximum message size and maximum number of messages are completely defined at system configuration time. These services include Sampling Port and Queuing Ports.
- Intra-partition communication: These services dene the mechanisms used for communication and synchronization between processes within the same partition. Blackboards and buffers are provided for intra-partition communication. Semaphores and events are provided for intra-partition synchronization.
- Health monitoring: the health monitor is the mechanism proposed by the ARINC-653 for reporting and monitoring errors. The error handling is the highest priority process and it is invoked whenever a fault takes place. The health monitor may ignore the fault and log it or call the error handler to manage the error which defines how the partition should respond.

The ARINC-653 specification Part 2 [3] defines several additional services as extended. One of these services, developed in Lithos and defined in the standard as Multiple Module Scheduler, is related to the ability to extend the single static module schedule by several scheduling plans defined in the configuration file and the posibility to change the current scheduling plan.

3 XtratuM

XtratuM is a hypervisor for real-time embedded systems that provides virtualised services to the partitions and manages their execution environment [4, 5, 7, 8, 6]. XtratuM virtualises the essential hardware devices (memory, timers and interrupts) to execute concurrently several OSes, such as LithOS.

As we have seen, the ARINC-653 standard is based on a partitioned architecture. XtratuM provides the partition concept as an execution environment virtualised to be executed on top of the hypervisor. Partition developement on top of XtratuM requires to write the code to be executed inside of the partition. The hypervisor takes control of the system at boot time and initialises the hardware, then the partition code is started. This partition code can be:

- An application compiled to be executed on a bare-machine.
- A real-time operating system and its applications.
- A general purpose operating system and its applications.

XtratuM is not a standard hypervisor but some parts are very closed to the functionalities defined in the ARINC-653 specification. The hypervisor provides the ARINC-653 partition management, inter-partition communications, health monitor, scheduling policy and other functionalities to accurately been adapted to the

3.1 Execution environments

Partitions can be executed directly on top of XtratuM by means of a minimal layer called XAL (XtratuM Abstraction Layer). XAL is a minimal partition development support for the development of C programs directly on top of XtratuM. This abstraction layer provides the basic and minimal services to setup a basic "C" execution environment. XAL is only useful for those partitions that are mono-thread and do not need an operating systems. All services provided the XtratuM hypercalls are available to the application. The XAL development environment is integrated by a library with the services and the minimal runtime to execute the partition and handle the virtualised interrupts.

In addition, XtratuM also supports RTEMS operating system. RTEMS is a free open source real-time operating system (RTOS) designed for embedded systems and adopted by the ESA for space applications and space missions. RTEMS (4.8.1) has been ported on top of XtratuM.

4 Lithos Overview

LithOS is a para-virtualised guest operating system which provides the primitives to create the system resources (blackboards, buffers, events, semaphores...) and the mechanisms to create threads, timers and the process scheduler. This is a non-standard interface designed for the efficient and accurate development of standards which define the OS personality.

LithOS implements the process concept presented in the ARINC-653 standard which is not present in XtratuM. Processes may operate concurrently in order to satisfy the application requirements. LithOS adds the multi-process support, the communication between processes and the process scheduler. LithOS uses the services provided by XtratuM to complete the mechanisms required to develop application based on ARINC-653.

The LithOS architecture supports partitioning in accordance with the IMA philosophy. Spatial partitioning is ensured by the partitions with its own data and context, and

by the configuration file which defines each partition memory area. Temporal partitioning is ensured by a cyclic priority scheduler which is periodically repeated. The order of execution is defined statically in the configuration file.

4.1 Lithos Architecture

The LithOS architecture is shown in Figure **??**. The XtratuM layer provides an execution environment which furnish a set of services, such as partition management, time management, inter-partition communication and health monitor.

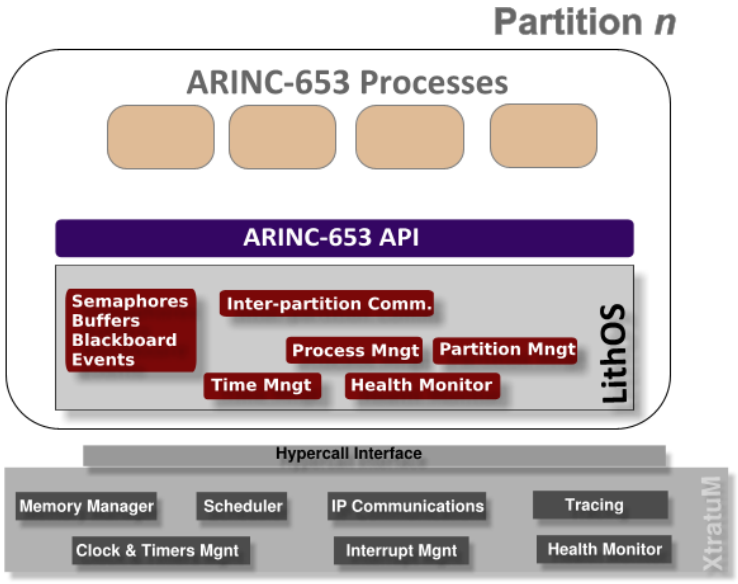

Fig. 1. Lithos architecture

LithOS implements the services to provide multiprocessing and internal mechanisms for synchonisation and communication among processes. The services related to partition management and interpartition communication are built from the basic services provided by XtratuM. Health monitoning and tracing facilities are implemented using the XtratuM services.

These are the services implemented by LithOS.

Partition management The standard defines basic services related to the partition such as set the partition mode or get the partition status. The set partition mode permits to restart the current partition (COLD or WARM RESET). The standard does not defines a partition identification allowing the services to manage other partitions (i.e. reset, start, stop, shutdown). In addition, the access to the status of other partitions is not considered either.

Process management These services are entirely implemented by LithOS to offer complete the ARINC-653 specification. So, a LithOS partition can create a set of processes (tasks or threads in other nomenclature) that permit to design multi-process applications. There is a clear differentiation between the partition initialisation phase (COLD or WARM RESET state) and the execution phase (NORMAL state). System resources (processes, ports, blackboards, ...) can only be created during the initialisation phase.

Time management LithOS uses the basic services provided by XtratuM to retrieve the current time or arm a timer. Internally, LithOS implements a timer data structure (heap structure) to build as many timers as needed by the application. The clock granularity is 1 nanosecond.

Inter-partition communication The inter-partition communication mechanisms (ports and channels) defined in XtratuM were inspired in the ARINC-653 specification. So, these services are used directly by LithOS' internal processes.

Intra-partition communication LithOS layer implements the services to communicate and synchronise processes. Inter-process communication is conducted via *buffers* and *blackboards* which can support the communication of a single message type between multiple source and destination processes. *Buffers* are stored using a queue discipline whereas *blackboards* only store a message. Inter-process synchronisation is conducted by using *semaphores* and *events*. *Semaphores* are counting semaphores and are used to control the access to shared resources. *Events* are synchonisation mechanisms which allow notification of an occurrence of a condition to processes which may wait for it. An *event* is composed of a bi-valued state variable (up and down).

Health monitoring (HM) XtratuM defines a HM service inspired in the ARINC-653 HM. Through the configuration file, the predefined HM_events are associated to predefined services. Some of them can be *propagated* to the faulty partition. LithOS provide the services to install an exception handler process which is in charge of manage the raised exceptions. Application based exceptions can be defined, raised and managed using the services provided by LithOS.

Table 4.1 summarizes LithOS services.

4.2 Extended services

The ARINC-653 standard defines several additional services as extended. One of these services is related to the ability to define several scheduling plans in the conguration file and the posibility to change the current scheduling plan. The services related to this functionality are defined in the standard as Multiple Module Schedule (MMS).

LithOS implements the MMS that permits to extend the single and static module scheduler to several scheduling plans. Plans are identified by means of a plan identifier. When XtratuM starts the partition execution, the plan identified as "0" is set as the current plan. A partition with the appropriated rights can request a plan change which, if accepted, is effective at the end of the current major frame (MAF).

The system architect can define as many plans as needed. A partition with the appropriated rights is in charge of conduct the system to the plan needed at each moment.

Partition management	**Health monitoring**
GET_PARTITION_STATUS	REPORT_APPLICATION_MESSAGE
SET_PARTITION_MODE	CREATE_ERROR_HANDLER
Process management	GET_ERROR_STATUS
CREATE_PROCESS	RAISE_APPLICATION_ERROR
SET_PRIORITY	**Blackboard management**
SUSPEND_SELF	CREATE_BLACKBOARD
SUSPEND	DISPLAY_BLACKBOARD
RESUME	READ_BLACKBOARD
STOP_SELF	CLEAR_BLACKBOARD
STOP	GET_BLACKBOARD_ID
START	GET_BLACKBOARD_STATUS
DELAYED_START	**Buffer management**
LOCK_PREEMPTION	CREATE_BUFFER
UNLOCK_PREEMPTION	SEND_BUFFER
GET_MY_ID	RECEIVE_BUFFER
GET_PROCESS_ID	GET_BUFFER_ID
GET_PROCESS_STATUS	GET_BUFFER_STATUS
Time management	**Event management**
TIMED_WAIT	CREATE_EVENT
PERIODIC_WAIT	SET_EVENT
GET_TIME	RESET_EVENT
REPLENISH	WAIT_EVENT
Inter-partition communication	GET_EVENT_ID
CREATE_SAMPLING_PORT	GET_EVENT_STATUS
WRITE_SAMPLING_MESSAGE	**Semaphore management**
READ_SAMPLING_MESSAGE	CREATE_SEMAPHORE
GET_SAMPLING_PORT_ID	WAIT_SEMAPHORE
GET_SAMPLING_PORT_STATUS	SIGNAL_SEMAPHORE
CREATE_QUEUING_PORT	GET_SEMAPHORE_ID
SEND_QUEUING_MESSAGE	GET_SEMAPHORE_STATUS
RECEIVE_QUEUING_MESSAGE	**Multiple schedule**
GET_QUEUING_PORT_ID	SET_MODULE_SCHEDULE
GET_QUEUING_PORT_STATUS	GET_MODULE_SCHEDULE_STATUS

Table 1. Lithos Services

Initially, plans are related to the system modes. Plan 0 is defined as the initialisation schedule plan. In this mode, partitions are initialised and the internal resources are created and initialised. One of the partitions with *system* attributes is in charge of the mode change requests. Plan 1 is considered as Maintenance mode. By default, this is the plan executed when a health monitor event selects as action a mode change. Plan 2 and next ones are operational modes. The system architect can define as many modes as needed for the system operation. Figure 2 shows the relation of the schedule plan management and the partition status, while Listing 1.1 shows an example of configuration of a multiple schedule definition.

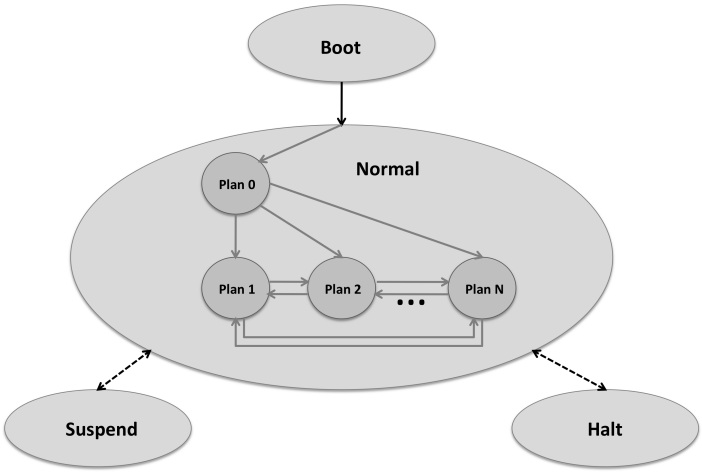

Fig. 2. LithOS Multiple schedule.

4.3 Non-portable services

ARINC-653 lacks of some services that can be considered relevant for partitioned systems. In order to cover these services, LithOS defines a set of non-portable services which are not included in the standard. These services are mainly related to partition and time management.

– **Partition management**
  - GET_PARTITION_ID_SELF_NP : Provides information about the partition identifier. System partitions.
  - GET_PARTITION_INFO_NP : Provides information about the state of other partitions. System partitions.

Listing 1.1. XML code of a multiple schedule definition

```
<CyclicPlanTable>
  <Plan id="0" majorFrame="80ms">
        <Slot id="1" start ="20ms"
              duration ="20ms" partitionId ="0"/>
        <Slot id="2" start ="40ms"
              duration ="20ms" partitionId ="1"/>
        <Slot id="3" start ="60ms"
              duration ="20ms" partitionId ="2"/>
  </Plan>
  <Plan id="1" majorFrame="200ms">
        <Slot id="0" start ="0ms"
              duration ="10ms" partitionId ="0"/>
        <Slot id="1" start ="10ms"
              duration ="150ms" partitionId ="3"/>
  </Plan>
  <Plan id="2" majorFrame="100ms">
        <Slot id="0" start ="0ms"
              duration ="10ms" partitionId ="0"/>
        <Slot id="1" start ="20ms"
              duration ="30ms" partitionId ="1"/>
        <Slot id="2" start ="50ms"
              duration ="40ms" partitionId ="2"/>
  </Plan>
  <Plan id="3" majorFrame="20ms">
        <Slot id="0" start ="0ms"
              duration ="5ms" partitionId ="0"/>
        <Slot id="1" start ="10ms"
              duration ="10ms" partitionId ="1"/>
  </Plan>
</CyclicPlanTable>
```

  - SUSPEND_PARTITION_NP : Suspends the execution of a partition. System partitions.
  - RESUME_PARTITION_NP : Resumes the execution of a suspended partition. System partitions.
  - STOP_PARTITION_NP : Stops the execution of a partition. System partitions.
  - RESET_PARTITION_NP : Performs a reset (cold or warm) of other partition. System partitions.
  - SHUTDOWN_PARTITION_NP : Requests for the shutdown of a partition. System partitions.
  - GET_SLOT_STATUS_NP : Provides the current slot information: slot identifier, duration, slot attributes. All partitions.

– **Time management**
  - TIMED_ABS_WAIT_NP : Allows the process to suspend itself until the specified time

- GET_EXEC_CLOCK_NP : Obtains the local clock of the partition
- SECONDS : Returns a time variable with a specified number of seconds
- MILLISECONDS : Returns a time variable with a specified number of milliseconds
- MICROSECONDS : Returns a time variable with a specified number of microseconds

4.4 Memory model

XtratuM builds a flat memory map for each LithOS partition. The address map of the partition is specified in the configuration file. At build time, the amount of memory space required by the partition as well as the partition resources are allocated from the partitions memory. Associated to each LithOS partition, there is a local configuration file which specifies the maximum number of local resources allocated to the partition: number of processes, blackboards, semaphores, events, buffers and the maximum sizes of messages and maximum number of messages to be sorted (buffers).

4.5 Partition scheduling

Partitions are scheduled under a multi-plan schedule. Each plan schedule is a sequence of slots which details the slot identifier, partition, slot attributes, offset with respect to the MAF origin and slot duration. When a LithOS partition is scheduled, the internal process scheduling is applied. This second scheduling level is a fixed priority scheduling policy as defined in the ARINC-653 standard.

5 Evaluation

LithOS has been validated according to the coverage of the official specification [2]. The ARINC-653 specification Part 1 defines the mandatory services and describes the invocation of those services and the data structures. The LithOS tests have been defined to prove the interface behavior is in compliance with the ARINC-653 specification.

5.1 Conformance tests

Conformance or functional tests are specification-based and are designed to analyse the specification and the behavior of Lithos. The scope of the conformance tests is to demonstrate compliance of the API behavior and determine whether the system meets with the ARINC-653 standard. A total of 625 conformance tests have been implemented. These tests can be grouped according to the functionality in:

– Behaviour tests: These tests, also called stress tests, are oriented to analyse the behavior of Lithos when a system resource exceeds the system limit creation and validates the OS system response under this situation. These tests determine the system robustness in terms of extreme load and determine the performance if the current load goes well above the expected maximum.

– Definitions tests: These tests are designed to check the libraries, services and attributes provided by the API (Application Programming Interface). The variables defined in Lithos are instantiated to every possible value.
– Interfaces tests: These tests are designed to judge the operation of a system under normal and abnormal conditions, which will be defined, and make sure the results are the expected in order to affirm the correctness of implementation. These procedures evaluate features like the ability of the OS to catch errors and the reaction under a specific error condition(return code and error handler). These tests contain intentionally injected errors to simulate error situations and prove if their behaviour is the expected by the specification.
– ARINC-653 Part 3: A standard specification for compliance test procedures to demonstrate and to prove that the interface behavior is in compliance with the ARINC 653 specification. Any application, which will be installed upon LithOS, can rely on this compliance and the portability of applications is more supported.

5.2 Performance tests

Performance or non-functional tests measure the quality of the system, such as overhead or performance. A complete performance evaluation of LithOS has been carried out by using as target the LEON3 processor at 50 MHz. Table 5.2 shows some of these measures. All time measures are in micro-seconds.

Service	Avg	Min	Max	SDev
Process context switch	11.4	11	12	0.36
GET_PARTITION_STATUS	5.26	5	6	0.26
SET_PARTITION_MODE	12.58	12	13	0.04
CREATE_PROCESS	89.20	88	90	2.54
CREATE_SEMAPHORE	16.25	15	17	0.11
CREATE_BUFFER	18.00	18	18	0.02
CREATE_BLACKBOARD	15.75	15	17	0.27
LOCK_PREEMPTION	11.00	10	12	0.12
UNLOCK_PREEMPTION	24.00	23	25	0.18
DISPLAY_BLACKBOARD(16b)	58.50	58	59	0.11
DISPLAY_BLACKBOARD(64b)	70.25	70	71	0.16
READ_BLACKBOARD(16b)	13.00	12	14	0.22
READ_BLACKBOARD(64b)	19.50	19	20	0.11
SEND_BUFFER(16b)	36.50	36	37	0.13
SEND_BUFFER(64b)	48.00	47	49	0.14
RECEIVE_BUFFER(16b)	44.50	44	45	0.13
RECEIVE_BUFFER(64b)	51.00	50	52	0.09
WAIT_SEMAPHORE	10.50	10	11	0.22

Table 2. LithOS service measurements

Inter-partition communication using ports and channels are not shown due to that these services are performed by XtratuM. In [6] the reader can find these measurements.

5.3 Comparison with RTEMS and XAL

In this section we perform a performance evaluation of LithOS compared with other two execution environments: XAL and RTEMS. It is used as reference the number of operation performed by a C program running on native LEON2 processor. This reference permits to compare the number of operations executed by the same scenario built as a bare-C/XAL partition, a RTEMS application and a LithOS application. The comparison allows to determine the performance loss in each execution environment.

The evaluation scenario for the RTEMS and LithOS consists of three tasks or processes (in ARINC -653 terminology):

- *Counter* task: a background, low priority task that continously increases a counter. The counter value is global and can be read by other tasks.
- *Timer* task: a periodic task with an intermediate priority level. The period of the task is calculated so as to generate a specified number of preemptions of the counter task in a reader task period.
- *Reader* task: a high priority service that periodically reads the counter value and stores the increment in the period. In the experiments the task period has been set to 1 second.

The C/XAL does not support tasks, in these case, previous tasks Timer and Reader have been substituted by interrupt handlers associated to internal timers.

These scenario has been executed several times for different values of the number of preemptions incurred by the counter task in a period of the reader task. The values used for the evaluation go from 1 to 1000 preemptions per second, which correspond to timer task periods between 1000 and 1 milliseconds. This scenario has been executed in a slot in the XtratuM schedule which is longer than the total duration of the experiment, so that no additional interference due to partition context switch is incurred. Figure 3 shows the results of the evaluation for the three execution environments with respect to a reference. The reference is the number of instructions executed in one second by a C program running on a native LEON2 platform. This figure plots the number of operations (NoO) for each execution environment with different values of the number of preemptions generated by the Timer task or interrupts generated by a timer in the C/XAL environment. The NoO is computed as $reference\_value - measured\_value/reference\_value$. The performance loss can be computed as $1 - NoO$

The above results show that the most performance execution environment is XAL that performs 0.996 of the reference value when the number of preemptions is 100 which corresponds to a periodic interrupt of 10 milliseconds. In the same conditions, LithOS perfoms 0.983 of the reference value which corresponds to an overhead of 1.7%. RTEMS achieves 3.30% in the same conditions.

When the number of preemptions are 1000 (Timer task period of 1 millisecond), the XAL environment and LithOS perfom 0.964 and 0.853 respectively. The overhead in these cases is 3.6 % and 14.7 %. RTEMS is not evaluated for number of preemptions higher than 100 due to the configuration of RTEMS for space application is configured with a clock resolution of 10 milliseconds.

These results show that the overhead introduced by LithOS at the task frequencies normally used in space applications (lower than 0.1 Khz) is lower than 2% that could be considered very low for a virtualisation layer.

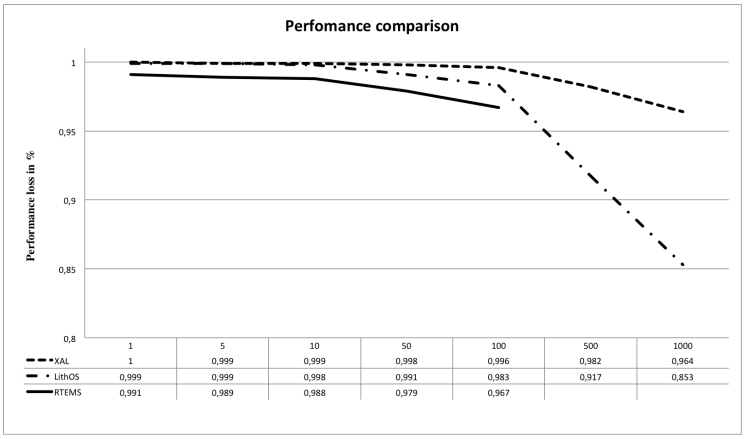

Fig. 3. Performance loss comparison

5.4 Footprint

Table 5.4 shows the footprint values of a LithOS partition. The table details the bss increment depending on the number of resources defined and the size of the data. With respect to the comparison with the rest of the other exectution environments, Table 5.4 shows the footprints of XtratuM, XAL, LithOS and RTEMS.

code	58 Kb
data	8 Kb
bss	16 Kb
Processes	8 KB * Number of processes (size of the stack)
Events	32 B * Number of events
Semaphores	40 B * No of semaphores
Blackboards	44 B * No of blackboards * Size of message
Buffers	172 B * No of buffers * No of messages * Size of message

Table 3. Footprint measurements

6 Conclusions

In this paper we presented LithOS, a guest real-time operating system ARINC-653 compliant for partitioned systems based on XtratuM. LithOS provides the partition management, intra-partition and inter-partition communication, time management and health monitoring services according to the specification. The ARINC-653 skin focuses on current developement performed on the ARINC-653 standard, in order to add flexibility and portability to the applications and provide the whole benefits of the ARINC-653 standard.

Component	text	data	bss
XtratuM	61.4K	6.5K	68.2K
XAL	22.7K	8.0K	0.7K
LithOS	43.1K	8.2K	46.0K
RTEMS	109.3K	3.2K	2.9K

Table 4. Memory footprint.

LithOS includes the multiple schedule services in order to deal with multimode systems. This service follows the standard defined as extended services.

Additionally, LithOS includes a set of services that are not included in the standard for partition management. These services are provided by XtratuM and are offered as services in LithOS. These services are labelled as non-portable in order to emphasise its non portability.

Finally, a performance evaluation of LithOS has been included to ensure the correctness of the behavior. Tests cases are currently validated by means of the ARINC-653 Part 3 in the standard and the services provided by LithOS are verified in conformity with the ARINC-653 Part 1. The performance evaluation includes the measurement of some of the implemented services.

7 References

[1] *Avionics Application Software Standard Interface (ARINC-653)*, March 1996. Airlines Electronic Eng. Committee.

[2] *Avionics Application Software Standard Interface (ARINC-653). PART 3 CONFORMITY TEST SPECIFICATION*, October 2006 2006. Airlines Electronic Eng. Committee.

[3] *Avionics Application Software Standard Interface (ARINC-653). PART 2 EXTENDED SERVICES* , January 2007 2007. Airlines Electronic Eng. Committee.

[4] M. Masmano, I. Ripoll, and A. Crespo. Introduction to XtratuM. 2005.

[5] M. Masmano, I. Ripoll, and A. Crespo. An overview of the XtratuM nanokernel. In *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2005.

[6] M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge. Xtratum: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop*, Dresden (Germany), 2009.

[7] M. Masmano, I. Ripoll, A. Crespo, J.J. Metge, and P. Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[8] M. Masmano, I. Ripoll, S. Peiró, and A. Crespo. Xtratum for leon3: an open source hypervisor for high integrity systems. In *European Conference on Embedded Real Time Software and Systems. ERTS2 2010.*, Toulouse (France), 19-21 May 2010.

[9] John Rushby. Design and verification of secure systems. *ACM Operating Systems Review*, 15(5):12–21, Dec 1981.

[10] James Windsor and Kjeld Hjortnaes. Time and space partitioning in spacecraft avionics. *Space Mission Challenges for Information Technology*, 0:13–20, 2009.