XtratuM for LEON3: an Open Source Hypervisor for High Integrity Systems

Miguel Masmano, Ismael Ripoll, Alfons Crespo and Salvador Peiro

Instituto de Informática Industrial. Universidad Politécnica de Valencia.
Camino de Vera s/n, 46022 Valencia, Spain
{mmasmano, iripoll, acrespo, speiro}@ai2.upv.es

Abstract: The growing complexity of the payload on-board satellite software experimented during the last years has raised the interest of the CNES and the ESA to explore the possibility of using a TSP-based architecture as base of the payload software of its new generation satellites. Such a solution can be implemented by using different approaches: virtualization, μ-kernels, separation kernels.

XtratuM is an open-source hypervisor targeted high-critical real-time systems which has been selected by ESA to be ported to the LEON3 processor in the frame of the Securely Partitioning Spacecraft Computing Resources project. This paper addresses the current status of the XtratuM open-source hypervisor for LEON3. In addition an early evaluation of it is also sketched.

Keywords: TSP, virtualization, hypervisor, high-integrity systems, real-time.

1 Introduction

Both, the advances in the processing power and the increment in resources achieved during the last decade have raised two important consequences in the payload on-board satellite software. On the one hand, it has permitted to turn the payload on-board satellite software from simple data processing, mainly responsible for only formatting and transferring the data to the ground into complex and autonomous data processing software. Despite of increasing notoriously the development and maintenance cost. On the other hand, multiple applications are able to run in a single processor, reducing the quantity of hardware necessary to build a satellite, and as a consequence its final cost. To facilitate such a model, the execution time and memory space of each application must be protected from the rest of the applications present in the system. Partitioned software architecture have evolved to provide such security. The separation kernel proposed in [1] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interface. The MILS [2] 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 [3] standard specifies the baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture. Although not explicitly stated in the standard, it was developed considering that the underlying technology used to implement the partitions is the separation kernel.

A virtual machine (VM) is a software implementation of an architecture that executes programs like a real machine. In addition, the low overhead, fairly closed to that of the native hardware, turns this solution into a candidate to build partitioned systems.

The European space sector, CNES firstly with the LVCUGEN project, and later, ESA with the *Securely Partitioning Spacecraft Computing Resources* project are currently assessing the feasibility and benefits of a TSP-based solution for payload on-board satellite software based on virtualization or related technologies (see for a description of the architecture proposed by ESA [4]).

The LVCUGEN project started by the end of 2008 with the aim of developing a highly reusable and configurable TSP-based architecture for on-board payload software. As base of this architecture, the XtratuM hypervisor [5,6] was selected to be adapted to the AT697 (LEON2) processor, which lacks of a MMU. This feature forced XtratuM to implement inter-partition read-only protection instead of the desired full-spatial isolation.

The Securely Partitioning Spacecraft Computing Resources project started by the middle of 2009 to build a TSP-based architecture for on-board payload software to be tested on the ESA's Eagle-Eye simulator. XtratuM was selected as the open source alternative to be adapted to the GR-CPCI-XC4V (LEON3 processor) processor, with MMU.

This paper describes the experiences of adapting XtratuM to the LEON3 processor. This processor, unlike its predecessor, includes a MMU enabling, thus, XtratuM to eventually implement full spatial isolation. Furthermore, it describes the XEF image format, a partition image format introduced in this version of XtratuM to improve the management of partitions during the deployment phase. An initial evaluation of the most significant metrics is also provided.

2 XtratuM hypervisor

XtratuM 1.x was initially conceived as an improved replacement of RTLinux's virtualization layer [7]. There was two design goal: avoid the legal problems related with the patent that protected the exact virtualization mechanism used by RTLinux, and avoid its technical limitations. XtratuM 1.x was designed as a Linux Kernel Module. Once loaded (as a Linux module), it takes over the essential hardware devices (basically interrupts and timers) to enable the concurrent execution of two or more OSes[1], being one of them a real-time OS. The other hardware devices, as well as the boot sequence, were left to be managed by Linux. This approach let us speed up the implementation of a first working prototype, however, it was not completely satisfactory: XtratuM and Linux still shared the same memory area, and both run in supervisor mode (Ring level 0).

After this first experience, the second version (XtratuM 2.0) was redesigned to be independent of Linux. In the rest of this paper, the term XtratuM will refer to this second version or above. This version is being used in the project LVCUGEN [8], whose goal is to build a TSP-based solution for payload on-board software (for the aerospace industry), highly generic and reusable. The TSP-based architecture has been identified as the best solution to ease and secure reuse, enabling a strong decoupling of the generic features to be developed, validated and maintained in mission specific data processing.

XtratuM is, according to the IBM categorisation [9], a type 1 (bare-metal) hypervisor that uses para-virtualization. The para-virtualized operations are as close to the native hardware as possible. Therefore, porting an operating system that already works on the native system is a simple task: just to replace some parts of the operating system HAL (Hardware Abstraction Layer) with the corresponding hypercalls[2].

XtratuM was designed to meet high integrity system requirements. Its most relevant features are:

- Bare-metal hypervisor.
- Implements para-virtualisation techniques.
- Designed for embedded systems: low footprint, some devices can be directly managed by a designated partition.
- Strong temporal isolation: fixed cyclic scheduler.
- Strong spatial isolation: all partitions are executed in processor user mode, and do not share memory.
- Fine grain hardware resource allocation via a XML configuration file.
- Robust communication mechanisms (ARINC 653-1 based sampling and queueing ports).
- Health monitoring capabilities.
- Two security levels: standard and system partitions.

2.1 Architecture and design

XtratuM has been implemented following a monolithic approach, running all its services in the processor highest-privileged mode and in a single memory space; all the services are directly reachable from any part of the hypervisor.

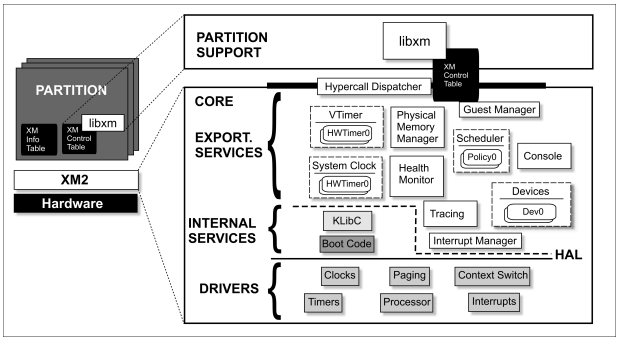

Fig. 1. XtratuM architecture.

Figure 1 sketches the complete system architecture.

The main components of this architecture are:

1. Hypervisor: XtratuM is in charge of virtualization services to partitions. It is executed in supervisor processor mode and virtualizes cpu, memory, interrupts and some specific peripherals. The internal XtratuM architecture includes: physical/virtual memory management, scheduling (fixed cyclic scheduler), interrupt management, clock and timers management, inter-partition communication (ARINC 653-1 based communication model) and health monitoring.
2. Partitions: a partition is an execution environment managed by the hypervisor which uses the virtualized services. Each partition consist of one or more concurrent processes (implemented by the operating system of each partition), sharing access to processor resources based on the requirements of the application. The partition code can be:

- An application compiled to be executed on a bare machine using directly the services of XtratuM. Note that XtratuM provides an execution environment that is not exactly like the original hardware but somewhat more "friendly" (console services, easy to use timers, etc.)
- An operating system (and RTOS and a general purpose one) and its applications.

[1] RTLinux virtualiser only is able to run Linux and one real-time OS.

[2] Hypercall is the name of the hypervisor services. It is based on the term *system call* which refers to the services of an operating system

Partition code need to be *virtualized* to be executed on top of the hypervisor. Depending on the type of execution environment, the virtualization implications in each case can be summarised as:

- Bare applications: the application has to be virtualized by using the services provided by XtratuM. The application is designed to run directly on the hardware and it has to be aware of it.
- Operating system application: when the application runs on top of a (real-time) operating system, it uses the services provided by the operating system and does not need to be adapted. But the operating system has to deal with virtualization (ported to be XtratuM aware).

Two different type of partitions can be defined: system and user. A system partition is able to change the execution state of a user partition (*suspend / resume / halt / reset*) or the whole system.

2.2 LEON2 version limitations

The current version of XtratuM for LEON2 processor has a series of limitations: the SPARC architecture does not provide any kind of virtualisation support. It implements the classical two privilege levels: supervisor and user; in order to guarantee isolation, partition's code has to be executed in user mode, only XtratuM can be executed in supervisor mode. This requirement imposes limitations to those operating systems which require both modes.

Additionally, the LEON2 processor lacks of a Memory Management Unit (MMU) which prevents XtratuM to implement:

- Full spatial isolation. Nevertheless, a kind of partial spatial isolation is provided (through the LEON2's memory write protection registers: a partition cannot overwrite memory which belongs to others, however, it is still able to read from any memory address. In addition, the trap raised by a partition when it is trying to write in write-protected area is received several cycles later (unlike MMU traps which is synchronous), being, thus, *rather* difficult to find out which was the offending instruction. And it is even still much more difficult to try to sort out the situation without killing the offending partition.
- Shared memory between partitions. The use of shared memory could be used to implement efficient, zero-copy inter-partition communication mechanisms. Currently, XtratuM only implements queuing and sampling ports, both of them requiring two copies of data (sender $\rightarrow$ XtratuM $\rightarrow$ receiver). The use of shared memory should enable the implementation of more efficient inter-partition communication. Besides, the lack of shared memory also prevents XtratuM from sharing sections between partitions. For example if two partitions run Linux, then two copies of the Linux code must be copied in memory.

Furthermore, during the evaluation of XtratuM 2.2.x were raised several suggestions about the binary format used by the image of XtratuM and the partitions:

1. A raw binary image could be corrupted without being detected by the bootloader.
2. An user payload embedded in the partition's header is missed. This payload shall be used by the partition's supplier to insert tracking information/versions/etc.
3. The binary image was not processed in any way, techniques such as compression, digestion, cryptography could be beneficially applied.

3 XtratuM for LEON3 processor

The LEON3, successor of the LEON2 processor, is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC-v8 architecture. Designed by Aeroflex Gaisler, it has been released under the GNU GPL licence. In addition to all the features already presented on the LEON2 processor, this version of the processor includes support for SMP systems, and a Memory Management Unit (MMU) (SPARC reference MMU with configurable TLB).

By using this MMU, XtratuM is able to:

- Implement full spatial isolation. No partition is longer able to read from memory areas belonging to other partitions.
- Support inter-partition shared memory: two or more partitions are able to share memory areas (specified in the XML configuration), permitting to design and implement more efficient inter-partition communication mechanisms. And, in addition, code sections could be shared by partitions avoiding duplicity of code.

The adaptation of XtratuM to this processor has consisted in virtualizing the MMU: each partition has its own virtual memory map where the top area is reserved for XtratuM. XtratuM itself is mapped on this area with supervisor permissions. The rest is filled according to the definition of the partition (XML configuration file). In addition, a partition has the capability of defining new memory maps and updating them.

In addition, a set of new hypercalls is provided, enabling a partition to manage its memory maps.

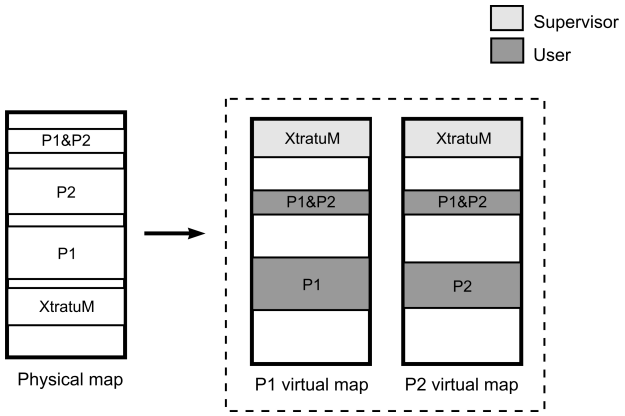

Fig. 2. Memory map of two partitions.

Figure 2 shows the two virtual memory maps build by XtratuM as a result of the following XML configuration file:

```
<Partition name=P1 ...>
 <PhysicalMemoryAreas>
  <Area start=0x10000 size=64KB mappedAt=0x8000/>
  <Area start=0x30000 size=32KB flags=shared/>
 </PhysicalMemoryAreas>
...
</Partition>
<Partition name=P2 ...>
 <PhysicalMemoryAreas>
  <Area start=0x20000 size=64KB mappedAt=0x8000/>
  <Area start=0x30000 size=32KB flags=shared/>
 </PhysicalMemoryAreas>
</Partition>
```

This XML configuration file defines two partitions P1 and P2 and allocates two memory areas to each partition. One of them, the area starting at `0x30000`, shared. In addition, the configuration establishes that the areas non-shared, the area starting at `0x10000` and the area starting at `0x20000`, are mapped at the same virtual memory address, that is, `0x8000`.

3.1 Memory management virtualisation

Memory management is from far, one of the hardest thing to be efficiently virtualized in a computer, both in terms of the mechanisms required in the hypervisor and modifications required to port an OS. In order to virtualize the memory XtratuM follows the XEN approach [10], that is:

1. Each partition is in charge of managing the page table (but the initial one, which is created by XtratuM).
2. XtratuM is mapped in the top of every memory map, thus avoiding a TLB flush when entering and leaving the hypervisor.

The initial memory map of each partition is built by XtratuM following the description found in the XML configuration file, it cannot be updated by the partition. If a partition requires a new memory map, then it has to define the new memory map by registering a set of pages from its own memory. Once registered, these pages become read-only, so subsequent updates must be validated by XtratuM.

3.2 Changes in the XtratuM core

Three fundamental changes have been performed in the XtratuM core:

- XtratuM implements a new module called **virtual memory manager** which is in charge of managing the virtual maps, and is able to create/release them and map/unmap physical pages.
- As mentioned above, XtratuM implements three new hypercalls: `XM_set_page_type()` which permits a partition to register new memory maps, `XM_update_page32()` which allows a partition to update an entry in an already existing memory map, and `XM_write_register32(PTD1_REG32,)` which enables a partition to change the current memory map with a new one.

3.3 Changes in the XML configuration file

The XML configuration file did not consider any aspect related with the MMU management, being mandatory to modify it after the addition of MMU support. However, this file, as long as it has been possible, should be kept compatible backwards. We have included two new attributes to the definition of the physical memory area element: `@flags` (`/Partition/PhysicalMemoryAreas/Area/@flags`) and `@mappedAt` (`/Partition/PhysicalMemoryAreas/-Area/@mappedAt`).

The attribute `@flags` enables the system integrator to give properties to each memory area. These properties are:

Uncacheable when this property is used, the memory area remains as not cached in the virtual memory map. This provides a finer-grain control to define which memory areas should be cached or not. By default a memory area is cached.

Read-only when set, the memory area is always mapped as read-only. The partition is unable to modify its content. By default memory areas have read/write permissions.

Unmapped if specified, the memory area is not mapped on the initial memory map. Nonetheless, this property does not prevent the partition to map the memory area. By default all memory areas are mapped on the initial memory map.

Shared this property enables the system integrator to allocate the same memory area to one or more partitions. By default memory areas are private and can be only allocated to one partition.

On the other hand, the attribute @mappedAt enables the system integrator to define the location of the memory area at the initial memory map. If this attribute is not specified, the memory area is mapped 1:1, that is, at its correspondent physical address.

3.4 Changes in the partitions

No special change has to be performed in partitions, allowing fully backward compatibility. A XtratuM 2.2 partition which already worked in a MMU-less system works correctly in this new version of XtratuM. However, the addition of MMU supports open the possibility of porting more complex OSes such as Linux which were not possible in a MMU-less system.

3.5 The XEF partition image format

This new format is designed with the aim of taking into account all the suggestions given about the LEON2 partition image, enumerated in the section 2.2, while keeping binary compatibility with the previous binary format. Its features are:

- It is a wrapper layer for the already existing binary XtratuM/partition format.
- Includes a digest sum to verify integrity. As digest algorithm we have chosen the MD5 algorithm.
- Includes a partition supplier's payload of 16 bytes. This spare space is left for the partition creator.
- (*Optional*) Implements compression. As compression algorithm XtratuM implements the LZSS one.
- Implements the concept of sections. An image can be integrated by several sections, each one located in a different memory address. This feature shall reduce the size of a binary image since including padding in the image is not longer necessary.

MD5 digest algorithm: discussion As digest algorithm we have chosen the MD5 algorithm, despite knowing it has already been compromised, because it has a reasonable trade-off between calculation time and the security level it offers. According to our tests, the time spent by more sophisticated options such as SHA-2, Tiger, Whirlpool in the LEON3 processor was not acceptable (as illustration, 100 KB took several seconds to be digested by a SHA-2 algorithm).

LZSS compression algorithm: discussion LZSS is a lossless data compression algorithm, a derivative of LZ77, that was created in 1982 by James Storer and Thomas Szymanski [11]. When compression is enabled, the partition binary image is compressed using a LZSS algorithm. The reasons because this algorithm was selected are:

- Reasonable compression rate (up to 80% compression rate in our experiments).
- Fast decompression algorithm.
- Fairly acceptable trade-off between the previous two parameters.
- Implementation simplicity.
- Patent-free technology.

Aside from LZSS, other algorithms which were regarded were: Huffman coding, gzip, bzip2, LZ77, RLE and several combinations of them. Table 1 sketches the results of compressing core binary of XtratuM (78480 bytes) with some of these compression algorithms.

Algorithm	C. size	C. rate
LZ77	43754	44,20%
LZSS	36880	53,00%
Huffman	59808	23,80%
Rice 32bits	78421	0,10%
RLE	74859	4,60%
Shannon-Fano	60358	23,10%
LZ77/Huffman	36296	53,76%

Table 1: Outcomes of compressing a xm_core.bin (78480 bytes)

XEF tools In order to support this new format, XtratuM includes two new utilities:

- **LibXEF**: a new library which allows, among other things, to create a XEF from an ELF file, compression and decompression algorithms, integrity algorithm, etc.
- **xmeformat**: once invoked, allows to translate an partition into the XEF format. It also can check the integrity of a XEF file.

4 Assessment

This section provides the initial evaluation of XtratuM for LEON3 processor. A development board GR-CPCI-XC4V (LEON3) processor at 50MHz with 16MB of flash PROM and 128MB of RAM has been used during this evaluation.

A set of performance indicators are initially defined. For each performance indicator, a set of scenarios are built and evaluated.

Loss of performance due to scheduling. This indicator measures the performance loss when a partition is executed directly on the hardware with respect to its execution as partition in a partitioned system on top of XtratuM.

Loss of performance due to the number of partitions. This indicator measures the performance lost when the number of partitions increases.

Partition context switch time (PCS). This is the time needed by XtratuM to stop the execution of a partition and to resume the next partition in the scheduling plan.

4.1 Scenario description

The scenario is composed by several bare partitions (*Counter*) that increase integer variable (counter) and a partition (*Reader*) that is able to read and print the counter values of the other partitions. This partition is scheduled once per MAF (last slot in the MAF) with enough duration to print all counter values in the serial port. Table 2 shows the defined scenarios. The MAF includes one or more *Counter* slots and only one *Reader* slot. (NPC: Number of Counter partitions; NSts: Number of slots per partition in the plan; SDur: Slot duration in milliseconds; MAF: total duration of the plan in microseconds).

Case	NPC	NSts	SDur	MAF	Plan
1	3	1	1000	4000	C1;C2;C3;R
2	3	5	200	4000	5*(C1;C2;C3;);R
3	3	10	100	4000	10*(C1;C2;C3;);R
4	3	50	20	4000	50*(C1;C2;C3;);R
5	5	1	100	1500	10(C1;C2;C3;);R
6	10	1	100	2000	C1;C2;C3;R
7	15	1	100	2500	C1;C2;C3;R
8	20	1	100	3000	C1;C2;C3;p4

Table 2. Scenario definition

4.2 Evaluation results

Scenarios 1 to 4 are compared to evaluate the performance loss due to the partition context switch. Scenario 1 is used as a reference. Table 3 shows the results of Case 1 after 100 MAFs. The partition reader prints the counter increments performed in the slots of every partition in a MAF. At the end of the experiment, the results for each partition are the average number of increments per MAF, the maximum and minimum number of increments in a MAF, and a standard deviation.

Case 1	Counter 1	Counter 2	Counter 3	Summary
Avg.	8331550	8331556	8331528	8331548
Max.	8331560	8331562	8331538	8331562
Min.	8331547	8331552	8331527	8331527
Stdev	3.27	4.02	5.22	3.22

Table 3. Case 1: 3 Partitions, 1 slot of 1 sec.

To compare different cases, a summary of the case is calculated. The average is calculated as the average of the partition's averages. The maximum is the maximum of all the maximums. The minimum is the minimum of all the minimums and the stdev is the maximum of the standard deviations.

Table 4 compares the summary results obtained in the first four cases. The average value is the average of the case summary. The difference value is the number of counter increases that have not been completed when the slot of Case 1 was split up into smaller slots (Cases 2 to 4). The performance loss is the difference expressed in percentage with respect to Case 1. This table also provides the an estimation in microsecond a of the time spent in performing the PCS as result of the counter values.

	Case 1	Case 2	Case 3	Case 4
Average	8331527	8325758	8317924	8259992
Difference	0	5769	13603	71535
Performance lost	0	0.069	0.163	0.859
PCS (estimation)	-	138	162	171

Table 4. Comparison of Cases 1 .. 4

Scenarios 5 to 8 have defined to evaluate the impact of the performance loss due to the number of partitions. Table 5 compares the summary results obtained in four scenarios.

Summaries	Case 5	Case 6	Case 7	Case 8
Avg.	831760	831606	831623	831684
Max.	831874	831760	831751	831773
Min.	831679	831581	831572	831523

Table 5. Comparison of Cases 5 .. 8

These results shows that the impact of the number of partitions in the performance of the system is not relevant from the point of view of the partition. It has an impact in the memory required to execute the partitions but this depends on the code and data of the partition.

The partition context switch measured adding breakpoints at the start and end of the PCS in the kernel shows a average value of 110 microseconds. The maximum PPCS measured has been 116 microseconds.

Still more measures to evaluate the impact when a partition contains an operating system are required.

5 Conclusions and future work

XtratuM for LEON2 processor already proved that virtualization can be implemented over the SPARCv8 architecture. However, due to the constraints imposed by this processor (mainly the lack of MMU), it was impossible to design and implement full spatial partitioning. A partial space isolation was provided instead. The use of the MMU provided by the LEON3 processor has enabled us to implement a real full spatial partitioning on XtratuM

Our main conclusion, after studying the LEON3 processor and because of our experience on the 32-bit Intel Architecture, is that it is feasible and rather beneficial to add the MMU support of the SPARCv8 architecture.

It is important to note that the use of a MMU increments the overhead impact with respect a MMU-less system. The reasons for this additional overhead are:

– Translation from virtual to physical space: it requires additional processor cycles. The TLB mitigates this effect, however, due to its small size, the size of the pages must be selected carefully.

– Physical memory access: without MMU, all the physical memory is fully accessible, however, once the MMU, and due to space constraints, the whole physical memory map is not longer mapped. When XtratuM needs to access to an unmapped page, it has to be mapped on the current memory map.
– TLB faulting: as mentioned above, LEON3 implements a small TLB with up to 32 entries. Using 4KB pages allows us having up to 128KB (4KB*32) of memory presents in TLB.
– TLB flushing: SPARCv8 implements the concept of context to avoid the necessity of flushing the TLB after each context switch, however, the overhead of context switch should be regarded.
– Page faulting is not really a problem, since should not be any.

6 Acknowledgement

This work has been partially funded by EADS-Astrium under an ESA contract and the Spanish Government Research Office under grant TIN2008-06766-C03-02/TIN.

We wish to thank Jean Jacques Metge and Paul Arberet from CNES for his support in the XtratuM activities.

7 References

[1] John Rushby. Design and verification of secure systems. volume 15, pages 12–21, Pacific Grove, California, dec 1981.

[2] Jim Alves-Foss, Paul W. Oman, Carol Taylor, and Scott Harrison. The mils architecture for high-assurance embedded systems. *IJES*, 2(3/4):239–247, 2006.

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

[4] J. Windsor and K. Hjortnaes. Time and space partitioning in spacecraft avionics. In *IEEE Conference on Space Mission Challenges for Information Technology*, July 19-23. Pasadena (USA) 2009.

[5] M. Masmano, I. Ripoll, and A. Crespo. An overview of the xtratum nanokernel. In *Workshop on Operating Systems Platforms for Embedded Real-Time applications*, 2005.

[6] 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.

[7] M. Barabanov. A Linux-Based Real-Time Operating System. Master's thesis, New Mexico Institute of Mining and Technology, Socorro, New Mexico, June 1997.

[8] P. Arberet, J.-J. Metge, O. Gras, and A. Crespo. TSP-based generic payload on-board software. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[9] IBM Corporation. IBM systems virtualization. Version 2 Release 1 (2005). http://publib.boulder.ibm.com/-infocenter/eserver/v1r2/topic/eicay/eicay.pdf.

[10] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.

[11] James A. Storer and Thomas G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982.

8 Glossary

CNES Centre National d'Études Spatiales
ESA European Space Agency
IMA Integrated Modular Avionics
LZSS Lempel-Ziv-Storer-Szymanski
MILS Multiple Independent Levels of Security and Safety
MMU Memory Management Unit
TLB Table Look-aside Buffer
TSP Time and Space Partitioning