AIR HYPERVISOR USING RTEMS SMP

Bruno Gomes, Daniel Silveira, Laura Gouveia, Luis Mendes

GMV, Av. D.João II Lote 1.17.02, Torre Fernão Magalhães 10°, 1998-025 Lisboa, Portugal – Email: bmgomes@gmv.com, daniel.silveira@gmv.com, lasequeiragouveia@gmv.com, lmurta@gmv.com

ABSTRACT

The new RTEMS version 5 provides full support for Symmetric Multiprocessing (SMP) on real-time space applications. GMV with its TSP/IMA AIR hypervisor virtualizes RTEMS 5 and fully supports RTEMS SMP implementation on multi-core On-board Computers. AIR I/O component integrates device control into a partitioned environment opening a door to optimal performant solutions for the application of Remote Terminal Units and also in Buses & Communication protocols.

This paper presents AIR hypervisor architecture, its set of available application programing interfaces and the paravirtualized supported operating system RTEMS. Two use cases are depicted exemplifying the advantages of AIR with RTEMS SMP on on-board computers with LEON4 multi-core processor.

Key words: IMA; TSP; AIR; Hypervisor; Operating Systems; I/O; SMP; RTEMS; Multi-Core.

1. INTRODUCTION

The growth in complexity of software systems functionalities associated with more powerful on-board computers lead to a complex task by the system integrator. This extensive task comprises the responsibility to provide functional validation as a single component for systems where multiple software applications might co-exist on the same computer while still respecting every application's critical class and also provide fault detection and containment at software level.

One of the objectives of the Integrates Modular Avionics (IMA) concept in aviation was to achieve an integrated system architecture that preserves fault containment properties while creating a clear separation between software modules that share the common hardware. This architecture uses Time and Space Partitioning (TSP) to share the computing platform between possibly multiple cooperating applications. The concept of partition is therefore an allocation of resources to an application in terms of memory space (spatial partitioning), CPU time (temporal partitioning), I/O device access, CPU privilege mode and communication via ports. ARINC-653 [1] standard defines the software baseline specification for application development within an IMA architecture.

The European Space Agency (ESA) has already identified the benefits of incorporating software TSP into the spacecraft avionics architecture to manage the growth of mission functions implemented in the on-board software [2]:

- Reduced integration effort
- Hardware resource savings
- Fault containment in an integrated system
- Mixed Software Criticality (by clearly defining the different levels of criticality)

The IMA for Space (IMA-SP) project defines a standard IMA platform specific for the space domain [3]. The platform does not only provide services on operating system level, but includes domain specific services, such as Fault Detection, Isolation, and Recovery (FDIR), mode control, on-board software maintenance and a generic I/O solution.

GMV AIR is a type-1 hypervisor based on the IMA-SP paradigm that allows a single host computer to simultaneously execute several independent Real Time Operating Systems (RTOS) partitions with temporal and spacial partitioning by defining a schedule based on the partition's criticality level [4] [5]. AIR fully supports the guest OS Real-Time Executive for Multiprocessor Systems (RTEMS) [6] versions 4.8i and 5 for SPARC v8 architecture processors such as LEON2/LEON3/LEON4. AIR also supports the ARMv7 instruction set with NEON instructions.

2. AIR OVERVIEW

AIR is an additional software layer between the application and hardware that ensures applications can be partitioned into a set of isolated systems in both time, through an execution schedule, and in space through its own memory region and processor cores. All AIR versions are open source under GPL v2 license [7].

The AIR hypervisor architecture is depicted in Figure 1

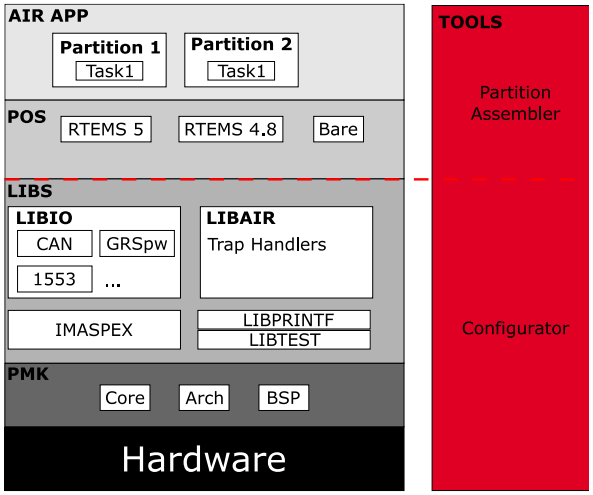

Figure 1. AIR Architecture Component Breakdown

and consists of the following modules:

- Partition Management Kernel (PMK): It holds the main functionality of the hypervisor and implements TSP, it corresponds to the TSP kernel of the TSP System Executive platform. In turn it can be divided in three modules:
  - Core: which holds the core functionality of AIR such as partition management, TSP paradigm, multi-core handling, syscall implementation and health monitor;
  - Arch: The generic functionality needed for AIR to run a processor architecture, currently is supporting the SPARC architecture and ARM processor boards executing the armv7 instruction set with or without NEON support and no virtualization extensions
  - BSP: The specific functionality needed for AIR to run a Board Support Package, which currently supports the LEON2, LEON3 and LEON4 processor boards as well as a dual core cortex A9.
- Partition Operating System (POS): It holds the RTOS being paravirtualized by AIR, it corresponds to the guest OS of TSP System Executive Platform; currently the module supports RTEMS SMP, Qualified RTEMS 4.8 Improved, RTEMS 5 and Bare bone execution without a RTOS.
- Libs: This module is composed by a set of libraries implementing functions to bridge the user application, RTOS and PMK. They correspond to the TSP System Executive extensions with the exception of IMASPEX. The libraries are:
  - IMASPEX: Implementation of standard ARINC 653 API enriched by ESA specifications taken in IMA-SP and AIR which hide the Core sub module API holding the system calls; it corresponds to the Time and Space Abstraction Layer of the TSP System Executive Platform;
  - LIBAIR: Implementation and interface of system calls used by a RTOS trap handler in order to paravirtualize the RTOS. Gives RTOS AIR Health Monitor support;
  - LIBIOP: Implementation of device drivers based on the same device drivers source code present in RTEMS;
  - LIBPRINTF: Implementation of the printf functionality, useful for debugging and pure Embedded RTOS that do not have support to print on a console or other device;
  - LIBTEST: A set of auxiliary functions used to execute, integrate and a validation test used to be applied in the test and validation campaign of AIR and the device drivers.
- Tools: The tools are implemented in parallel to the modules presented before, as it configures and bind the previous modules in order to generate the executable. The tools are divided in two submodules:
  - Configurator: Configures AIR for a specific architecture, Board Support Package (BSP) and processor. Generates accordingly a set of makefiles for the RTOS and libs modules. It is also used at application level to generate the application's makefile;
  - Partition Assembler: Executed at user application level, is responsible for aggregating all the built partitions into a single executable file, in order for the PMK at booting stage to effectively manage and deploy the partitions in memory.

2.1. LIBAIR and IMASPEX

AIR Libs supports the development of applications [5] for an AIR partition especially when no guest RTOS is used. Table 1 summarizes the available services that enable an application to be ARINC based compliant. In case of a partition executing a guest OS most of the services except partition management, communication and health monitor, can be executed by the available paravirtualized guest OS native functions.

2.2. LIBIOP

In a partitioned system, the quantification of time spent in I/O tasks is even more critical, since it shall be known on whose behalf I/O tasks are performed. The costs for these tasks should be booked to the applications that actually benefit from it. Robust partitioning demands that applications use only those time resources that have been reserved for them during the system design phase. I/O activities shall, hence, be scheduled for periods when the applications that use these specific capabilities are actually being executed. Furthermore, safety requirements may forbid that some partitions are

Table 1 – Available AIR Services

Services	System Call
Partition Management	Get Status, Get ID, Set Mode, Virtual Core Number, Virtual Core ID
Communication (Queuing/ Sampling ports)	Create, Get Status, Send/Write, Receive/Read, Get ID
Time Management	Get elapsed ticks, Get microseconds per tick, Get /Set time of day
Cache Management	Flush, Activate, Deactivate, Freeze
Health Monitor	Raise, Get Status
Interrupt Handling	Enable, Disable, Mask Unmask IRQ
FPU Control	Enable / Disable FPU
Scheduling	Get/Set Schedule, Get Status

interrupted by hardware during their guaranteed execution time slices. As a consequence, it must be ensured that I/O devices have enough buffering capabilities at their disposal to store data during the time non-interruptible applications are running [8]. LIBIOP was built and respects the characteristics of a partitioned system:

- Is generic and therefore decoupled from the application;
- Is robust, in the sense that it can be used by more than one partition without interference;
- Routes data to its rightful owner (a given application in a given partition);
- Is quantifiable (i.e. its execution time must be bound and measurable);
- Does not interrupt, disrupt or have any kind of impact in the time and space partitioning of the applications.

In an IMA system the hypervisor must have a small trusted computing base so it can be easily certified to the highest levels of criticality. Including the I/O in the hypervisor would increase its size considerably and add complexity to otherwise simple software. By integrating I/O into a single dedicated system partition it is possible to add new functionalities to the platform in a short amount of time. No need for kernel reconfiguration or reimplementation and the partition can be handled the same way an application partition does, strengthening the sense of software reuse and building blocks. The clear location for drivers and protocols in this separate memory space eases board maintenance, where patches can be uploaded to this memory area without impacting kernel code. Also, having a dedicated LIBIOP partition makes hardware interrupts needless since even if a hardware interrupt were to be used to signal the arrival of new data, there were no means to activate a driver to obtain the data given the device driver is only activated according to the pre-defined partition schedule. The way to handle I/O is thus polling.

Figure 2 exemplifies the I/O partition routing capability by handling I/O device data to the correct channel.

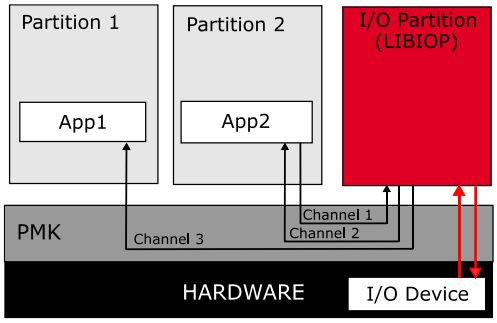

Figure 2 AIR I/O communication scheme.

Parallel computation in multi-core processors allows the possibility of deploying the LIBIOP in a dedicated core, resulting in considerable improvement in terms of data throughput and latency. An alternative approach to parallelization can be the employment of specific operating system features that enable the occupation of scheduled unused time. This alternative may provide mechanisms to schedule tasks in this otherwise unused time. Co-partitions are able to run in the unused time of an application execution window, without compromising the real time characteristics of the system [9]. Deploying the LIBIOP partition as a Co-Partition would allow to reduce the latency and improve performance as well.

Currently LIBIOP supports SpaceWire, Ethernet, MIL-STD-1553 and CAN bus interfaces.

2.3. AIR CONFIGURATOR

AIR Configurator is a two step command line interface tool that provides system developers the ability to auto-generate all the required development environment and subsequently to validate and auto-generate all the applications dependencies given the high level XML system description.

On a first stage, the Configurator interacts with the developer in order to establish and create the system development environment by selection of the target architecture and Board Support Package (BSP). This stage generates all the necessary makefiles for building the supported RTOS as well as AIR's PMK and Libs.

Once at application level, the execution Configurator will validate the system description XML file while parsing its content. All necessary supporting files to interact with AIR and all required makefiles for every defined partition are auto-generated based on the XML settings. Also, the required supporting files for the Partition Assembler are auto-generated allowing the auto-execution of this tool at the end of the building stage.

3. SUPORTED RTOS

AIR partition configuration supports a RTOS RTEMS personality and bare applications that do not require a RTOS. AIR supports RTEMS version 5 and a space tailored version of RTEMS 4.8 [10] (named 4.8i in AIR).

3.1. RTEMS

RTEMS is an open source RTOS that supports open standard application programming interfaces (API) such as POSIX. AIR works with RTEMS paravirtualized, as paravirtualization means the process of redirecting to specific LIBAIR calls the handling of processor registers, the clock control and by adding the memory address virtualization. In detail, RTEMS is paravirtualized by technically doing the following:

- RTEMS only accesses virtual core information. Replace specific direct processor registers handling with LIBAIR handlers that actually manage the virtual core register;
- Trap table is paravirtualized in RTEMS - RTEMS trap table is virtual since the real trap table is handled solely by AIR;
- RTEMS interrupt handler is virtualized by AIR given the real handling is only performed by AIR;
- Manage RTEMS clock interrupt handler in order to support the partition time rather than direct timer registers;
- AIR controls the real interrupts Enable/Disable;
- AIR manages RTEMS partition handling permission of the Floating Point Unit (FPU);
- Change makefiles to add AIR library support into RTEMS
- Memory configuration is handled by AIR, for example to change the entry point from 0x40000000 to a virtual 0x41000000.

SMP is supported by RTEMS since version 5.

3.2. SMP

A SMP system is a tightly coupled multiprocessor system with identical processors running independently from each other where each processor shares the same memory and I/O devices usually connected by buses. A single operating system manages each core equally and therefore any core can execute any task just as well as any other core in the system. This contrasts with an Asymmetrical Multiprocessing (AMP) system where each core has a specific predetermined task or each core has a specific set of peripherals attached.

The SMP operating system may then be configured to try to keep all cores busy running application threads, in effect dynamically load balancing the system's work. Alternatively, SMP operating systems can also allow the assignment of a set of tasks/threads to a subset of processor cores to try to keep the subset of cores busy with the configured set of tasks. As a result of the hardware abstraction and load balancing in the system, the SMP operating system simplifies the task of developing software to run on multi-core hardware. From the programmer's view, there is only one OS to write an application for, which will automatically distribute the workload to the configured available cores. An SMP operating system therefore may provide the same programming semantics as a uniprocessor system.

AIR, by supporting a guest OS with SMP functionalities, gives the system developer the ability and flexibility to create a system with a heterogeneous set of partitions by eventually mixing partitions specificities. An application may be divided into a couple of partitions with one subset of the application being executed in a multi-core partition with SMP functionalities while the remaining application subset still being executed independently in a single core partition. This possibility not only allows the traditional application segmentation for safety reasons but enables as well the possibility of having an application segmented for safety and multi-core SMP optimization reasons.

4. PERFORMANCE EVALUATION

To characterize AIR performance, CoreMark [11] benchmark for embedded systems was ported to AIR with a RTEMS 4.8i partition. Based on Table 2, the overhead resulting from the presence of the hypervisor is about 1~2%, growing inversely proportional to the window size. For partition execution windows smaller than 0.001 seconds, the effect in the performance of the application starts to be noticeable.

While using CoreMark to characterize the effects of parallelization, the performance gains from an SMP configuration, using a multi-core AIR partition, is depicted in Table 3. The benchmark result was improved by 20% when parallelized over a dual-core partition. The performance gain does not improve considerably for a partition with three cores allocated, since gain increase rate decreases with the number of cores used, given the shared resource contention.

Since RTEMS version 5 supports SMP, a valid

Table 2 - CoreMark in AIR with different schedule window slots

Window Slot (s)	Execution Time (s)	Iterations per second	CoreMark	Performance Loss
NO AIR	12.280.	166	1.105	-
30	12.220	164	1.091	1.23%
1	12.240	163	1.089	1.38%
0.5	12.240	163	1.088	1.39%
0.1	12.280	163	1.086	1.71%
0.01	12.450	161	1.074	2.77%
0.001	12.669	157	1.050	4.88%

Table 3 - CoreMark in AIR with a SMP Partition

Number Cores	Execution Time (s)	Iterations per second	CoreMark	Performance Gain
1	12.280	163	1.086	-1.71%
2	10.0100	200	1.332	20.58%

performance test is yet to be performed in order to compare RTEMS SMP multi-core application performances versus the same application in an AIR multi-core RTEMS SMP partition.

5. USE CASES

Two real examples are presented, showcasing AIR multi-core SMP applicability.

5.1. Multi-core Implementation of the On-Board Software Reference Architecture with TSP Capability (MORA-TSP)

MORA-TSP objective was to demonstrate the feasibility and performance evaluation of an end-to-end process, tools and building blocks from application level specification using the component based approach of the On-Board Software Reference Architecture (OSRA) down to representative implementation of the combination of OSRA, TSP kernel, SMP operating system and multicore. AIR is used as the TSP kernel with RTEMS 5 as the guest OS with SMP feature enabled. A four core LEON4-N2X board is the target to demonstrate several scheduling scenarios of partitions and tasks. The scenarios were designed to be generic, but the partition names and tasks are derived from EagleEye TSP project. Only the IO partition in the TSP system has access to the device memory and this memory allocation is bound to run on only one core hence, the IO partition is scheduled to run in only one core and not allowed to switch cores during run-time.

One of this project's scenarios is presented in Figure 3. AIR allows the implementation of more than one multi-core partition to run RTEMS SMP configuration with in this case implementing Earliest Deadline First scheduling of tasks. This exemplifies the multitude of scheduling options system designers have while using a single multi-core on-board computer.

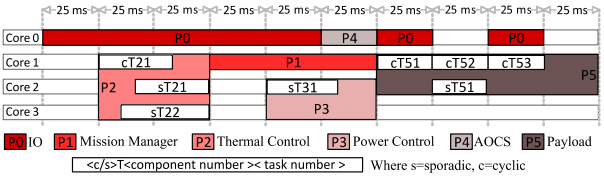

Figure 3. MORA-TSP SMP scenario.

One additional possibility emerged during the course of this project that may be explored in the future. The partition's number of cores is defined in the initial system configuration meaning a partition cannot alter its number of cores even if there's an additional available core at some point in the schedule. Even if AIR were to support this feature, RTEMS would also have to be adapted to be aware of a change in the number of available cores after booting.

It should be noted that the presented SMP scenario does not comply with ARINC 653 specification. ARINC 653 does not allow more than one partition to be executed at the same time, so instead the only possible scenario is a single partition where the respective processes can be allocated to multiple cores.

5.2. GNSSW-LEON4

The second use case is being developed under ESA's GNSSW-LEON4 for Space activity where an on-board Software Defined Radio GNSS receiver has been implemented for the GR-740 On-board-Computer harnessing the usage of multi-core through RTEMS SMP and AIR. This real time on-board software requires predictability in order to ensure the execution of all tracking software within a temporal deadline (one second).

The application test setup is depicted in Figure 4, where the on-board software is feed with a sample data via UDP and also transmits its output for performance analysis.

This ongoing case brought the challenge of porting an already optimized SMP application into the TSP paradigm. The challenge of implementing an already optimized software into a new software architecture has been the most challenging but also most productive case

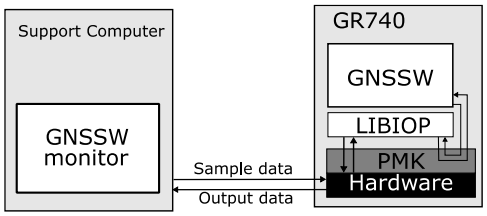

Figure 4. GNSSW simulation setup

since it has been able to push AIR to its limits. The biggest limitation is due to the required data throughput, which is inherent to the fact of being compliant to an ARINC specification. Nonetheless, it establishes a real application performance baseline for AIR to improve.

6. CONCLUSIONS

AIR hypervisor focus is to deliver an effective and simple procedure to build systems based on the IMA-SP directives. Being developed by strictly following these well-known standards, AIR is a solid solution to assist system designers and developers in quickly porting or establishing a new TSP system for space, allowing the developers to keep their focus on the application while enabling another layer of safety and possible optimization.

AIR support is in hand with industry increasing multi-core on-board computers offer by supporting new features offered by the RTOS. MORA-TSP project is an example of how AIR is able to support increasingly optimized applications and easily bring SMP into the TSP paradigm.

AIR IO solution is flexible and efficient for applications able to execute some on-board data handling. Project GNSSW-LEON4 proves AIR is prepared for the increase in computational power of on-board computers and multi-core use in the near future.

7. REFERENCES

[1] Airlines Electronic Engineering Committee (AEEC), “ARINC Specification 653-1. Avionics Application Software Standard Interface,” 2003.

[2] J. Windsor and K. Hjortnaes, "Time and space partitioning in spacecraft avionics," *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*, 2009.

[3] J. Windsor, M.-H. Deredempt and R. De-Ferluc, “Integrated modular avionics for spacecraft — User requirements, architecture and role definition,” in *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, Seattle, WA, USA, 2011.

[4] GMV, “AIR,” 01 01 2019. [Online]. Available: http://www.gmv.com/en/Products/air/. [Accessed 05 02 2019].

[5] D. SIlveira, B. Gomes, G. Sanches and L. Murta, “AIR User Manual,” 01 01 2019. [Online]. Available: https://gmvdrive.gmv.com/index.php/s/eScXCAybbecmT9b. [Accessed 05 02 2019].

[6] RTEMS, “RTEMS: An Open Real-Time Operating System,” [Online]. Available: http://www.rtems.com/. [Accessed 10 02 2019].

[7] GMV, “AIR git repository,” GMV, [Online]. Available: https://spass-git-ext.gmv.com/AIR/AIR. [Accessed 18 02 2019].

[8] C. Silva, Integrated Modular Avionics for Space Applications: Input/Output Module, Lisbon: Instituto Superior Técnico, 2012.

[9] T. Schoofs and J. Cristovão, “Sharing the costs of common tasks in the AIR partitioning operating systems,” in *Dasia 2011 - Data Systems in Aerospace*, San Anton, 2011.

[10] Edisoft, “RTEMS by Edisoft,” Edisoft - Defense & Aerospace, [Online]. Available: http://www.edisoft.pt/product-list/rtems/. [Accessed 05 02 2019].

[11] “EEMBC’s CoreMark®,” EEMBC, [Online]. Available: https://www.eembc.org/coremark/. [Accessed 5 2 2019].