Received 9 March 2023, accepted 29 March 2023, date of publication 5 April 2023, date of current version 14 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3264825

TOPICAL REVIEW

A Comprehensive Survey on the Use of Hypervisors in Safety-Critical Systems

SANTIAGO LOZANO[1,2], TAMARA LUGO[1], (Member, IEEE), AND JESÚS CARRETERO[1], (Senior Member, IEEE)
[1]Computer Science and Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain
[2]SENER Aeroespacial, Tres Cantos, 28760 Madrid, Spain

Corresponding author: Jesús Carretero (jcarrete@inf.uc3m.es)

This work was supported in part by Comunidad de Madrid Government "Nuevas Técnicas de Desarrollo de Software de Tiempo Real Embarcado para Plataformas, MPSoC de Próxima Generación," under Grant IND2019/TIC-17261.

ABSTRACT Virtualization has become one of the main tools for making efficient use of the resources offered by multicore embedded platforms. In recent years, even sectors such as space, aviation, and automotive, traditionally wary of adopting this type of technology due to the impact it could have on the safety of their systems, have been forced to introduce it into their day-to-day work, as their applications are becoming increasingly complex and demanding. This article provides a comprehensive review of the research work that uses or considers the use of a hypervisor as the basis for building a virtualized safety-critical embedded system. Once the hypervisors developed or adapted for this type of system have been identified, an exhaustive qualitative comparison is made between them. an exhaustive qualitative comparison is made between them. To the best of our knowledge, this is the first time that all this information is collected in a single article. Therefore, the main contribution of this article is that it collects and categorizes the information of each hypervisor and compares them with each other, so that this article can be used as a starting point for future researchers in this area, who will be able to quickly check which hypervisor is best suited to their research needs.

INDEX TERMS Aerospace, automotive, aviation, embedded, hypervisor, multicore, safety-critical, virtualization.

I. INTRODUCTION

Virtualization is one of the most powerful tools in the present and near future for the efficient use of modern multicore platforms. However, although it is generally accepted that the future of electronic systems is multicore technology, sectors with critical security requirements (such as the space, aviation, or automotive sectors) have traditionally been reluctant to adopt this technology. For example, even though that multicore systems began to come into existence in 2005, 2008 is the first year in which the subject is directly addressed in two articles by the American Institute of Aeronautics and Astronautics (AIAA): one on Multiple Levels of Independent Security (MILS) [1] and another dealing with the future of jet fighter mission computers [2]. Most probably, the reason behind this slow permeation of multicore technology in critical sectors is related to interference problems between cores that need to consume the same resources and how this affects the Worst-Case Execution Time (WCET) [3]. Years ago, works such as those of Kinnan [4] and Wilhelm et al. [5] already pointed out some of the problems of shared resources, how they cause variability in execution times and how this un-predictability impacts the implementation and certification of these systems. A decade later, bounding the WCET to obtain deterministic behaviour remains one of the main challenges in, for example, avionics platforms, as reflected in the work of Annighoefer et al. [6] These problems can be mitigated, in many cases, by intelligently planning architectures to improve predictability, as the works of Cullman et al. [7] and Kliem and Voigt [8] point out.

However, it is essential to address the problems that multicore processors pose, as they are becoming impossible to

The associate editor coordinating the review of this manuscript and approving it for publication was Rosario Pecora.

avoid in the present and future of critical real-time systems. In their article [9], in which they review the challenges of the future in terms of avionics architectures, Bieber et al. explain how multicore architectures have been replacing monocore architectures since the mid-2000s, so that monocore processors are progressively less common and more expensive. In addition to economic reasons, monocore processors have a ceiling when it comes to computing power. It is especially limiting the power consumption and the heat dissipated by the processing units the more powerful they are. This means that the future of avionics and other sectors of critical needs inevitably pass-through multicore processors. For this reason, software tools are needed to help take advantage of their processing capacity, maintaining high levels of safety and security. Among these tools, virtualization has emerged as one of the most powerful, gaining popularity even in technologically conservative sectors such as automotive or avionics.

To the best of our knowledge, there are not many surveys comparing virtualization solutions for safety-critical embedded systems. Gu and Zhao compared some virtualization technologies for real-time embedded systems (including, but not focused on, safety-critical systems) and discussed some of their technical issues [10]. However, the review focused heavily on work done on Xen and KVM, and it is outdated, as the article dates from 2012. Taccari et al. did a review of virtualization technologies for real-time embedded systems, but with a clear focus on those running on the ARM architecture [11], and the study is also rather outdated technologically (2014).

Recently, Cinque et al. categorised the hypervisors used in mixed-criticality systems into four groups, according to the internal structure by which they achieve virtualization: separation kernel hypervisors, general-purpose hypervisors, hardware-extensions hypervisors, and lightweight hypervisors [12]. For each group they mention several examples, list some of their features, and discuss whether those make them more or less suitable to be used in industry. However, their work clearly differs from ours on several points. The most important is that the work of Cinque et al. is not and does not pretend to be exhaustive, so it does not consider all the solutions used in industry or in research works but mentions a few that exemplify each category. In addition, the comparison between those solutions uses different parameters than ours, and focuses especially on dimensions such as certification, testing, or dependability support, in order to check the maturity of each solution for use in industry. Our survey is more exhaustive covering all hypervisors typically used in industry and research and comparing them using criteria that are not necessarily industry-focused, so that the survey can serve as a starting point for future research, even if it does not deal with certification or testing issues.

This article provides a comprehensive review of the research work that uses, or considers the use, of a hypervisor as the basis for building a virtualized safety-critical embedded system. Section III introduces the fundamentals of virtualization in real-time systems and the basic of hypervisors. Section II sets out the criteria for the selection of articles for this review. Section IV shows a study of virtualization in safety-critical embedded systems, with emphasis on aerospace and automotive industries. Sections V and VI present a comparison of some relevant safety-critical real-time hypervisors. Finally, Section VIII describes the main conclusions of this work.

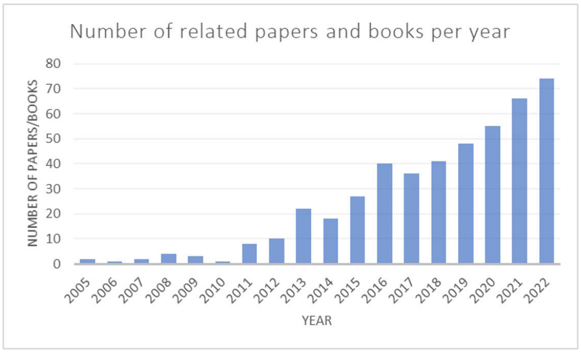

FIGURE 1. **Number of papers published each year according to the previously defined search.**

II. PAPER SELECTION CRITERIA

The interest of the research topic for the community of real-time systems is undoubted. However, as we want to restrict the overview on hypervisors to safety-critical systems, we have searched for a term consisting of the word *"hypervisor"* together with different words related to industries in which safety-critical embedded systems are developed in the most popular and relevant databases of scientific and engineering research articles: *IEEE Xplore* and *Science Direct*. Table 1 shows the results obtained.

Note that not all the articles and books resulting from the search are of interest to this survey, as some only mention the hypervisor concept in a context not directly related to this technology. However, sorting the search results in chronological order clearly shows the upward trend in the importance and depth of hypervisors in safety-critical embedded systems. After eliminating duplicate results, Table 2 presents the distribution of articles and books by year and Figure 1 shows them graphically.

These results are supplemented with searches in Google Scholar, to complement the articles found in these two large databases with the most relevant articles found in smaller databases. From all these results, only journal articles and conference proceedings are selected, discarding books, which usually collect valuable technical information but do not usually present new information that has not been previously presented in articles. Finally, among the remaining articles, a review is carried out to select only those articles that meet one of the following criteria:

- The article deals with the use of an existing hypervisor (open-source or proprietary licensed) in the context of a safety-critical embedded system.

TABLE 1. Number of papers resulting from the search of the word "hypervisor" and words referring to safety-critical sectors.

	Automotive	Aerospace	Aviation	Avionics	Safety-Critical	Spacecraft	Aircraft	Vehicle
IEEE Xplore	44	29	6	22	37	4	5	24
ScienceDirect	144	68	39	71	117	22	64	322
Total	188	97	45	93	154	26	69	346

TABLE 2. Distribution of the results of the previous search by year.

2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
2	1	2	4	3	1	8	10	22	18	27	40	36	41	48	55	66	74

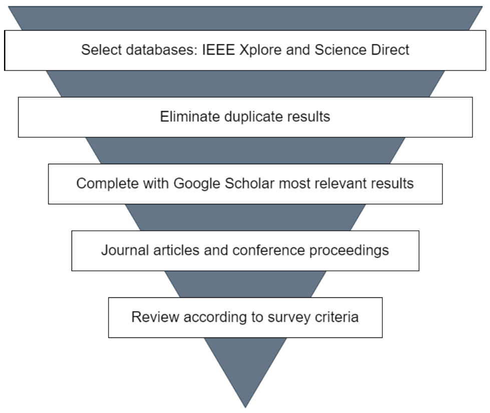
Select databases: IEEE Xplore and Science Direct
Eliminate duplicate results
Complete with Google Scholar most relevant results
Journal articles and conference proceedings
Review according to survey criteria

FIGURE 2. Articles selection criteria.

- The article presents modifications made to an open-source hypervisor for use in the context of a safety-critical embedded system.
- The article presents the development of a new hypervisor for use in safety-critical embedded systems.

The criteria for selecting the articles to be reviewed in this survey are reflected graphically in Figure 2.

III. FUNDAMENTALS

A. BRIEF HISTORY OF VIRTUALIZATION

Despite being a topic that has been booming in recent years, the roots of virtualization date back to the 1960s. Christopher Strachey is considered one of the pioneers on the subject, being the first person [13] to publish an article dealing with the concept of time-sharing at the International Conference on Information Processing at UNESCO, Paris, in June, 1959 [14]. The time-sharing technique is based on the sharing of a computer by several users at the same time. This technique would imply over time revolutionary changes in the industry, including the appearance of the concept of virtualization.

The first experimental time-shared system was accomplished in 1961 by a MIT group, led by Professor Fernando J. Corbató [15]. In 1963, the same group developed an operational version for the IBM 7094 mainframe, called CTSS (Compatible Time-Sharing System). This system would be the basis of the renowned MIT's Project MAC (*Mathematics and Computation*, later renamed to *Multiple-Access Computer*) [16]. One of the main objectives of the project was the creation of a large, multiple-access computer system, available to meet the needs of a large number of users individually. In this context, MIT contacted several computer vendors, including GE and IBM. At the time, IBM did not consider the demand for a time-sharing computer to be large enough to invest in. GE, on the other hand, committed to developing a time-sharing computer, so MIT chose GE as its supplier. In May 1964, a GE computer was used for a demonstration of a time-sharing system at Dartmouth College [17]. This probably became a wake-up call for IBM, especially when Bell Labs announced that it needed a similar system.

In response, IBM designed the CP/CMS, the first time-sharing OS that also introduced the first full virtualization platform [18]. The first version, CP-40 OS, provided an environment for up to 14 simultaneous virtual machines [19]. Although this OS was never commercially distributed, it was the forerunner of CP-67 and CP-370 versions, that were the base for the VM/370 OS, which was used with one of IBM's best-known mainframes, the System/370. Virtualization and timesharing allowed to the companies to own a single mainframe and to provide a terminal to each employee, instead of providing a computer to each of them [20].

A further development was the VM86 mode, a hardware virtualization technique that allowed multiple 8086 processors to be emulated by the 386 chip, thus allowing the execution of real mode applications that are incapable of running directly in protected mode while the processor is running a protected mode operating system [21].

In 1974, Gerald J. Popek and Robert P. Goldberg established the characteristics that a system had to fulfill to support virtualization [22]. Their article described the properties and functions of virtual machines and virtual machine monitors that we still use today. According to its definition, a virtual machine (VM) can virtualize any hardware resource, including processors, memory, and network connectivity. A virtual machine monitor (VMM) is the layer of software that provides the execution environment for the virtual machine. In their article they also described the three properties that a VMM had to satisfy:

- **Equivalence**: The environment that it provides to the virtual machines must be identical to the native hardware

that the VMM runs on. Thus, a program running on the VMM must behave in the same way as if it was running directly on the physical machine.
- **Resource control**: The VMM must have full control over system resources.
- **Efficiency**: If possible, there should be no difference between a virtual machine and a physical equivalent.

These properties are still valid today, although the term Virtual Machine Monitor is no longer so common, being gradually replaced by the term *hypervisor*.

Over time, throughout the 1980s and 1990s, as Moore's prediction continued to hold true [23], computers became increasingly powerful, cheap, and small. Personal computers appeared to replace mainframes and terminals [24], slowing the virtualization trend for some time. However, in recent years we have experienced another boom in virtualization technologies due to different reasons, some of which are:

- **Resource optimization**: The great power of today's computers means that, in many cases, they are idle most of the time since the use required of them does not consume all their resources. Virtualization allows several applications, each one even with different operating systems or execution environments, to run on the same hardware in isolation and without interaction between them. This has allowed to optimize hardware resources in several types of applications [25].
- **Isolation as a security measure**: One of the great advantages of hypervisors is the isolation among virtual machines so that, ideally, malicious activity on one virtual machine or container does not affect the rest [26] Also, virtualization provides a way to implement application redundancy without having to purchase additional hardware. If one application fails, another application (running on a different virtual machine) can take over.
- **Physical space reduction**: The fact that virtualization makes it possible to use fewer hardware resources allows to save physical space.
- **Less power consumption**: Like the previous point, this is a direct consequence of the resource optimization. The fact of being able to use fewer hardware resources for the same functionality can lead to less power consumption, and eventually to a smaller carbon footprint [27].
- **Easy migration and legacy protection**: Hypervisors, by definition, decouple the OSs and applications of the host hardware, thus benefiting the migration of virtual machines from one host to another without disruption. This is a great advantage when it comes to efficient work-balancing or when designing plug-and-play systems [28].

B. HYPERVISORS

As explained above, VMMs have their origin in the 70s, and their birth responds to very specific needs. Today, VMMs allow us to take full advantage of the capabilities of new processors, which are becoming more and more powerful. The term Virtual Machine Monitor has progressively decayed,

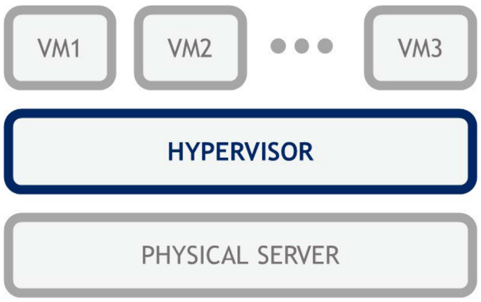

FIGURE 3. Graphical depiction of a hypervisor.

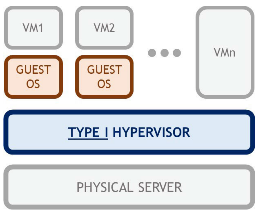

FIGURE 4. Type I hypervisor.

and today it is much more common to refer to them as *hypervisors*.

Ankita Desai et al. [29] defined a hypervisor as a thin software layer that provides abstraction of hardware to one or several operating systems, by allowing them to run on the same host hardware. Indeed, as shown in figure 3, a hypervisor is a layer of software that creates and manages virtual machines or partitions, to which it provides abstraction of the hardware. It is more debatable whether an operating system that uses the hardware virtualized by the hypervisor is necessary. As we will see later, there are hypervisors that make the virtualized hardware directly available to users, allowing them to manage it as if it were real hardware. Programming an application on these hypervisors would be equivalent to programming what is commonly known as a "bare-metal application", except that the hardware on which it runs is virtual, not real.

Hypervisors can be classified into two types depending on the environment in which they run (see Figs. 4 and 5):

- **Type I hypervisors** or bare-metal hypervisors, in which the hypervisor runs directly on top of the host hardware. In this, the hypervisor has the responsibility of scheduling and allocating system resources to each of the virtual machines, and no operating system runs below it.
- **Type II hypervisors**, in which the hypervisor runs as an application on top of a host operating system. The host OS does not necessarily have to know about the hypervisor, it treats it as any other process.

Hypervisors can also be roughly divided into two types based on the type of virtualization they offer:

- **Full virtualization** hypervisors that allow running unmodified guests operating systems. The hypervisor completely emulates the physical platform it runs on, so the operating systems running on it don't even know

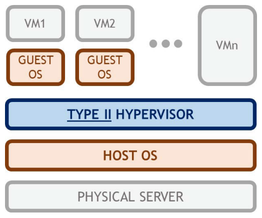

FIGURE 5. Type II hypervisor.

they are running on a virtualized platform. The great advantage of this approach is the flexibility it offers, by allowing any guest OS to run. However, it tends to carry a significant overhead, with up to 30% more latency when compared to running directly on physical hardware [30].

- **Paravirtualization** hypervisors that cannot run unmodified OSs, but the guest OS must be aware that it has been virtualized and provides special hooks to directly take advantage of the services offered by the hypervisor [31]. In other words, the hypervisor does not have to translate the instructions of each VM, but receives direct instructions from it, usually called *hypercalls*. Of course, this alternative is much less flexible, since each guest OS must be modified to work with the hypervisor on which it runs, but it has the advantage of offering higher performance in terms of access time to hardware, as theorized and demonstrated by Dordevic et al. in an article comparing the two virtualization techniques [32].

Some hypervisors, such as VMware, ESXi, are able provide both features. They support paravirtualization, but they are also able to run unmodified guests. In this case, paravirtualization is regarded as a performance optimization technique that allows significantly better performance of the guest OSs by reducing the number of abstraction layers sitting between a guest execution environment and the hardware [33].

Some modern processors offer hardware tools to achieve (ideally) full virtualization, reducing the overhead of classic full virtualization. They are enabled with the mechanisms to support the virtual machine environment. Intel provi hardware-assisted virtualization mechanisms for Intel processors. As an example, Intel VT-x provides virtualization mechanisms for processor virtualization [34], input/output memory management unit (IOMMU) provides memory protection from I/O devices by enabling system software to control which areas of physical memory and I/O device may access. [35], and Single Root I/O Virtualization (SR-IOV) that allows an I/O device to be shared by multiple Virtual Machines [36]. This particular case goes by different names: Xen calls it hardware virtual machine (HVM) [37], but it is generally referred to as hardware-assisted virtualization or accelerated virtualization.

Sometimes, this type of virtualization continues to have a too large overhead and is combined with some paravirtualized drivers, which is why it is also referred to as hybrid virtualization [38].

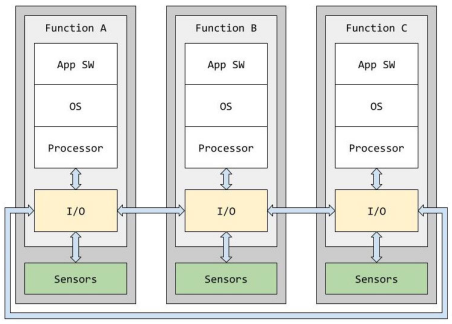

FIGURE 6. Federated architecture example.

Most modern hypervisors for real-time systems offer the possibility to specify core/task affinities, using affinity masks which specify on a per-task basis on which processors a task may be scheduled. However, core affinities restrict task migration, which may generate some problems in architectures providing symmetric multithreading (SMT), where tasks are allocated to the next free core. However, even if SMT may seem more efficient, it could lead to an increase in cache failures and, thus, in increasing in the worst-case execution time (WCET) of the tasks. Thus, it is still a common practice to map hard-real time tasks to a specific core indicating its affinity. This feature is needed to provide ARINC 653 partitioning model for safety-critical avionics RTOS. However, soft real-time tasks are usually allowed to migrate form one core to the other [39].

IV. VIRTUALIZATION IN SAFETY-CRITICAL EMBEDDED SYSTEMS

This section shows the current state of the art of virtualization in safety-critical embedded systems through the actual landscape in the major sectors using them: aviation, aerospace, and automotive sectors.

A. VIRTUALIZATION IN THE AVIATION INDUSTRY

Traditionally, flight systems have had a *federated* architecture, in which each function consists of a black box with dedicated computational resources [40] (see Fig. 6).

Each of the black boxes can contain a completely different configuration (hardware or software) inside, and they are physically isolated from each other. Naturally, this architecture is excellent in terms of fault isolation and fault tolerance, but it presents a series of serious problems: duplication of resources, lack of flexibility (adding a functionality requires adding a new box) and high cost, not only economic, but also regarding power consumption and weight. It is in this context that the concept of Integrated Modular Avionics (IMA) was born, in the early 1990s [41]. In an IMA architecture, the coexistence of different avionics functions on the same platform is pursued, without interference between them. For this, the different functions share a series of hardware resources (CPU, communications, I/O devices…) and are separated by

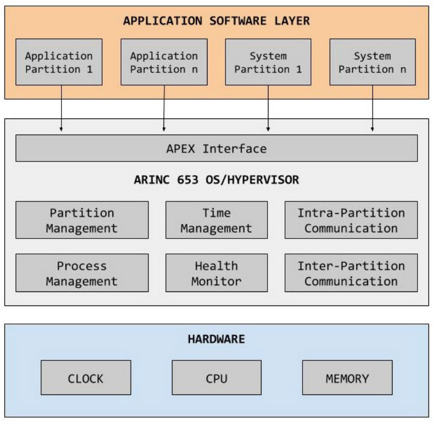

FIGURE 7. Graphical depiction of the ARINC 653 standard.

robust partitioning mechanisms inherent to the architecture itself. Today, the advantages of using IMA are not discussed in the industry and virtually every airplane model that enters service uses this philosophy [42].

One of the key enablers for this paradigm shift was the release of the ARINC 653 software standard in 1996. ARINC 653 is a software specification for time and space partitioning in safety-critical real-time systems. The first draft describing ARINC 653 was published in 1996 [43], and two supplements have since been published, the most recent being in 2007. Originally, it defined the general structure that the operating system of an IMA architecture should follow, but ARINC 653 can also be applied to a hypervisor, since some of its characteristics fit even better with it. The objective of ARINC is to specify the characteristics of a software execution environment in which several applications can run, separated from each other in virtual containers called partitions. Ideally, these containers should be perfectly isolated from each other, so that the execution or failure of one partition does not affect any other partition. The way to achieve this isolation is to separate the hardware resources spatially and temporally. The similarity between ARINC 653, in this regard, with the hypervisor concept is remarkable, and this is reflected in the equivalence between figures 3 and 7.

In addition to defining the services to be offered, one of the most important features of ARINC 653 is that it standardizes the interface between the hypervisor/OS and the application layer. This interface is called APEX (Application/Executive), and it offers several advantages in line with the IMA philosophy: portability, reusability, modularity and easy integration of software blocks.

The services offered by ARINC 653 can be divided into several modules:

- **Partition Management**: This module provides means to modify the operating mode of partitions, and it is in charge of scheduling the partitions.
- **Process Management**: Each partition can have multiple periodic or aperiodic processes. This module provides means for modifying the operational mode of processes, and it includes process scheduling.
- **Time Management**: This module ensures that hard real-time requirements are met and provides time-related services, such as reading time or wait/timeout services for processes. There must be a single time source for all partitions, regardless of their execution.
- **Inter-Partition Communication**: Communication among the different partitions is carried out through ports and channels. Conceptually, a port could be seen as a gate at the borders of a partition, while channels link two or more ports. The standard also defines two modes of operation: sampling mode (oriented to fixed-size synchronous messages) and queuing mode (oriented to asynchronous messages of variable size).
- **Intra-Partition Communication**: This module is in charge of providing communication among processes inside a partition, which is realized through buffers and blackboards. Events and semaphores are also used, to provide synchronization among processes.
- **Health Monitor**: This module is responsible for defining, detecting, and reacting to different errors at the process, partition or system level.

In their article VanderLeest et al. [44] make an interesting reflection on how the ARINC 653 standard seems to prohibit interruptions, since they can undermine the determinism of a system, allowing one partition to steal time from others. Indeed, most implementations of ARINC 653 do assume that interrupts are not allowed when following the IMA philosophy, but they reason that interrupts do not have to involve non-determinism, and they design interrupts that offer predictability via a decreasing time budget, so that the system is predictable at the scale of major time frame. They also developed a prototype to demonstrate how these interrupts work using the Xen hypervisor, although the hypervisors used in safety-critical use cases will be reviewed in more depth in section V.

The hypervisor concept fits well with the IMA philosophy and the ARINC 653 standard. Virtualization provides, by definition, time and space partitioning. The rest of the functionalities that ARINC 653 describes (for example, health monitoring or the APEX interface) must be implemented, either in the hypervisor itself or at other levels of the software architecture (typically, the application interfaces, such as APEX, POSIX or OSEK are covered at OS level). In a 2015 article [45], VanderLeest et al. conclude that hypervisors are the tool that will allow the aviation industry to firmly adapt to multicore systems, which are the only way to increase processor performance. Moreover, ARINC 653 is also one of the most powerful candidates to standardize the space industry as well, as the requirements of the civil aviation world that prompted the definition of the standard are also applicable to the space industry. The adoption of ARINC 653 would bring benefits in terms of cost reduction, modular certification, and less integration effort.

The DO-297 standard (IMA Development Guidance and Certification Considerations) is the document used by certification authorities such as the FAA and EASA to approve aviation IMA systems [42]. In turn, this document recommends ARINC 653 to define the interfaces and specify the behaviour of the system.

B. VIRTUALIZATION IN THE AEROSPACE INDUSTRY

The space industry has always been closely related to the aviation industry, so it is reasonable that there are numerous articles that discuss the possibility of applying aviation standards and practices to the aerospace industry. An example is the article by Windsor and Hjortnaes, in which they analyze the advantages of incorporating TSP techniques into the spacecraft avionics architectures based on the IMA aeronautical concept and the ARINC 653 standard [46]. They also consider the areas where these techniques could have the most impact and give different examples of use cases.

Actually, in some respects the operation of space systems is even more delicate than that of military and commercial aircraft. The success of space missions depends on being able to obtain deterministic behaviour over long periods of time and, in many cases, with very limited real-time support from the ground operating teams. In addition, the environment in which a space system operates can be significantly more hostile than that of other critical real-time systems, due to factors such as vacuum, radiation or extreme temperatures. All of this makes the processes to design, develop and test space equipment time-consuming and costly. Likewise, the avionics in such equipment often consist of old and robust components, often created specifically for use in the space industry (such as LEON processors [47]), which have proven their reliability over years of successful missions.

For the same reasons as in aviation (increased complexity of software applications, increased processing capacity of hardware platforms, promotion of interoperability and reusability…), in recent years there is also a trend in the space sector towards COTS components, in order to take advantage of the powerful resources they offer, while reducing development costs, power consumption and physical space on the spacecraft. This has led to the emergence of a new space market, commonly referred to by that name: *New Space* [48]. This new market is based on the development of small space platforms that have a significantly lower associated cost than classic space missions and, therefore, less stringent reliability requirements. Both the reduction in cost and the openness to commercial components have led New Space to welcome private companies [49], something not common in the space industry, which has traditionally been driven by public organizations. The entry of private companies has considerably increased the number of players in the industry, which has boosted its competitiveness and the emergence of new low-cost, high-performance applications such as space debris removal, Earth observation missions or satellite-based global communication networks.

It seems clear that there is a problem in combining all or most of the functionality of a space system on a single high-capacity hardware platform: there are safety-critical functionalities whose failure would have a catastrophic impact on the mission objective, while other functionalities are not so critical and can therefore be subject to less demanding and less costly verification and validation processes. In this context, virtualization is a key technology since it guarantees by its inherent characteristics the temporal and spatial separation of different software modules, so that there is no interference between them and the failure of one does not affect the others. When virtualization is used to deploy functionalities of different criticality levels on the same platform, the resultant systems are usually referred to as mixed-criticality systems [50].

C. VIRTUALIZATION IN THE AUTOMOTIVE INDUSTRY

Since the appearance of the first electronic ignition systems in the 1970s, the number of electronic components in cars has constantly increased. Whereas previously mechanical systems accounted for most of the complexity of cars, electrical and electronic systems have become more sophisticated over time [51]. In fact, it is currently estimated that 35% of the cost of a vehicle corresponds to electronics, greatly exceeding the 20% that it supposed in 2000 or the 30% that it supposed in 2010, and it is estimated that this figure will continue to grow up to 50% by 2030, according to the report made by Winkelhake [52]. Today, most vehicles are based on a powerful electrical and electronic (E/E) architecture on which the engine, braking, steering and other comfort and safety features depend. Embedded software has begun to gain paramount importance in automotive design and development, and it will continue to gain importance as companies continue to develop the technology, especially in areas such as autonomous driving. This means that the number of ECUs (Electronic Control Units) is increasing in modern vehicles: a current car has more than 150 ECUs for different purposes [53]. Faced with this challenge, diferent solutions have been proposed in recent years. Among these, virtualization is one of the most studied, since it would allow combining several ECUs on the same hardware platform, maintaining a temporal and spatial separation between them. This separation is essential, since critical systems such as driving assistance, which are usually managed by safety-certified RTOSs, coexist in the same vehicle with information and entertainment systems, which have significantly more lax safety constraints [54]. Using virtualization, the functionality previously distributed among many ECUs can be collected in a smaller number of DCUs (Domain Control Units), each system adhering to its own safety requirements [55].

Although not as direct as between the space and aviation sectors, there are many similarities between these and the automotive sector: Gaska and Chen [56] make an interesting analogy between the architecture of an automotive system and an IMA avionics architecture. They discuss how multicore processing, along with hypervisor technology, are key enablers of the future of these types of systems, and propose various candidates for both multicore platforms (Intel Xeon FPGA SoC, NVidia Tesla SoC or Xilinx Ultrascale+) and hypervisors (VxWorks, Lynx or Green Hills). However, the article does not show any type of test or deep analysis that allows us to opt for any of the alternatives.

Cruz et al. proposed in a recent paper [57] an x86-based virtual PLC (vPLC) architecture that decouples the logic and control capabilities from the I/O components, while virtualizing the PLC logic within a real-time hypervisor. In vPLC the dedicated PLC I/O bus is replaced by a deterministic and high-speed networking infrastructure, using SDN to enable the flexible creation of virtual channels on the I/O fabric. This contribution could have significant impact in automotive manufacturing as it allows to achieve high-speed, deterministic I/O communication for the hypervisors and to oversee partition behavior by software, thus pushing virtualization of IoT devices and their control using hypervisors.

V. HYPERVISORS FOR SAFETY-CRITICAL REAL-TIME SYSTEMS

Having explained the basis for understanding the hypervisor concept and some of the reasons why virtualization not only has a place but may play a fundamental role in the future of several safety-critical industries, the following section reviews the scientific works that have applied virtualization of some kind or have used hypervisors to study its impact on space, aviation or automotive systems.

For this survey, hypervisors have been divided in proprietary and open-source, following their license policy. Section V-A will introduce some hypervisors that have great importance and presence in the industry but, due to their proprietary licensing and high price, are hardly studied in academic works. Next sections are devoted to the three main open source hypervisors, Xen, KVM, and XtratuM, because of their importance and the greater existence of evidence both in scientific literature or through dissemination by private companies. Other less popular open-source hypervisors are grouped together in section V-E.

A. PROPRIETARY HYPERVISORS

Note that although a hypervisor inherently offers TSP features, it is not the only option to implement them. A RTOS can offer TSP features if it is able to fully isolate software modules, so each of them can run different criticality partitions. For example, there are several proprietary RTOS, oriented to safety-critical real-time industrial applications, that offer TSP features and have already been certified according to standards such as DO-178C or ISO 26262, so that the importance of complying with them is understood when the developed software is actually to be used in real use cases). It is the case of WindRiver VxWorks, Green Hills Integrity, LynxOs, SYSGO PikeOS, JetOS or DDCI Deos. In some cases, given the increasing complexity of these types of applications and the increase in processing power in current hardware platforms, they have ended up including virtualization capacity, or resulting in the appearance of new virtualization products from the companies that develop them. These products are:

- WindRiver Helix
- Green Hills Integrity Multivisor
- LynxSecure Separation Kernel Hypervisor
- SYSGO PikeOS
- DDCI Deos
- RTS Hypervisor

These hypervisors have a proprietary license and are aimed at critical real-time systems such as aviation or automotive, so they are quite expensive. This makes their use confined almost exclusively to commercial purposes, so there is little published literature on them. Therefore, the information compiled in the following table, which compares each of these options, has been compiled mainly from sources provided by the developers themselves. In addition, it must be considered that these hypervisors are all developed by large companies and are direct competitors, which is why they offer similar functionality in features such as inter-partition communication, real-time support or safety and security services. Therefore, a comparison between the hypervisors listed above is made below, based on criteria by which they differ:

- **Hypervisor Type**: As explained in section III-B, hypervisors can run directly on the hardware (type 1 or bare-metal) or on an OS host (type II).
- **Supported HW Architecture**: Although it is not an exhaustive analysis (the compatibility case with each different processor or SoC could be studied), the processor architectures supported by each of the hypervisors are listed.
- **Virtualization Type**: As explained in section III-B, hypervisors use different virtualization techniques that can be roughly grouped into two types: paravirtualization and full-virtualization.
- **Supported guest OSs**: The different operating systems that can be run as guests on each of the hypervisors. Note that, in the cases of hypervisors that offer full-virtualization, any operating system can be run unmodified, although it is usually more efficient to run a paravirtualized operating system if possible.
- **Nationality of the developer**: This is an important aspect to take into account, since there may be cases in which an original equipment manufacturer faces restrictive regulation when it uses the product of a company from another country, in the event that there is competition between both countries (as sometimes happens with the USA, Russia and many European countries).

The comparison is reflected in Table 3.

B. XEN

Xen is a type-1 hypervisor, originally developed by the University of Cambridge Computer Laboratory, that is now being developed by the Linux Foundation, with support from Intel.

TABLE 3. Proprietary hypervisors comparison.

	Hypervisor Type	Supported HW Architectures	Virtualization Type	Supported Guest OS	Developer Nationality
WindRiver Helix	Type 1	ARM x86	Full Virtualization	Any unmodified OS	American
Green Hills Integrity Multivisor	Type 2	ARM x86	Full Virtualization	Any unmodified OS	American
LynxSecure Separation Kernel Hypervisor	Type 1	ARM PowerPC x86	Paravirtualization Full Virtualization	Any unmodified OS (fully virtualized) LynxOS and Linux (paravirtualized)	American
SYSGO PikeOS[a]	Type 1	ARM PowerPC x86 SPARC	Paravirtualization Full Virtualization	Any unmodified OS (fully virtualized) PikeOS and Linux (paravirtualized)	German
DDCI Deos[a]	Type 2	ARM PowerPC x86	Paravirtualization	RTEMS	American
RTS Hypervisor	Type 1	x86	Full Virtualization	Any unmodified OS	German
OpenSynergy's COQOS Hypervisor	Type 1	ARM	Full Virtualization	Any unmodified OS	German

[a]Note that in the case of PikeOS and Deos, the same product offers typical RTOS functionality and virtualization capabilities.

Xen is free and is one of the few type-1 hypervisors that is available as open-source. It is also, by far, the hypervisor on which most literature and research are based, so this section is dedicated to briefly describing its characteristics and analyzing the results of the main research works with respect to Xen, especially those related to real-time safety-critical systems.

As other hypervisors do, Xen allows to run many instances of an OS (or different OS) in parallel on a single machine. A running instance of a virtual machine is called a domain or guest. However, although the hypervisor is the first program running after exiting the bootloader and runs directly on the hardware, it needs a special first domain (called *domain 0* or *dom0*) that has specific privileges and is responsible for controlling the hypervisor and starting other guest domains. These other domains are called *domUs* (because they are *unprivileged* domains, in the sense they cannot control the hypervisor or manage other domains). *Dom0* has direct access to the hardware, so the hypervisor does not contain device drivers. Instead, the devices are attached to *dom0* and use standard Linux drivers. *Dom0* can then share these resources with the rest of the domains. Although Xen is not as vulnerable to single-point failures as a type II hypervisor (in which the host OS crash automatically causes the guest OS to crash), the fact that most physical resources reside on *dom0* means that, if there is a failure in *dom0*, the rest of the system will lose communication capabilities, both externally and between domains. Even if their functionality is degraded, the *domUs* could remain operational even after the failure of *dom0*, but any additional failure in the *domU* would be unrecoverable, since *dom0* would not be able to reboot it. You could return the system to its initial configuration by rebooting *dom0* but obviously doing so would interrupt system availability for mission-critical applications. There are methods to mitigate this problem by enhancing the autonomy of *domUs*. For example, a *domU* can be given direct hardware access through Xen's pass-through virtualization feature.

Xen was originally developed for x86 processors in 2003, and for that architecture it provides both paravirtualization and full virtualization. The porting of Xen to ARM involved major changes in its architecture and, as a result, its code for ARM architectures is considerably smaller than that of x86 architectures but, as a main limitation, it stands out that Xen for ARM only supports paravirtualization [58].

After this brief explanation of Xen, the most important works that use this hypervisor, especially those that apply it to safety-critical embedded systems, are listed below.

VanderLeest and White [45] use Xen over the Zynq UltraScale+ MPSoC as a case study. Although the Zynq Ultrascale+ contains a quad-core A-53 processor, as a possible solution to the problem of shared resources, they propose a simplification in which different partitions do not run simultaneously on different cores. This prevents interference in, for example, access to the L2 cache or memory bus bandwidth. However, this simplification does not allow the efficient use of the Zynq Ultrascale+ large processing capacity. In the event that more than one partition needs to access the same I/O resource, the article proposes two ways to do it: via software (although this can be a bottleneck, limiting bandwidth or latency) or via hardware, implementing an arbitrator in the FPGA of the MPSoC. It must be taken into account that the certification of arbitration logic must be done at a level equivalent to the highest level of criticality of any of the serviced guests (if implemented in software, in accordance with the DO-178C standard; in case of being implemented in hardware, in accordance with the DO-254 standard). The possibility of using the LynxSecure Separation Kernel Hypervisor and the Mentor Graphics Multicore Framework is also discussed in the article, although neither of these options is analyzed in depth.

In their article [59], Daniel Sabogal and Alan D. George explain the development of a framework called Virtualized Space Applications (ViSA). ViSA leverages the capabilities of the Xen hypervisor to provide a safe environment (software-based fault tolerant) on the Zynq Ultrascale+ and improve the dependability and availability of a flight system. The framework manages to improve the system in these aspects, but presents problems when the APU of the Zynq Ultrascale+ is irradiated. In addition, there is work to be done to solve one of the main problems of Xen-based systems, which is the dependency on dom0. Currently, no article has been published that continues this work.

In 2010, DornerWorks introduced a prototype implementation of the ARINC 653 standard, extending the Xen hypervisor. In a 2013 article, VanderLeest and other DornerWorks researchers explain these ARINC 653 extensions made on Xen a little more in depth and name the resulting hypervisor ARLX (ARINC 653 Real-time Linux on Xen). A few years later, ARLX was renamed Virtuosity, and VanderLeest announced that they were adapting the hypervisor to FACE conformance [60]. Today, Virtuosity remains an open-source hypervisor and DornerWorks benefits from it by offering maintenance and support. Virtuosity presents the limitations of any Xen-based hypervisor, especially in terms of certification: m the control partition (dom0) runs Linux as a guest OS. Despite its widespread use, it is very difficult to certify Linux according to the most demanding aviation or automotive standards, due to the little documentation on some of its main components, as well as the great effort involved in certifying an operating system so large.

Kistijantoro and Gilbran extended the ARLX partition scheduler to use the primary-backup scheme, so that the scheduler can guarantee certain services even in the event of partition failure, using backup partitions [61]. Although they demonstrated that this method improved the overall reliability of the system, it still presented several significant problems: it resulted in an unacceptably high maximum latency, the scheduler was not able to autonomously detect the failure of a partition (it depended on the partition being capable of reporting it) and did not take into account the deadline of each process within a partition.

Bijlsma et al. [62] propose a safety mechanism for autonomous vehicles that, among other tools, uses Xen to isolate software modules in the same SoC. During their experiments, using Xen's Null scheduler, they measured the time it took for the hypervisor to shutdown a faulty VM, and found that it was much longer than the time it took to pause it, so they opted for this option. Even so, although the mean time in 2000 experiments was 55us (acceptable to avoid a collision and prevent the fault from propagating), they obtained outliers of up to 1.5ms. This large variation indicates that a more deterministic scheduling is needed in the hypervisor, to be able to use it in such critical functionality.

Karthik et al. used Xen to offer an integrated cockpit solution, in which four different automotive systems ran on the same heterogeneous SoC [54]. However, it should be noted that Xen was used (on an ARM Cortex-A15 processor) only to run three of those systems, which used Linux and Android and were not safety-critical. A microcontroller (ARM Cortex-M4) and an RTOS were used exclusively to run the other system, which was safety-critical, without any kind of virtualization.

Xen on ARM has also been used to integrate Linux with a real-time operating system such as ERIKA OS, which obtained the OSEK-VDX certification for automotive applications, on the same platform [63]. However, this option still has many limitations in terms of certification: the deployment architecture is very specific (a dual-core platform in which each OS runs on a different core) and ERIKA OS can run only as a guest domU, so it depends entirely on the privileged domain dom0, which runs a general-purpose Linux. As in previous examples, there is also the challenge of certifying the Xen hypervisor itself, which does not seem approachable in its current form for meeting DAL A/B/C according to the DO-178C standard or an automotive-grade standard such as ASIL.

Recently, Schulz and Annighöfer conducted an empirical study to test the suitability of Xen to operate on safety-critical real-time systems [64]. In their experiments they obtained some promising results, but in some scenarios they observed unpredictable behaviour in terms of latencies and execution times. Although further research is needed, they conclude that Xen is not realistically feasible for such systems in its current state.

C. KVM

KVM is an open-source hypervisor originally released in 2007 for the x86 architecture and, since 2012, ported to the ARM architecture. KVM is integrated into the Linux kernel (since its version 2.7.20 for x86 and its version 3.9 for ARM), so it takes advantage of a large part of its functionality, such as memory management or CPU scheduling. In fact, as Dall and Nieh explain [65], although in x86 KVM resides entirely in the kernel, in ARM it is divided into two parts: one (called Highvisor) that resides in kernel space and corresponds to most of the hypervisor's functionality, and another (Lowvisor) that resides in Hyp mode and is in charge of enforcing isolation and performing the context execution switches between VMs and host. This is because trying to implement KVM entirely in the host kernel would have involved a series of modifications on the kernel that would have been negative in terms of performance and portability. A big advantage of this approach is that porting KVM from one ARM platform to another is easier than with a bare-metal hypervisor like Xen. Since ARM platforms are not very standardized and it is very common for them to support a version of Linux higher than 3.9, this advantage is key when compared to Xen [58].

KVM does not offer virtualization of hardware devices but relies on external tools that run in user space, such as QEMU. Together, KVM and QEMU allow running unmodified guest OSs [66]. However, to avoid the overhead that full-virtualization implies, there is also the possibility of running paravirtualized OSs using Virtio [67].

There are numerous papers analyzing Xen and KVM performance overhead on x86 architectures [68], [69] [70], but only a few that do so on ARM architectures and using embedded systems. Among them, perhaps the most complete is that of Raho et al. [71], who make a comparison in which they also include container technology (Docker, in particular). The conclusion they reach is that the overhead performance of any of the solutions is very small, with slight differences depending on the test run. They also analyze how KVM is more easily portable than Xen and how Docker, although fast and easy to deploy, is a less secure alternative to hypervisors, because hypervisors use hardware extensions to offer greater isolation (VMs do not share kernel space, while containers do). However, in a recent paper Müller et al. measured the overhead of KVM on a self-driving car-oriented Nvidia Drive AGX SoC and concluded that KVM produces too high an overhead in this particular case, which makes it unusable in real-world use cases [72].

D. XTRATUM

XtratuM [73] is a type-1 hypervisor originally developed by researchers at the Universidad Politécnica de Valencia and currently maintained by the Spanish company fentISS. XtratuM is targeted to real-time safety-critical systems, especially in the space sector [74], being designed based on the ARINC 653 standard. Currently, it supports Linux, RTEMS and LithOS (an operating system from the same developers) as paravirtualized guest OSs and allows to run bare-metal partitions using XRE, a minimal execution environment offered by the hypervisor itself. Unlike Xen, it uses no control partition (dom0): the hypervisor manages the partitions and its communications, IRQs and HW I/O access. The latest versions of XtratuM support SPARCv8, ARMv7 and RISC-V architectures [75].

XtratuM can be downloaded under the GNU General Public License, although fentISS also offers a commercial version called XtratuM Next Generation (XNG). This version is currently the most widely maintained by developers and, although it offers similar functionality to the GPL version, its internal structure is significantly different.

ESA and CNES are two of the main promoters of XtratuM, financing projects and research in which its development is continued and the possibility of using it in space missions is being evaluated [76]. However, the ARINC 653 oriented nature of XtratuM makes it a good candidate for aviation applications as well, although as of today the costly process to certify it according to the corresponding safety standards (such as DO-178C) has not started. Efforts have also been made to use XtratuM in automotive systems,such as the work carried out in [77] to design of criticality-aware scheduling for advanced driver assistance systems

Although it is a hypervisor widely used in private industry and, for competitive reasons, companies are sometimes not interested in disseminating knowledge about it, there are a few academic papers that mention or use XtratuM on safety-critical embedded systems. These are described below:

Larrucea et al. define a series of characteristics that a safety-critical hypervisor should meet to comply with the IEC 61508 standard, defined by the International Electrotechnical Commission, which covers the functional safety of electrical, electronic and programmable electronic equipment [78]. In the same article, they map the defined features to the XtratuM capabilities, demonstrating how it successfully covers them.

Muttillo et al. carried out a series of tests, on different hardware platforms that used both the LEON3 and LEON4 processors, in which they demonstrated that the performance of XtratuM competes with that of a highly proven hypervisor like PikeOS, improving it in some aspects (such as timing and memory access), although it is somewhat less predictable in the overhead introduced [79].

Researchers from the Korea Aerospace Research Institute made the effort to port a version of RTEMS that would support Symmetric Multi-Processing (SMP) on top of XtratuM [80], [81]. Currently, fentISS has released versions of XtratuM with which bare-metal partitions can be run on the hypervisor in SMP, both on LEON4-based boards and on boards based on the Zynq-7000 SoC. In addition, it is developing a BSP that will allow to run a Linux SMP guest OS on Zynq-7000.

Campagna et al. [82] presented a prototype of an architecture in which the XtratuM hypervisor runs on a LEON3 processor. As they describe in their article, they run three partitions on top of XtratuM: two of them running a number-crunching application and a checker partition that checks the outputs produced by the other two. The purpose of the paper is to study the solvency of their solution to the injection of errors. The failure model they assume is Single Event Upset (SEU), which models the impact of ionizing radiation on the processor as a result of a memory bit flip. They show that using a hypervisor is an effective method for task segregation and scheduling, as well as error detection. The use of the hypervisor has an overhead close to the minimum possible overhead and it is capable of detecting 96.2% of SEU faults.

During a study on the robustness of the separation kernels, XtratuM was used on a LEON3 as a use case and 9 notable vulnerabilities were discovered that had not been detected during the validation campaigns of the hypervisor development [83]. This not only demonstrates the effectiveness of the error injection method used, but it was a considerable help to further strengthen XtratuM, as part of the hypervisor developer team was involved in the experiment.

XtratuM is also one of the foundations of XANDAR, a project that aims to provide a toolchain for developing safety-critical embedded systems [84]. The toolchain, developed by a large group of partners from industry and academia, has been tested in both avionics and automotive use cases.

Onaindia et al. propose an architecture oriented to real-time systems that monitors and it is able to reduce system power consumption [85]. The lowest-level component of this architecture is a hypervisor. For the prototype, because of its features and its support for Xilinx Zynq-7000 SoCs, XtratuM is used (and extended) as hypervisor, and the prototype is tested on two use cases of avionics and railway systems. XtratuM has also been used, on a representative computer system used in avionics, as the basis for building a feedback control mechanism implemented at the hypervisor level [86].

E. OTHER HYPERVISORS

Some research papers using or developing hypervisors other than Xen, KVM or XtratuM are listed and briefly described below:

Missimer et al.'s **Quest-V** [87] uses hardware virtualization for safe and secure resource partitioning, offering partitions (called sandboxes) that can run their own operating system, called Quest, or Linux. However, the work is currently discontinued and does not support ARM multicore platforms with hardware virtualization.

Rodosvisor is a type 1 hypervisor developed by Tavares et al. It supports paravirtualization and full-virtualization and it is inspired by the ARINC 653 standard, but it is not fully compliant with it, since it implements the services defined by the standard, but not strictly following the corresponding API. In the experiments in the article, two bare-metal (OS-less) partitions are deployed, and a third partition runs RODOS, an RTOS for embedded systems. However, apart from one article enhancing the hypervisor for integration into the POK operating system in 2016 [88], there are no other known articles describing a continuation of the work, porting the hypervisor to other HW platforms, or supporting new guest OSs.

Pinto et al. presented in 2016 a hypervisor called **RTZVisor** (Real Time TrustZone-assisted Hypervisor) oriented to space applications, which used the ARM TrustZone to provide virtualization on a Xilinx Zynq platform [89]. A few months later, they introduced two extended versions of the hypervisor, called μRTZVisor [90] and SecSSy [91], designed to increase the safety and security of its predecessor. These hypervisors have some interesting features, such as the ability to run different almost unmodified guest OSs, but they have other important limitations: they disable the caches and MMUs of the guest OSs, they are limited to ARM processors offering ARM TrustZone technology and, despite being tested on a multicore platform (Zynq ZC702), they do not support multicore processing, so they use only one of the processor cores.

Although RTZVisor is probably the most advanced of its kind, it is neither the only, nor the first hypervisor to be based on ARM's TrustZone technology. Winter proposed in 2008 a method that, using TrustZone, provided a virtualization framework and implemented a prototype in which he deployed a non-secure guest in a secure Linux environment [92]. Cereia and Cibrario carried out a similar exercise, implementing a virtualization layer that allowed to deploy an RTOS and a guest OS, pointing out some of the limitations that ARM TrustZone imposed on them at the time: it only allowed the execution of two OSs and, while the guest OS did not it could access or interfere with the RTOS, it did not work in reverse, so it would not support two secure RTOS [93]. These limitations are shared by ARM TrustZone-based hypervisors proposed in later work, such as the Secure Automotive Software Platform by Kim et al. [94] or the open-source Xvisor presented by Cicero et al. [95] **VOSYSmonitor**, from Virtual Open Systems, also allows parallel execution of a secure partition (running an RTOS) and a partition without real-time guarantees (GPOS), but has the particularity that it allows the non-critical partition (GPOS) to use another hypervisor (such as Xen or KVM), so it could be argued that it also supports multi-guest OS [96]. VOSYSmonitor gives full priority to the RTOS, allowing the GPOS(s) to run when there are no active tasks on the RTOS.

Dasari et al. conducted a series of experiments with the ETAS Lightweight Hypervisor (**LWHVR**), a commercially viable solution in the automotive industry, which they extend by implementing Reservation Based Scheduling (RBS) [97]. ETAS LWHVR is a hypervisor oriented to multicore microcontrollers, with a low overhead and memory footprint. One of the cores works as a leader, and in it runs all the SW that has direct access to the HW. Different VMs can run in the rest of the application cores. This architecture makes the use of this hypervisor not viable in monocore systems and, probably, inefficient in the case of processors with few cores (such as dual-core processors).

Jailhouse is a simplicity-oriented hypervisor based on Linux. The hypervisor is implemented as a Linux kernel module, just like Xen or KVM. As Ramsauer et al. [98] explain, it does not perform any kind of scheduling, and simply provides static partitioning, directly allocating hardware resources to each partition. This has the advantage that legacy applications can be run with no active hypervisor overhead and simplifies certification efforts. For this reason, among others, Jailhouse is the hypervisor chosen to cement a computing platform called SELENE, which aims to serve as a basis for developing different safety-critical applications, from flight applications to autonomous robotics [99]. However, the fact of this hypervisor being based on Linux implies other complications in terms of safety and certification, such as the fact that it requires other software elements (UEFI Firmware code or bootloader, for example) that must be considered in the certification process. In addition, it still has limitations in essential aspects such as communication between VMs, for which it does not offer end-to-end timing guarantees. Some of these issues may be faced during the development of SELENE, which is scheduled to end in December 2022. Boomerang [100] is another proposal that leverages the features of a hypervisor to develop a system in which to run a critical partition along with a non-critical guest OS.

Bao [101] is also a proposal based on this concept of static partitioning, in which the hypervisor is freed from resource management, once the CPU cores, memory or I/O devices have been assigned to each guest OS. One of the limitations of this simplistic approach is that the number of guest OSs is limited by the number of physical CPUs, unless other virtualization technology runs on top of the static partitioning hypervisor.

OKL4 is a popular Type I hypervisor developed by Open Kernel Labs (the company was acquired by General Dynamic Mission Systems in 2012, and the hypervisor is no longer open-source), intended to be deployed in embedded systems. It is especially popular in the mobile phone industry (estimated to have been deployed in hundreds of millions of them [102]) and is capable of paravirtualizing various high-end operating systems, including Linux, Windows and VxWorks. In addition, it can offer a simple POSIX interface by itself, so it is able to function as a minimal OS for the implementation of safety or security critical applications [103]. Although it does not support full-virtualization, OKL4 is able to take advantage of the virtualization extensions of some ARM processors to reduce the effort required to paravirtualize an OS.

seL4 is also a microkernel of the L4 family available, at different levels of maturity, for ARM, x86 and RISC-V architecture processors. Its development started in 2006, with the intention of providing a basis for secure, reliable and safe systems. The kernel is open source, available under the GNU GPL v2 license, and most of the libraries and tools are under the BSD 2 clause. seL4 can run standalone as an OS with TSP capabilities, but it can also be configured as a bare-metal hypervisor on which, in addition to running native applications, Linux virtual machines can be deployed [104].

Another open-source hypervisor geared towards having a small footprint for its use in embedded systems is **NOVA** [105]. According to its developers, the hypervisor is the base of every other component of a system that uses it, so it should be as small and trusty as possible. However, unlike OKL4, NOVA offers full virtualization, thus, as explained above, it results in a slightly more complex and less efficient, yet more flexible hypervisor.

ACRN [106] is a lightweight hypervisor oriented to IoT and embedded systems. Although its architecture makes it quite flexible and allows multiple non-safety-critical VMs to be deployed, the number of safety-critical VMs is limited to two at best. Another important limitation is that it is based on Intel virtualization technology, so its supported HW is limited to some processors from this manufacturer. Among the guest OSs that it supports we can find Ubuntu, Android, Windows, as well as others more interesting in terms of safety, such as VxWorks and Zephyr.

Elektrobit also offers its hypervisor implementation, called **Corbos**, of which there is not much published information. It is known to be a microkernel-based hypervisor that allows at least Linux partitions to be deployed alongside other safety-critical partitions. Corbos is mentioned in an article by Lampka and Lackorzynski, to exemplify an automotive architecture that uses virtualization to harness the computing power of an ECU, without sacrificing the safety and security of the most critical software [107].

In 2014, Kim et al. introduced a hypervisor geared towards critical real-time systems, called QPlus-Hyper [108]. This hypervisor allowed the execution of an RTOS and a GPOS on the same platform, using the virtualization extensions present in some ARMv7 cores. However, there is no evidence that the development of this hypervisor has been continued since 2015, when this hypervisor was used to carry out a proof of concept that investigated how a GPU that is shared between several guest OSs could be virtualized Other than that, it is only briefly mentioned in a 2019 article that discusses cache-interference issues on clustered multicore platforms [109]. For this reason, and due to the lack of public information on the internal structure of the hypervisor, it has been decided not to take it into account in the comparison.

Reinhardt and Morgan evaluate a type I hypervisor called RTA-HV, developed by ETAS Ltd and aimed at efficient use of resources in multicore systems. In their research, they paravirtualized guest OSs on the Infineon AURIX TC27X platform [110]. As highlighted in the article, a non-intrusive hypervisor has the added advantage that it acts as an abstraction layer, which encourages software reuse, facilitating porting to another platform. However, in the work presented in their article there are certain limitations, mainly due to the low virtualization support on the part of the HW used. For example, they did not achieve complete temporal isolation between partitions and could only do a one-to-one mapping between partitions and CPU cores. Despite this, they analyze and reason that hypervisors are a good solution to the problem of consolidating different systems in the same ECU.

Manic et al. [53] used Blackberry's QNX hypervisor to deploy two virtual machines, with different OSs, on the same ECU. The objective of the experiment was, in addition to being able to carry out independent processing in each of these virtual machines, to demonstrate that they can share a single graphic display without affecting either the performance or the safety of the vehicle. QNX is a Type I hypervisor certified to ISO 26262 ASIL D (in addition to the industry standard IEC 61508 SIL 3). It supports both safety (QNX Neutrino RTOS and QNX OS for Safety) and non-safety (Linux or Android) guest OSs, and can be deployed on the latest ARMv8 and x86-64 SoCs.

Lemerre et al., from the Atomic Energy and Alternative Energies Commission, extended the RTOS **PharOS**, adding a paravirtualization layer that allowed it to run Trampoline (an OSEK/VDX-compliant RTOS) partitions, which run as time-triggered tasks within PharOS [111]. Configured in this way, PharOS can be considered a type II hypervisor.

HTTM is a relatively recent type 2 hypervisor that offers full virtualization on MIPS architectures, so there is no need to use any hardware-specific extensions [112]. Its main weakness in terms of its application in safety-critical systems is that it must run on a Linux host, which limits its ability to deliver real-time performance, and that it currently only allows the creation of a guest VM. Even so, the work is still evolving, and there are recent papers evaluating and trying to improve the efficiency of HTTM [113]. Although it is theoretically open-source, we have not been able to find the source code in any repository and we do not know the license under which it is distributed.

Minos is an open source type 1 hypervisor that allows multiple VMs to be deployed on SoCs based on the ARMv8-A architecture. Through paravirtualization, it can host several Linux, Android and Zephyr guest VMs, and is oriented to IoT and embedded devices. There are hardly any research papers mentioning Minos, but details about its internal structure and source code can be consulted through its Github repository [114].

There are a couple of initiatives that propose hypervisors targeted at SoCs that cannot be considered Type 1 or Type 2, since they are implemented as another hardware module (implemented in the SoC's FPGA), rather than as a software layer. Developers of these hypervisors are often referred to as type 0 hypervisors. Janssen et al. are the first to coin this term [115], although they did not go so far as to define a complete hypervisor, but rather a prototype running on a Microblaze core that allows bare-metal applications to be deployed isolated from each other. Jiang et al. developed BlueVisor, which does achieve this "type 0 hypervisor" in which all its components run in hardware, and allows paravirtualized guest OSs (FreeRTOS, uCOS-II and XilKernel) to be deployed on softcore processors at the highest privileged level [116]. However, due to the current immaturity of these initiatives and because their nature is different from that of the hypervisors reviewed in this survey, we leave these type 0 hypervisors out of the comparison.

VI. SAFETY-CRITICAL REAL-TIME HYPERVISORS COMPARISON

Having analyzed the main hypervisors that have been applied to safety-critical embedded systems, and according to the information gathered from the cited sources (mainly academic research papers), this section compiles their characteristics and compares them.

There are different characteristics according to which hypervisors can be compared. Based on user requirements and their knowledge of embedded systems, Hamelin et al. propose a set of practical criteria by which any potential user could select the most suitable one for their application or system [117]. Since in section V virtually all hypervisors for use in a safety-critical real-time embedded system have been reviewed, it is interesting and potentially useful for future research to classify these hypervisors according to some of the parameters established by Hamelin et al:

- **Hypervisor Type** (type 1 or 2).
- **Supported HW architectures** (ARM, x86/64…).
- **Supported Guest OS** (Linux, RTEMS, FreeRTOS…).
- **Communication Services** (inter-partition communication).
- **API** (POSIX, OSEK, ARINC 653).
- **License**.

To these criteria we can add a few more, having reviewed the state of the art and seen where many of the hypervisors studied differ:

- **Virtualization Type** (full virtualization or paravirtualization).
- **Scheduling** (ARINC 653, no scheduling…).
- **Real-time partitions support** (and limitations, if any).
- **Developers' nationality/country of origin** (in addition to being of interest for analyzing the investment of each country in this type of technology, it could have an impact on its use in certain countries).
- **Ongoing maintenance** (evidence of maintenance in the last 3 years).
- **Multi-Guest OS** (supports more than two operating systems running simultaneously on different partitions).
- **Multicore** (runs on target hardware using more than one CPU core).

Note that this comparison will take into account the most prominent hypervisors in sections V-B, V-C, V-D and V-E. Proprietary hypervisors that, due to their high price, are oriented towards commercial exploitation and are not interesting for future research, have their own comparison table according to other criteria in section V-A.

VII. DISCUSSION

On the one hand, Tables 4 to 7 show that paravirtualization is the most popular method of virtualization (70% of the hypervisors analyzed offer paravirtualization), so it seems that efficiency is usually more highly valued than the flexibility of the virtualization solution. Along the same lines, type 1 hypervisors, which are more efficient because they have direct access to the hardware, are notably more common than type 2 hypervisors. Although it depends on the consideration given to Xen, KVM, Virtuosity or Jailhouse, which cannot be easily classified between type 1 and type 2, only 25% of hypervisors are undoubtedly type 2: Quest-V, RTA-LWHVR, NOVA, PharOS and HTTM.

On the other hand, we can see that Linux is clearly the most common guest OS supported by the hypervisors analyzed. Counting hypervisors offering full virtualization, up to 85% of hypervisors support an embedded Linux distribution as guest OS. This is not surprising in that Linux offers a robust open-source kernel, proven over many years, has a large community and provides access to a large number of

TABLE 4. Hypervisors comparison (part I).

	Xen	**XtratuM***	**KVM**	**Virtuosity**	**Quest-V**
Hyp. Type	1/2[a]	**1**	1/2[a]	1/2[a]	2
Virtualization Type	Paravirtualization[b] Full Virtualization	Paravirtualization	Paravirtualization Full Virtualization	Paravirtualization	Full Virtualization
Supported HW Architectures	x86 **ARM**	**ARM** SPARC **RISC-V**	x86 **ARM** PowerPC	x86 **ARM**	x86
Supported Guest OS	Any unmodified OS that runs on the supported HW architectures	LithOS Linux **RTEMS**	Any unmodified OS that runs on the supported HW architectures	Linux **FreeRTOS**	Any unmodified OS that runs on the supported HW architectures
Inter-Partition Communication	Yes	Yes	Yes	Yes	Yes
API	Own API	**APEX**	Own API	**APEX**	**POSIX**
License	GNU GPL v2	GNU GPL v2 Professional version also available	GNU GPL	GNU GPL v2	GNU GPL v3
Scheduling	Borrowed Virtual Time Simple Earliest Deadline First Credit ARINC 653	**ARINC 653**	Completely Fair	**ARINC 653**	No (static partitioning)
Multi-Guest OS	Yes	Yes	Yes	Yes	Yes
Real-Time Support	No[c]	**Yes**	No[c]	No[c]	**Yes**
Developers Nationality	Worldwide	Spanish	Worldwide	American	American
Ongoing Maintenance	Yes	Yes	Yes	Yes	No
Multicore	Yes	Yes	Yes	Yes	Yes

[a]Hypervisors like Xen, KVM, Virtuosity (Xen-based) or Jailhouse cannot easily be categorised as type 1 or type 2. On the one hand, they extend the Linux kernel to make it a type 1 hypervisor, but the host OS remains fully functional and all other guest OSs run as Linux processes on this host, so in this sense the hypervisor should be considered as type 2.

[b]Xen supports full virtualization and paravirtualization for x86 architectures. For ARM architectures, it only supports paravirtualization.

[c]Hypervisors such as Xen, KVM or Virtuosity (Xen-based) support real-time guest OSs. However, it should be noted that all these guests (domUs) will always depend on the correct functioning of the host (dom0), which is based on the Linux kernel and is not formally deterministic.

TABLE 5. Hypervisors comparison (part II).

	Rodosvisor	**RTZVisor**	**Xvisor**	**VOSYSmonitor**	**RTA-LWHVR**
Hyp. Type	**1**	**1**	**1**	**1**	2
Virtualization Type	Paravirtualization Full Virtualization	Paravirtualization	Paravirtualization Full Virtualization	Full Virtualization	Unknown
Supported HW Architectures	PowerPC	**ARM**	x86 **ARM** **RISC-V**	**ARM**	PowerPC
Supported Guest OS	Any unmodified OS that runs on the supported HW architectures	**FreeRTOS**	Any unmodified OS that runs on the supported HW architectures	Any unmodified OS that runs on the supported HW architectures	RTA-OS Unknown
Inter-Partition Communication	Yes	Yes	No	No	Yes
API	Own API **(similar to APEX)**	Own API	Own API	Own API	Own API
License	Unknown	Unknown	GNU GPL v2	Proprietary	Proprietary
Scheduling	**ARINC 653**	Round-Robin	Priority Round-Robin Priority Earliest Deadline First	Preemptive Priority	Reservation Based
Multi-Guest OS	Yes	Yes	Yes	No[a]	Yes
Real-Time Support	**Yes**	**Yes**	**Yes**	Up to 1 partition	No
Developers Nationality	Portuguese	Portuguese	Italian Indian	French	German
Ongoing Maintenance	No	No	Yes	Yes	Yes
Multicore	No	No	Yes	Yes	Yes

[a]VOSYSmonitor limits the number of real-time partitions that the system can host to one. By itself, it also guarantees to be able to run a single general-purpose partition, but allows the use of another hypervisor (such as Xen, for example) to expand the number of general-purpose partitions on the system.

software tools and development environments. As for real-time operating systems, we can find that the hypervisors analyzed support some very popular ones, such as FreeRTOS, RTEMS, Zephyr or VxWorks. ARM is the hardware architecture on which most hypervisors can be deployed (70% of the hypervisors analyzed have support for some ARM

TABLE 6. Hypervisors comparison (part III).

	Jailhouse	**Bao**	**OKL4**	**NOVA**	**ACRN**
Hyp. Type	1[a]	**1**	**1**	2	1[a]
Virtualization Type	Paravirtualization	Paravirtualization	Paravirtualization	Full Virtualization	Paravirtualization
Supported HW Architectures	x86 **ARM**	**ARM** **RISC-V**	**ARM** x86 MIPS	x86 **ARM**	x86
Supported Guest OS	Linux **FreeRTOS** Zephyr	Linux **FreeRTOS** **Erika RTOS**	Linux Windows **VxWorks**	Any unmodified OS that runs on the supported HW architectures	Linux Windows **VxWorks** Zephyr
Inter-Partition Communication	Yes	Yes	Yes	Yes	Yes
API	Own API	Own API	**POSIX**	Own API	Own API
License	GNU GPL v2	GNU GPL v2	Proprietary	GNU GPL v2	BSD 3-Clause
Scheduling	No (static partitioning)	No (static partitioning)	Round-Robin Priority	Preemptive Priority	Round-Robin Priority Borrowed Virtual Time
Multi-Guest OS	Yes	Yes	Yes	Yes	Yes
Real-Time Support	**Yes**	**Yes**	Up to 1 partition	**Yes**	Up to 2 partitions
Developers Nationality	German	Portuguese	American	German	Chinese
Ongoing Maintenance	Yes	Yes	Yes	Yes	Yes
Multicore	Yes	Yes	Yes	Yes	Yes

[a]Although Jailhouse is a type 1 hypervisor, it requires a Linux partition that loads the hypervisor firmware. After handing over control to the hypervisor, the Linux partition must continue to run. Something similar happens with the ACRN hypervisor, which requires a pre-launched partition that has exclusive access to certain hardware elements.

TABLE 7. Hypervisors comparison (part IV).

	Corbos	**Minos**	**PharOS**[a]	**seL4**	**HTTM**
Hyp. Type	**1**	**1**	2	**1**	2
Virtualization Type	Full Virtualization	Paravirtualization	Paravirtualization	Paravirtualization	Full Virtualization
Supported HW Architectures	Unknown	**ARM**	**ARM** **RISC-V**	**ARM** x86 **RISC-V**	MIPS
Supported Guest OS	Any unmodified OS that runs on the supported HW architectures	Linux Android Zephyr	PharOS (native) Trampoline	Linux	Any unmodified OS that runs on the supported HW architectures
Inter-Partition Communication	Yes	Yes	Yes	Yes	No
API	**POSIX**	Own API	**OSEK**	Own API	Own API
License	Proprietary	GNU GPL v2	Apache v2.0	GNU GPL v2	Unknown
Scheduling	Unknown	Unknown	**ARINC 653** Preemptive Priority	Preemptive Priority	Completely Fair
Multi-Guest OS	Yes	Yes	Yes	Yes	No
Real-Time Support	**Yes**	**Yes**	**Yes**	**Yes**	No
Developers Nationality	German	Chinese	French	Worldwide	Pakistani
Ongoing Maintenance	Yes	Yes	Yes	Yes	Yes
Multicore	Yes	Yes	Yes	Yes	Yes

[a]PharOS is originally an open source RTOS. However, Lemerre et al. added a virtualization layer to the RTOS in order to deploy VMs using another RTOS (Trampoline) [112], which technically makes PharOS a type 2 hypervisor.

architecture), followed by x86 architectures (50%) and RISC-V architectures (25%) and PowerPC (clearly below with 15%). The first hypervisors to adapt to the RISC-V architecture, a modern alternative to the more classical processors that is gaining momentum in the space, aviation, and automotive sectors, are XtratuM, XVisor, Bao, PharOS and seL4.

Although in the case of open source hypervisors it is common for there to be contributions from different developers from all over the world (this is especially noticeable in the larger hypervisors in terms of development time and lines of code, such as Xen and KVM), it can be seen that there are certain countries that stand out above the rest in terms of the development of virtualization technologies for embedded systems: the United States, Germany and France alone account for 50% of the hypervisors analyzed. This trend is even more pronounced in the case of proprietary hypervisors with more expensive licenses: in Table 3, all the hypervisors analyzed are American or German. In any case, these figures are not so surprising considering that these countries have a very strong embedded systems industry, especially safety-critical embedded systems, such as aviation, space, defense, or automotive. The number of Portuguese initiatives is striking, although only one of them is still actively supported (Bao). Besides them, only China is the country of origin of multiple (two) hypervisor development initiatives: ACRN and Minos.

Even with all these data, it is difficult to advise or advise against the use of one of these hypervisors over the others, because it will depend to a large extent on the particular needs of each investigation. As a general rule, hypervisors for which there is no ongoing maintenance (Quest-V, Rodosvisor and RTZVisor) should be avoided. If a low-overhead solution is required, it is better to opt for type 1 hypervisors offering paravirtualization, such as XtratuM, XVisor, OKL4, Minos or seL4. If the HW platform has sufficient resources, especially in terms of processing cores, it might be interesting to opt for simpler solutions offering static partitioning to obtain even lower overhead, such as Bao or Jailhouse. Even so, in the case of Jailhouse, as in the case of Xen, KVM or Virtuosity, it must be taken into account that their use is associated with the mandatory deployment of at least one Linux partition, so they might not be interesting in case the system to be developed has real time requirements. These hypervisors, as well as type 2 hypervisors (RTA-LWHVR, NOVA, PharOS or HTTM) and type 1 hypervisors offering full virtualization (Corbos and VOSYSmonitor) can be interesting in case system resources are not a very limiting factor, and a flexible solution is sought.

Finally, as regards their application in safety-critical systems, XtratuM, which has been applied in several aerospace projects and research works and is in the process of certification in some of them, as well as static partitioning solutions (Bao and Jailhouse) for simpler systems, would be particularly recommendable. In all these cases there is support for real-time operating systems, although XtratuM would be the most recommendable due to its API, which is much more similar to the APEX required by the ARINC 653 standard.

VIII. CONCLUSION

This article provided a comprehensive review of hypervisors, both proprietary and open-source, used as the basis for building a virtualized safety-critical embedded system. Once the hypervisors developed or adapted for this type of system have been identified, an exhaustive qualitative comparison was made made among them.

From our study we can draw several reflections, such as that paravirtualization is the most popular method of virtualization or that type 1 hypervisors are notably more common than type 2 hypervisors. Also, we have seen that Linux is clearly the most common guest OS supported by the hypervisors analyzed, that ARM is the hardware architecture on which most hypervisors can be deployed (although RISC-V is gaining momentum) and that, if the HW platform has sufficient resources, especially in terms of processing cores, it might be interesting to opt for simpler solutions offering static partitioning.

The aim is that this survey can serve as a starting point for future researchers in this area, who will be able to quickly check which hypervisor is best suited to their research needs. Experimental work is going on to study the performance features of several open-source hypervisors while executing safety-critical aerospace applications in an FPGA platform using ARM architecture.

REFERENCES

[1] C. Boettcher, R. DeLong, J. Rushby, and W. Sifre, "The MILS component integration approach to secure information sharing," in *Proc. IEEE/AIAA 27th Digit. Avionics Syst. Conf.*, Oct. 2008, pp. 1.C.2-1–1.C.2-14.

[2] B. Sutterfield, J. A. Hoschette, and P. Anton, "Future integrated modular avionics for jet fighter mission computers," in *Proc. IEEE/AIAA 27th Digit. Avionics Syst. Conf.*, Oct. 2008, pp. 1.A.4-1–1.A.4-11.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, and T. Mitra, "The worst-case execution-time problemoverview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[4] L. M. Kinnan, "Use of multicore processors in avionics systems and its potential impact on implementation and certification," in *Proc. IEEE/AIAA 28th Digit. Avionics Syst. Conf.*, Oct. 2009, pp. 1.E.4-1–1.E.4-6.

[5] R. Wilhelm, C. Ferdinand, C. Cullmann, D. Grund, J. Reineke, and B. Triquet, "Designing predictable multicore architectures for avionics and automotive systems," in *Proc. Workshop Reconciling Perform. With Predictability (RePP)*, Oct. 2009, pp. 2–3.

[6] B. Annighoefer, M. Halle, A. Schweiger, M. Reich, C. Watkins, S. H. VanderLeest, S. Harwarth, and P. Deiber, "Challenges and ways forward for avionics platforms and their development in 2019," in *Proc. IEEE/AIAA 38th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2019, pp. 1–10.

[7] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," in *Proc. Embedded Real Time Softw. Syst.*, May 2010, pp. 36–42.

[8] D. Kliem and S.-O. Voigt, "A multi-core FPGA-based SoC architecture with domain segregation," in *Proc. Int. Conf. Reconfigurable Comput. (FPGAs)*, Dec. 2012, pp. 1–7.

[9] P. Bieber, F. Boniol, M. Boyer, E. Noulard, and C. Pagetti, "New challenges for future avionic architectures," *Aeropsacelab J.*, vol. 4, p. 1, May 2012.

[10] Z. Gu and Q. Zhao, "A state-of-the-art survey on real-time issues in embedded systems virtualization," *J. Softw. Eng. Appl.*, vol. 5, no. 4, pp. 277–290, 2012.

[11] G. Taccari, L. Taccari, A. Fioravanti, L. Spalazzi, and A. Claudi, "Embedded real-time virtualization: State of the art and research challenges," in *Proc. 16th Real-Time Linux Workshop*, Oct. 2014, pp. 1–7.

[12] M. Cinque, D. Cotroneo, L. D. Simone, and S. Rosiello, "Virtualizing mixed-criticality systems: A survey on industrial trends and issues," *Future Gener. Comput. Syst.*, vol. 129, pp. 315–330, Apr. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004787

[13] J. A. N. Lee, "Claims to the term 'time-sharing,'" *IEEE Ann. Hist. Comput.*, vol. 14, no. 1, pp. 16–54, Jan. 1992.

[14] C. Strachey, "Time sharing in large fast computers," in *Communications of the ACM*, vol. 2. New York, NY, USA: Assoc Computing Machinery Broadway, 1959, pp. 12–13.

[15] F. J. Corbató, M. Merwin-Daggett, and R. C. Daley, "An experimental time-sharing system," in *Proc. May Spring Joint Comput. Conf. AIEE-IRE (Spring)*, 1962, pp. 335–344, doi: 10.1145/1460833.1460871.

[16] J. A. N. Lee, R. M. Fano, A. L. Scherr, F. J. Corbato, and V. A. Vyssotsky, "Project MAC (time-sharing computing project)," *IEEE Ann. Hist. Comput.*, vol. 14, no. 2, pp. 9–13, Jan. 1992.

[17] J. G. Kemeny and T. E. Kurtz, "Dartmouth time-sharing," *Science*, vol. 162, no. 3850, pp. 223–228, Oct. 1968.

[18] R. J. Creasy, "The origin of the VM/370 time-sharing system," *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 483–490, Sep. 1981.

[19] L. A. Belady, R. P. Parmelee, and C. A. Scalzi, "The IBM history of memory management technology," *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 491–492, Sep. 1981.

[20] J. Siebert, "Instructional use of a mainframe interactive image analysis system," *Photogramm. Eng. Remote Sens.*, vol. 49, no. 8, pp. 1159–1165, 1983.

[21] T. Shanley, *Protected Mode Software Architecture*. Oxfordshire, U.K.: Taylor & Francis, 1996.

[22] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[23] R. R. Schaller, "Moore's law: Past, present and future," *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jul. 1997.

[24] D. M. Lee, "Usage pattern and sources of assistance for personal computer users," *MiS Quart.*, vol. 10, pp. 313–325, Dec. 1986.

[25] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[26] I. Mavridis and H. Karatza, "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing," *Future Gener. Comput. Syst.*, vol. 94, pp. 674–696, May 2019.

[27] A. N. Al-Quzweeni, A. Q. Lawey, T. E. H. Elgorashi, and J. M. H. Elmirghani, "Optimized energy aware 5G network function virtualization," *IEEE Access*, vol. 7, pp. 44939–44958, 2019.

[28] C.-W. Lin, B. Kim, and S. Shiraishi, "Hardware virtualization and task allocation for plug-and-play automotive systems," *IEEE Design Test*, vol. 38, no. 5, pp. 65–73, Oct. 2021.

[29] A. Desai, R. Oza, P. Sharma, and B. Patel, "Hypervisor: A survey on concepts and taxonomy," *Int. J. Innov. Technol. Exploring Eng.*, vol. 2, no. 3, pp. 222–225, 2013.

[30] A. Carvalho, V. Silva, F. Afonso, P. Cardoso, J. Cabral, M. Ekpanyapong, S. Montenegro, and A. Tavares, "Full virtualization on low-end hardware: A case study," in *Proc. IECON 42nd Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2016, pp. 4784–4789.

[31] K. Gilles, S. Groesbrink, D. Baldin, and T. Kerstan, "Proteus hypervisor: Full virtualization and paravirtualization for multi-core embedded systems," in *Proc. Int. Embedded Syst. Symp.* Cham, Switzerland: Springer, 2013, pp. 293–305.

[32] B. Dordevic, V. Timcenko, S. Savic, and N. Davidovic, "Comparing hypervisor virtualization performance with the example of citrix hypervisor (XenServer) and Microsoft Hyper-V," in *Proc. 19th Int. Symp. INFOTEH-JAHORINA (INFOTEH)*, Mar. 2020, pp. 1–6.

[33] B. Djordjevic, V. Timcenko, N. Kraljevic, and N. Macek, "File system performance comparison in full hardware virtualization with ESXi, KVM, Hyper-V and xen hypervisors," *Adv. Electr. Comput. Eng.*, vol. 21, no. 1, pp. 11–20, 2021.

[34] R. C. Bhushan and D. K. Yadav, "Modelling and formally verifying Intel VT-x: Hardware assistance for processors running virtualization platforms," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 4, pp. 241–247, 2019.

[35] N. Amit, M. Ben-Yehuda, and B.-A. Yassour, "IOMMU: Strategies for mitigating the IOTLB bottleneck," in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 256-274.

[36] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "SR-IOV support for virtualization on InfiniBand clusters: Early experience," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013, pp. 385–392.

[37] A. Chierici and R. Veraldi, "A quantitative comparison between xen and kvm," *J. Phys., Conf. Ser.*, vol. 219, no. 4, Apr. 2010, Art. no. 042005.

[38] J. Nakajima, Q. Lin, S. Yang, M. Zhu, S. Gao, M. Xia, P. Yu, Y. Dong, Z. Qi, K. Chen, and H. Guan, "Optimizing virtual machines using hybrid virtualization," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2011, pp. 573–578.

[39] V. Bonifaci, B. Brandenburg, G. D'Angelo, and A. Marchetti-Spaccamela, "Multiprocessor real-time scheduling with hierarchical processor affinities," in *Proc. 28th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2016, pp. 237–247.

[40] M. A. Sánchez-Puebla and J. Carretero, "A new approach for distributed computing in avionics systems," in *Proc. 1st Int. Symp. Inf. Commun. Technol.*, 2003, pp. 579–584.

[41] P. J. Prisaznuk, "Integrated modular avionics," in *Proc. IEEE Nat. Aerosp. Electron. Conf., (NAECON)*, May 1992, pp. 39–45.

[42] T. Gaska, C. Watkin, and Y. Chen, "Integrated modular avionics-past, present, and future," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 30, no. 9, pp. 12–23, Sep. 2015.

[43] A. E. E. Committee, "ARINC 653: Avionics application software standard interface (draft 15)," Tech. Rep., 1996.

[44] S. H. VanderLeest, "Taming interrupts: Deterministic asynchronicity in an ARINC 653 environment," in *Proc. IEEE/AIAA 33rd Digit. Avionics Syst. Conf. (DASC)*, Oct. 2014, pp. 8A3-1–8A3-11.

[45] S. H. VanderLeest and D. White, "MPSoC hypervisor: The safe & secure future of avionics," in *Proc. IEEE/AIAA 34th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2015, pp. 6B5-1–6B5-14.

[46] J. Windsor and K. Hjortnaes, "Time and space partitioning in spacecraft avionics," in *Proc. 3rd IEEE Int. Conf. Space Mission Challenges Inf. Technol.*, Jul. 2009, pp. 13–20.

[47] J. Andersson, M. Hjorth, F. Johansson, and S. Habinc, "LEON processor devices for space missions: First 20 years of LEON in space," in *Proc. 6th Int. Conf. Space Mission Challenges Inf. Technol. (SMC-IT)*, New York, NY, USA, Sep. 2017, pp. 136–141.

[48] D. Paikowsky, "What is new space? The changing ecosystem of global space activity," *New Space*, vol. 5, no. 2, pp. 84–88, Jun. 2017.

[49] M., M. Corici, S. Covaci, and M. Guta, "5G and beyond for new space: Vision and research challenges," in *Proc. Adv. Commun. Satell. Syst. 37th Int. Commun. Satell. Syst. Conf. (ICSSC)*, Nov. 2019, pp. 1–16.

[50] A. Hughes and A. Awad, "Quantifying performance determinism in virtualized mixed-criticality systems," in *Proc. IEEE 22nd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2019, pp. 181–184.

[51] H. Guissouma, H. Klare, E. Sax, and E. Burger, "An empirical study on the current and future challenges of automotive software release and configuration management," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2018, pp. 298–305.

[52] U. Winkelhake and U. Winkelhake, "Vision digitised automotive industry 2030," in *The Digital Transformation of the Automotive Industry: Catalysts, Roadmap, Practice*, 2022, pp. 85–145.

[53] M. Z. Manic, M. Z. Ponos, M. Z. Bjelica, and D. Samardzija, "Proposal for graphics sharing in a mixed criticality automotive digital cockpit," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2020, pp. 1–4.

[54] S. Karthik, K. Ramanan, N. Devshatwar, S. Paul, V. Mahaveer, S. Zhao, M. Vishwanathan, and C. Matad, "Hypervisor based approach for integrated cockpit solutions," in *Proc. IEEE 8th Int. Conf. Consum. Electron. Berlin (ICCE-Berlin)*, Sep. 2018, pp. 1–6.

[55] M. R. Kabir, N. Mishra, and S. Ray, "VIVE: Virtualization of vehicular electronics for system-level exploration," in *Proc. IEEE Int. Intell. Transp. Syst. Conf. (ITSC)*, Sep. 2021, pp. 3307–3312.

[56] T. Gaska, Y. Chen, and D. Summerville, "Leveraging driverless car investment in next generation integrated modular avionics (IMA)," in *Proc. IEEE/AIAA 35th Digital Avionics Syst. Conf. (DASC)*, Sep. 2016, pp. 1–9.

[57] T. Cruz, P. Simoes, and E. Monteiro, "Virtualizing programmable logic controllers: Toward a convergent approach," *IEEE Embedded Syst. Lett.*, vol. 8, no. 4, pp. 69–72, Dec. 2016.

[58] J. Knorr, "Exploring Xen/kVM in prototyping an automotive use-case," Ph.D. dissertation, ISEP, Arlington, VA, USA, 2019.

[59] D. Sabogal and A. D. George, "Towards resilient spaceflight systems with virtualization," in *Proc. IEEE Aerosp. Conf.*, Mar. 2018, pp. 1–8.

[60] S. H. VanderLeest, "Designing a future airborne capability environment (FACE) hypervisor for safety and security," in *Proc. IEEE/AIAA 36th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2017, pp. 1–9.

[61] A. I. Kistijantoro and A. Gilbran, "Improving ARINC 653 system reliability by using fault-tolerant partition scheduling," in *Proc. 5th Int. Conf. Adv. Inform., Concept Theory Appl. (ICAICTA)*, Aug. 2018, pp. 182–187.

[62] T. Bijlsma, A. Buriachevskyi, A. Frigerio, Y. Fu, K. Goossens, A. O. Ors, P. J. van der Perk, A. Terechko, and B. Vermeulen, "A distributed safety mechanism using middleware and hypervisors for autonomous vehicles," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1175–1180.

[63] A. Avanzini, P. Valente, D. Faggioli, and P. Gai, "Integrating Linux and the real-time ERIKA OS through the xen hypervisor," in *Proc. 10th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2015, pp. 1–7.

[64] B. Schulz and B. Annighofer, "Evaluation of adaptive partitioning and real-time capability for virtualization with xen hypervisor," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 58, no. 1, pp. 206–217, Feb. 2022.

[65] C. Dall and J. Nieh, "KVM/ARM: The design and implementation of the Linux ARM hypervisor," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 333–348, Feb. 2014, doi: 10.1145/2644865.2541946.

[66] J.-S. Ma, H.-Y. Kim, and W. Choi, "KVM-QEMU virtualization with ARM64bit server system," in *Cloud Computing*, Y. Zhang, L. Peng, and C.-H. Youn, Eds. New York, NY, USA: Springer, 2016, pp. 334–343.

[67] D. Hildenbrand and M. Schulz, "Virtio-mem: Paravirtualized memory hot(un)plug," in *Proc. 17th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, Apr. 2021, pp. 1–14.

[68] V. K. Manik and D. Arora, "Performance comparison of commercial VMM: ESXI, XEN, HYPER-V & KVM," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*. New York, NY, USA, Mar. 2016, pp. 1771–1775.

[69] G. P. C. Tran, Y.-A. Chen, D.-I. Kang, J. P. Walters, and S. P. Crago, "Hypervisor performance analysis for real-time workloads," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2016, pp. 1–7.

[70] H. Shi, X. Li, and Y. Zhao, "Database performance comparison of xen, KVM and OSv using Cassandra," in *Proc. 2nd IEEE Adv. Inf. Manag., Communicates, Electron. Autom. Control Conf. (IMCEC)*, May 2018, pp. 27–30.

[71] M. Raho, A. Spyridakis, M. Paolino, and D. Raho, "KVM, Xen and docker: A performance analysis for ARM based NFV and cloud computing," in *Proc. IEEE 3rd Workshop Adv. Inf., Electron. Electr. Eng. (AIEEE)*, Nov. 2015, pp. 1–8.

[72] T. M$^{\acute{}}$uller, H. Askaripoor, and A. Knoll, "Performance analysis of KVM hypervisor using a self-driving developer kit," in *Proc. IECON 48th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2022, pp. 1–7.

[73] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, "XtratuM: A hypervisor for safety critical embedded systems," in *Proc. 11th Real-Time Linux Workshop*, 2009, pp. 263–272.

[74] D. Gastón Ochoa, "A study of Xtratum as a tool for space and time partitioning in safety-critical avionics software," Ph.D. dissertation, ETSI_Sistemas_Infor, Sophia Antipolis, France, 2020.

[75] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. L. Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, "De-RISC: The first RISC-V space-grade platform for safety-critical systems," in *Proc. IEEE Space Comput. Conf. (SCC)*, Aug. 2021, pp. 17–26.

[76] J. Galizzi, M. Pignol, M. Masmano, M. Munoz, J. Coronel, T. Parrain, and P. Combettes, "Temporal duplex-triplex on COTS processors with XtratuM," in *Proc. DASIA Data Syst. Aerosp.*, vol. 736, 2016, p. 20.

[77] J. Savithry, A. G. Ortega, A. S. Pillai, P. Balbastre, and A. Crespo, "Design of criticality-aware scheduling for advanced driver assistance systems," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 1407–1410.

[78] A. Larrucea, J. Perez, I. Agirre, V. Brocal, and R. Obermaisser, "A modular safety case for an IEC-61508 compliant generic hypervisor," in *Proc. Euromicro Conf. Digit. Syst. Design*, Aug. 2015, pp. 571–574.

[79] V. Muttillo, L. Tiberi, L. Pomante, and P. Serri, "Benchmarking analysis and characterization of hypervisors for space multicore systems," *J. Aerosp. Inf. Syst.*, vol. 16, no. 11, pp. 500–511, Nov. 2019.

[80] S.-W. Kim, J.-W. Choi, J.-Y. Jeong, and B.-S. Yoo, "Development of RTEMS SMP platform based on XtratuM virtualization environment for satellite flight software," *J. Korean Soc. Aeronaut. Space Sci.*, vol. 48, no. 6, pp. 467–478, Jun. 2020.

[81] S.-W. Kim, B.-S. Yoo, J.-Y. Jeong, and J.-W. Choi, "Overhead analysis of XtratuM for space in SMP envrionment," *IEMEK J. Embedded Syst. Appl.*, vol. 15, no. 4, pp. 177–187, 2020.

[82] S. Campagna, M. Hussain, and M. Violante, "Hypervisor-based virtual hardware for fault tolerance in COTS processors targeting space applications," in *Proc. IEEE 25th Int. Symp. Defect Fault Tolerance VLSI Syst.*, Oct. 2010, pp. 44–51.

[83] S. Grixti, N. Sammut, M. Hernek, E. Carrascosa, M. Masmano, and A. Crespo, "Separation kernel robustness testing: The XtratuM case study," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2016, pp. 524–531.

[84] L. Masing et al., "XANDAR: Exploiting the X-by-construction paradigm in model-based development of safety-critical systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1–5.

[85] T. Poggi, P. Onaindia, M. Azkarate-askatsua, K. Gruttner, M. Fakih, S. Peiro, and P. Balbastre, "A hypervisor architecture for low-power real-time embedded systems," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 252–259.

[86] A. Crespo, P. Balbastre, J. Simo, J. Coronel, D. Gracia-Perez, and P. Bonnot, "Hypervisor-based multicore feedback control of mixed-criticality systems," *IEEE Access*, vol. 6, pp. 50627–50640, 2018.

[87] E. Missimer, R. West, and Y. Li, "Distributed real-time fault tolerance on a virtualized multi-core system," in *Proc. OSPERT*, 2014, p. 17.

[88] A. Carvalho, F. Afons, P. Cardoso, J. Cabral, M. Ekpanyapong, S. Montenegro, and A. Tavares, "Functionality farming in POK/rodosvisor," in *Proc. Int. J. Comput. Sci. Softw. Eng.*, vol. 5, pp. 161–174, Aug. 2016.

[89] S. Pinto, A. Tavares, and S. Montenegro, "Space and time partitioning with hardware support for space applications," in *Proc. Data Syst. Aerosp. (DASIA), Eur. Space Agency, (Special Publication) ESA SP*, 2016, pp. 1–7.

[90] J. Martins, J. Alves, J. Cabral, A. Tavares, and S. Pinto, "μRTZVisor: A secure and safe real-time hypervisor," *Electronics*, vol. 6, no. 4, p. 93, Oct. 2017.

[91] S. Pinto, J. Martins, J. Lopes, M. Abreu, and A. Tavares, "SecSSy hypervisor: Security-safety synergy for aerospace," in *Proc. DAta Syst. Aerosp. (DASIA)*, Jun. 2017, pp. 1–8.

[92] J. Winter, "Trusted computing building blocks for embedded Linux-based ARM trustzone platforms," in *Proc. 3rd ACM Workshop Scalable Trusted Comput.*, Oct. 2008, p. 2130, doi: 10.1145/1456455.1456460.

[93] M. Cereia and I. C. Bertolotti, "Virtual machines for distributed real-time systems," *Comput. Standards Interface*, vol. 31, no. 1, pp. 30–39, Jan. 2009.

[94] S. W. Kim, C. Lee, M. Jeon, H. Y. Kwon, H. W. Lee, and C. Yoo, "Secure device access for automotive software," in *Proc. Int. Conf. Connected Vehicles Expo. (ICCVE)*, Dec. 2013, pp. 177–181.

[95] G. Cicero, A. Biondi, G. Buttazzo, and A. Patel, "Reconciling security with virtualization: A dual-hypervisor design for ARM TrustZone," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2018, pp. 1628–1633.

[96] P. Lucas, K. Chappuis, B. Boutin, J. Vetter, and D. Raho, "VOSYS-monitor, a TrustZone-based hypervisor for ISO 26262 mixed-critical system," in *Proc. 23rd Conf. Open Innov. Assoc. (FRUCT)*, Nov. 2018, pp. 231–238.

[97] D. Dasari, M. Pressler, A. Hamann, D. Ziegenbein, and P. Austin, "Applying reservation-based scheduling to a μC-based hypervisor: An industrial case study," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 987–990.

[98] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no VM exits! (almost)," 2017, *arXiv:1705.06932*.

[99] C. Hernandez, J. Flieh, R. Paredes, C.-A. Lefebvre, I. Allende, J. Abella, D. Trillin, M. Matschnig, B. Fischer, K. Schwarz, J. Kiszka, M. Ronnback, J. Klockars, N. McGuire, F. Rammerstorfer, C. Schwarzl, F. Wartet, D. Ludemann, and M. Labayen, "SELENE: Self-monitored dependable platform for high-performance safety-critical systems," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 370–377.

[100] A. Golchin, S. Sinha, and R. West, "Boomerang: Real-time I/O meets legacy systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 390–402.

[101] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Proc. Workshop Next Gener. Real-Time Embedded Syst. (NG-RES)*, vol. 77, M. Bertogna and F. Terraneo, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik, 2020, pp. 3:1–3:14. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11779

[102] G. Heiser and B. Leslie, "The OKL4 microvisor: Convergence point of microkernels and hypervisors," in *Proc. 1st ACM Asia–Pacific Workshop Workshop Syst.*, Aug. 2010, p. 1924, doi: 10.1145/1851276.1851282.

[103] R. J. Wolfe, "OKL4 hypervisor software development kit plugin," Tech. Rep., 2019.

[104] E. de Matos and M. Ahvenjärvi, "seL4 microkernel for virtualization use-cases: Potential directions towards a standard VMM," 2022, *arXiv:2210.04328*.
[105] U. Steinberg and B. Kauer, "NOVA: A microhypervisor-based secure virtualization architecture," in *Proc. 5th Eur. Conf. Comput. Syst.*, Apr. 2010, Art. no. 209222, doi: 10.1145/1755913.1755935.
[106] H. Li, X. Xu, J. Ren, and Y. Dong, "ACRN: A big little hypervisor for IoT development," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ.*, Apr. 2019, p. 3144, doi: 10.1145/3313808.3313816.
[107] K. Lampka and A. Lackorzynski, "Using hypervisor technology for safe and secure deployment of high-performance multicore platforms in future vehicles," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 783–786.
[108] H. Joe, D. Kang, J.-A. Shin, V. Dupre, S.-Y. Kim, T. Kim, and C. Lim, "Remote graphical processing for dual display of RTOS and GPOS on an embedded hypervisor," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–4.
[109] Y. Lim and H. Kim, "Cache-aware real-time virtualization for clustered multi-core platforms," *IEEE Access*, vol. 7, pp. 128628–128640, 2019.
[110] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive E/E-systems," in *Proc. 9th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2014, pp. 189–198.
[111] M. Lemerre, E. Ohayon, D. Chabrol, M. Jan, and M.-B. Jacques, "Method and tools for mixed-criticality real-time applications within PharOS," in *Proc. 14th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput. Workshops*, Mar. 2011, pp. 41–48.
[112] Q. U. Ain, U. Anwar, M. A. Mehmood, and A. Waheed, "HTTM–design and implementation of a type-2 hypervisor for MIPS64 based systems," *J. Phys., Conf. Ser.*, vol. 787, Jan. 2017, Art. no. 012006.
[113] Q. Ain and M. A. Mehmood, "Runtime performance evaluation and optimization of type-2 hypervisor for MIPS64 architecture," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 2, pp. 295–307, Feb. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157819308390
[114] M. Project. (2022). *Minos Flexible Virtualization Solution for Embedded System*. [Online]. Available: https://github.com/minosproject/minos
[115] B. Jansen, F. Korkmaz, H. Derya, M. Hubner, M. L. Ferreira, and J. C. Ferreira, "Towards a type 0 hypervisor for dynamic reconfigurable systems," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2017, pp. 1–7.
[116] Z. Jiang, N. C. Audsley, and P. Dong, "BlueVisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 75–84.
[117] M. Polenov, V. Guzik, and V. Lukyanov, "Hypervisors comparison and their performance testing," in *Proc. Comput. Sci. Line Conf.*, vol. 1. Cham, Switzerland: Springer, Jul. 2019, pp. 148–157.

SANTIAGO LOZANO is currently pursuing the Ph.D. degree in computer science and technology with the University Carlos III of Madrid. He is an Aerospace and Defense Engineer with SENER Aeroespacial, where he participated in numerous European projects, such as the Development of the Vega-C Rocket Navigation Unit or the Future Combat Air System (FCAS) Program.

TAMARA LUGO (Member, IEEE) received the master's degree in electronic systems and applications engineering. She is currently pursuing the Ph.D. degree in computer science and technology with the University Carlos III of Madrid, Spain. She is a Telecommunications and Electronics Engineer. Her research interest includes embedded real-time systems applied to space systems.

JESÚS CARRETERO (Senior Member, IEEE) has been a Full Professor of computer architecture and technology with the University Carlos III of Madrid, Spain, since 2002. He is currently involved in three other EU projects, coordinating the ADMIRE FET-HPC. His research interests include high-performance computing systems, large-scale distributed systems, and real-time systems. He is a Senior Member of the IEEE Computer Society. He was an Action Chair of the IC1305 COST Action "Network for Sustainable Ultrascale Computing Systems (NESUS)." He has been the General Chair of HPCC 2011, MUE 2012, ISPA 2016, and CCGRID 2017.