IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 26, 2021, accepted May 5, 2021, date of publication May 17, 2021, date of current version May 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3080842

SPHERE: A Multi-SoC Architecture for Next-Generation Cyber-Physical Systems Based on Heterogeneous Platforms

ALESSANDRO BIONDI[1], (Member, IEEE), DANIEL CASINI[1], (Member, IEEE), GIORGIOMARIA CICERO[1], NICCOLÒ BORGIOLI[1], GIORGIO BUTTAZZO[1], (Fellow, IEEE), GAETANO PATTI[2], (Member, IEEE), LUCA LEONARDI[2], (Member, IEEE), LUCIA LO BELLO[2], (Senior Member, IEEE), MARCO SOLIERI[3], (Member, IEEE), PAOLO BURGIO[3], (Member, IEEE), IGNACIO SANUDO OLMEDO[3], (Member, IEEE), ANGELO RUOCCO[3], LUCA PALAZZI[3], MARKO BERTOGNA[3], (Senior Member, IEEE), ALESSANDRO CILARDO[4], (Senior Member, IEEE), NICOLA MAZZOCCA[4], (Senior Member, IEEE), AND ANTONINO MAZZEO[4], (Senior Member, IEEE)

[1]Scuola Superiore Sant'Anna, 56127 Pisa, Italy
[2]University of Catania, 95124 Catania, Italy
[3]University of Modena and Reggio Emilia, 41121 Modena, Italy
[4]University of Naples Federico II, 80138 Naples, Italy

Corresponding author: Alessandro Biondi (alessandro.biondi@santannapisa.it)

This work was supported in part by the Italian Ministry of University and Research (MIUR) through the SPHERE project funded within the PRIN-2017 framework under Grant 93008800505.

ABSTRACT This paper presents SPHERE, a project aimed at the realization of an integrated framework to abstract the hardware complexity of interconnected, modern system-on-chips (SoC) and simplify the management of their heterogeneous computational resources. The SPHERE framework leverages hypervisor technology to virtualize computational resources and isolate the behavior of different subsystems running on the same platform, while providing safety, security, and real-time communication mechanisms. The main challenges addressed by SPHERE are discussed in the paper along with a set of new technologies developed in the context of the project. They include isolation mechanisms for mixed-criticality applications, predictable I/O virtualization, the management of time-sensitive networks with heterogeneous traffic flows, and the management of field-programmable gate arrays (FPGA) to provide efficient implementations for cryptography modules, as well as hardware acceleration for deep neural networks. The SPHERE architecture is validated through an autonomous driving use-case.

INDEX TERMS Cyber-physical systems, embedded systems, real-time systems, hypervisor, FPGA.

I. INTRODUCTION

Today's commercial-of-the-shelf (COTS) heterogeneous multicore platforms offer great opportunities for developing high-performance embedded computing systems. At the same time, the increased complexity of emerging applications (e.g., self driving cars, humanoid robots, augmented reality, and virtual interactive environments) demand for the integration of different subsystems and functionalities, imposing additional requirements to embedded systems designers that highlighted the limits of current development frameworks. Examples of such requirements include

- support for heterogeneous sensors (as multiple cameras, radars, and lidars) producing intensive data streams;
- real-time communication among distributed control units;
- hardware acceleration of complex artificial intelligence algorithms;
- tight interaction with the environment, requiring fast and predictable sensory-motor control loops;

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

- fail-safe and fault tolerant behaviors in safety-critical scenarios;
- security features to protect safety-critical components from cyber attacks.

In addition, the full exploitation of modern heterogeneous computing platforms is quite complex and poses a number of new technical problems that seriously limit their usage. Some of the most crucial problems are listed below:

- **Virtualization support:** When multiple applications with different degrees of criticality need to co-exist on a shared processing platform, virtualization techniques need to be offered by the system to manage the access to shared hardware resources. Examples of shared resources include the CPU cores, caches, DRAMs (Dynamic Random Access Memories), and hardware accelerators, which may potentially give rise to significant contention delays when uncontrolled accesses are permitted [1], [2].
- **Isolation and timing predictability:** A predictable virtualization needs to provide techniques to limit delays incurred accessing the underlying shared hardware platform. This include the support for algorithms for temporal isolation [3], cache isolation [4], [5], and memory bandwidth reservation for both CPU cores and hardware accelerators [6], [7]. Adopting these techniques is also beneficial for making the system more secure, protecting a critical application from potential cyber-attacks.
- **Multi-SoC communications:** When interconnected, heterogeneous platforms need to rely on efficient and predictable communication networks. This is particularly important for emerging autonomous-driving applications, which might be so computationally expensive to require multiple systems-on-chips (SoCs) for being deployed.
- **Real-time guarantees:** Given the many sources of non-determinism in such complex, virtualized, heterogeneous, and potentially distributed architectures, performing timing analysis and providing real-time guarantees is far more difficult with respect to the case of a single processor platform.

This work presents the SPHERE project, which aims at providing an integrated framework to abstract the hardware complexity of interconnected, cutting-edge, multi-core platforms and simplify the management of heterogeneous computational resources. It leverages hypervisor technology to virtualize computational resources and isolate the behavior of different subsystems running on the same platform, while providing safety, security, and real-time communication mechanisms.

The framework provides technologies to guarantee the safe and secure behavior of applications for cyber-physical systems (CPSes) running on modern heterogeneous applications while accounting for the requirements for such workloads.

Contribution. SPHERE addresses these challenges by providing a multi-SoC architecture for:

- enabling isolation in a mixed-criticality setup where non-critical applications are executed on a shared hardware platform with critical tasks without harming their functional and temporal correctness;
- supporting predictable I/O virtualization mechanisms, with guaranteed bounds on the maximum communication lateness;
- handling time-sensitive networks with multiple traffic flows requiring different temporal constraints; and
- managing portions of field-programmable gate arrays (FPGA) to provide efficient implementations for cryptography modules, as well as hardware acceleration for deep neural networks.

The SPHERE architecture is validated through an autonomous driving use-case.

Paper structure. The remainder of this paper is organized as follows. Section II presents the SPHERE architecture at a high level. Section III presents the proposed multi-domain execution environment, also introducing new technologies at the hypervisor level. Section IV presents the techniques that SPHERE uses to handle the FPGA. Section V addresses the virtualization of the communication system. Section VI describes a case study based on autonomous driving. Section VII concludes the paper.

II. OVERVIEW OF THE ARCHITECTURE

SPHERE is a multi-SoC architecture for next-generation CPSes based on FPGA-based heterogeneous computing platforms. The architecture is illustrated in Figure 1 for the case of two SoCs, which is the one considered by our case study. Each SoC provides a multi-domain execution environment implemented by hypervisor technology, which is detailed in Section III. The application running above the SPHERE architecture is distributed between the two SoCs. The two SoCs communicate by means of a Time-Sensitive Networking (TSN)-enabled IEEE 802.1Q switch deployed on the FPGA fabric of each SoC. Both the domains are allowed to use TSN communications thanks to an I/O virtualization mechanism offered by SPHERE and a TSN scheduling logic running at the application level. To support modern CPS applications that make use of machine learning algorithms, hardware accelerators for deep neural networks (DNNs) are also deployed on the FPGA fabric. Furthermore, FPGA-based accelerators for cryptographic services are available. The whole set of modules deployed on the FPGA fabric is managed by OS-level and hypervisor-level mechanisms provided by SPHERE.

III. MULTI-DOMAIN EXECUTION ENVIRONMENT

The SPHERE framework supports mixed-criticality applications by offering multiple execution domains. Our reference design consists of two domains: a non-critical one, based on the Linux operating system for running general-purpose software, and a critical one, based the Erika operating system for running real-time software. The two execution domains are provided by a hypervisor. SPHERE supports

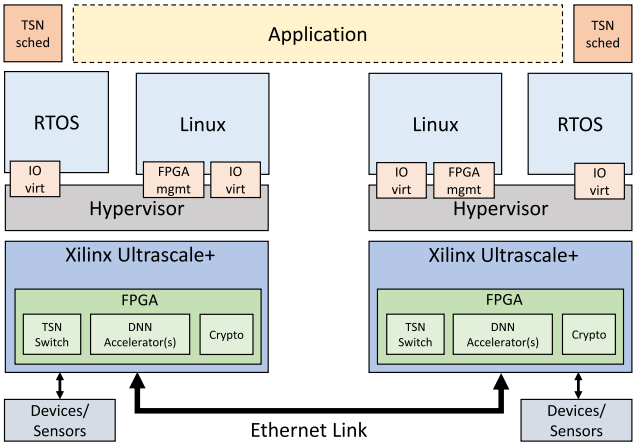


FIGURE 1. Illustration of the SPHERE architecture.

two hypervisors to ensure diversity in the implementation of the framework, namely CLARE-Hypervisor (Sec. III-A) and Jailhouse (Sec. III-B). Both the hypervisors provide isolation mechanisms for time-predictable software, which are reviewed in Section III-C. This section also presents two novel hypervisor-level features supported by SPHERE, namely time-predictable I/O virtualization (Sec. III-D) and direct interrupt dispatching (Sec. III-E).

A. CLARE-HYPERVISOR

CLARE-Hypervisor is a bare-metal type-1 hypervisor at the core of the CLARE software stack [8]. It has been developed by following an innovative design that integrates cutting-edge safety, security, and real-time resource management mechanisms by-design. CLARE-Hypervisor provides a fully-static approach with off-line configurations and optimizations to control the allocation of computing resources and the settings related to virtual CPUs, timing performance, and a set of mechanisms that allow regulating the interference among domains. Strong isolation among execution domains with mixed and independent levels of safety and security is ensured by the hypervisor while also enabling safe inter-domain communications, which can occur via either a port-based or a shared-memory-based low-latency mechanism. CLARE-Hypervisor provides a novel FastBoot technology and is characterized by a small code base (suitable to SIL4 certification) that does not depend on other software systems, which allows to drastically reduce the attack surface and time/space overhead. The hypervisor also offers modern security mechanisms such as address space layout randomization (ASLR), control flow integrity (CFI), TrustZone support, and safety mechanisms such as run-time health monitoring and fault recovery.

B. JAILHOUSE

Jailhouse [9], [10] is an open-source (GPL licensed) bare-metal hypervisor whose design is focused on safety-critical and real-time environments, built around two distinctive key traits. The first is static partitioning of hardware resources, an approach that eradicates the very possibility of contention between mixed-criticality domains caused by sharing. Devices are assigned to virtual machines, and not virtualized for them. The second trait is exploiting the Linux boot process to improve portability and minimize the hypervisor code base. In spite of being a type-1 hypervisor, it is not directly loaded from the boot-loader, but from a Linux kernel driver. Although Jailhouse currently features no virtualization support, it is the most advanced open-source hypervisor with respect to real-time applicability to high-performance embedded platforms, thanks to memory management and arbitration primitives [11].

C. ISOLATION MECHANISMS

In mixed-criticality systems it is of utmost importance to ensure a high degree of isolation among domains. In addition to the classical isolation mechanisms offered by almost all hypervisors, such as spatial isolation in memory, the hypervisors of the SPHERE framework offer advanced isolation features. For instance, in modern Arm SoCs, it is very common to have inclusive, data+instructions last-level caches shared by multiple cores that can represent a relevant source of mutual interference for domains running on different cores. CLARE-Hypervisor provides strong isolation mechanisms for regulating the access to the memory subsystem. In particular, it provides **(i)** *cache coloring* for reserving a dedicated portion of shared caches to each virtual machine (VM) or the hypervisor, **(ii)** *memory-bandwidth reservation* for controlling contention among VMs in accessing the main memory, and **(iii)** *bank-aware memory allocation* for limiting the sources of unpredictability introduced by memory controllers. CLARE-Hypervisor also offers isolation mechanisms for security purposes and for FPGA-based hardware accelerators. Jailhouse also supports these mechanisms in a fork maintained by partners of the SPHERE project.

Note that, since the hypervisor is directly and actively involved in dispatching interrupts, it represents itself a software component to be isolated to reduce potential interference. Shared levels of cache represent a clear source of contention for hypervisor instruction cacheability. Hence, cache-level isolation is also a desired feature in hypervisors used for time-critical systems.

D. TIME-PREDICTABLE I/O VIRTUALIZATION

SPHERE aims at providing a predictable I/O communication mechanism, ensuring that the maximum lateness is bounded. To achieve this goal in a virtualized environment, the I/O handling strategy needs to allow multiple VMs to share one or more I/O devices.

In SPHERE, this is done using a predictable, software-based I/O virtualization mechanism, whose structure is shown in Figure 2, which refers to a case where the network device is shared among different VMs. Applications running in the VMs perform I/O requests by interacting with a para-virtualized application programming interface (API) offered by the hypervisor, including function calls to send

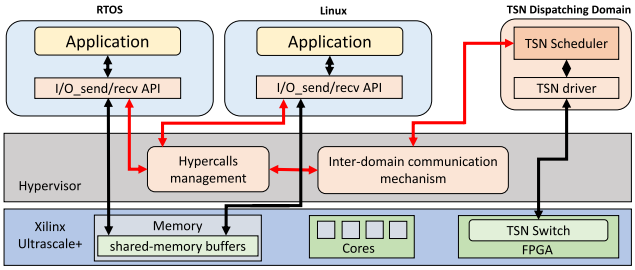


FIGURE 2. I/O Virtualization Architecture for time-sensitive networking.

and receive I/O messages. These functions use hypercalls to obtain buffers to communicate, maintained in a portion of shared memory that can be accessed by a target VM and by the I/O dispatching domain. The latter is a particular VM in charge of interacting with the actual I/O devices on behalf of the other domains. In this way, I/O peripherals are accessed by a single entity, which predictably arbitrates the accesses to shared devices.

The I/O dispatching domain collects I/O requests in different queues, one for each pair of device, VM, and communication type (input or output). It also implements a scheduling policy that selects the request to be performed by choosing among different requesters and different devices.

SPHERE provides a general scheduler for I/O requests that employs round-robin arbitration among requests in different queues (and hence either coming to different VMs or targeting different devices) and uses first-come-first-served (FCFS) arbitration to choose among those in the same queue. These two scheduling policies guarantee fairness in accessing I/O peripherals from different VMs, while also enabling the derivation of predictable timing bounds. Furthermore, the scheduler is flexible and modular enough to allow integrating special-purpose I/O schedulers, e.g., the TSN packet scheduler discussed later in Section V-B.

The shared memory buffers are used by a mechanism for inter-domain communication that moves I/O messages between the I/O dispatching domain and the other VMs. Shared buffers are implemented with wait-free queues [12], which provide predictable and constant timing effects while guaranteeing data integrity.

SPHERE provides the I/O virtualization mechanism with a timing analysis that allows bounding I/O communication latencies as well as predicting the worst-case response times of computational activities (i.e., tasks) interacting with I/O peripherals. The analysis is derived from fine-grained modeling of the system, which includes virtual machines, tasks, interrupt-service routines (ISRs), hypercalls, and I/O requests [13]. The individual contributions to the latencies are computed for each component, which potentially runs different scheduling policies, and are then combined to derive the end-to-end bounds to I/O delays. For example, the overall delay may be in part due to a VM hosting a real-time operating system (RTOS) that uses a fixed-priority scheduling policy, while its I/O requests are executed by the I/O dispatching domain through an FCFS+round-robin scheme. Additional effects due to I/O-driven interruptions and potential priority inversion phenomena due to hypercalls give rise to a complex scenario. The analysis proposed by SPHERE allows dealing with this tricky scenario while also being modular enough to be extended to other scheduling policies thanks to its composability.

E. DIRECT INTERRUPT DISPATCHING

Interrupt management plays a crucial role for achieving a predictable response to events, especially in the presence of a hypervisor. Most commercially-available Arm v8 systems are equipped with GICv2 (Generic Interrupt Controller), where the IRQs (Interrupt ReQuests) are controlled through two components: 1) the distributor (GICD: GIC distributor), responsible for enabling/disabling the IRQs and set their target and priority; 2) the CPU interface (GICC: GIC CPU interface), responsible for calling the interrupt handler and signaling the corresponding hardware device. To isolate critical applications from non-critical ones, it is not safe to go through the normal route, as a VM that has full access to the GICD and GICC could interfere with the IRQ operations of other VMs. The adopted solution is to leverage the hardware feature provided by the virtual distributor and the virtual CPU interface, so enabling the hypervisor to first handle IRQs for its guest VMs and then explicitly inject them to the appropriate VM.

In a normal setting, each IRQ requires two context switches (from VM to hypervisor and back), and the hypervisor needs to keep track of each configuration and request from every VM and for every IRQ. Measurements show that this route is at least 4x slower than the normal one (without hypervisor) and can get even slower depending on the complexity of the hypervisor IRQ handler. To reduce such a latency, we can leverage the static-partitioning feature of the hypervisor (where VMs do not share the same core) by extending the partitioning philosophy to the interrupts. In this case, the CPU Interface for the IRQs becomes private to the VM that gets that particular CPU assigned. Therefore, the hypervisor can skip the GICC emulation and its relative bookkeeping operations, and only emulate the distributor. In this way, the GICH (GIC Hypervisor registers) and GICV (GIC Virtual machine registers) are not used and can be turned off. Adopting this solution, for each IRQ received, the path is very similar to the one without hypervisor, and the interrupt latency is kept at nearly the same values. Another limitation is that direct dispatch will route all IRQs to VM space, skipping hypervisor space altogether. This is due to hardware design choices. A workaround for hypervisors to use signals for their internal functioning is to use either *fast interrupt requests* (FIQs, i.e., secure, fast interrupts, independently routable from IRQs) or SDEI (Software Delegated Exception Interface). Both solutions need some support at firmware level. An SDEI-based implementation is available in the official Jailhouse repository and an improved implementation has been prototyped for SPHERE. Evaluation experiments from

a bare-metal application on the Xilinx Zynq Ultrascale+ MPSoC (multi processor system-on-chip) have shown that interrupt virtualization with Jailhouse can cost more than 5x slowdown in the worst-case interrupt response (from around 260 to 1,410 ns). Using direct dispatch, instead, the latency is restored close to the case without Jailhouse (430 ns). Other experiments, carried to stress the memory hierarchy to produce inter-core interference, showed a dramatic degradation (from 7x to 15x) in the worst case with respect to the isolated case, in all aforementioned cases. Combining direct dispatch with cache-coloring protection, instead, allowed achieving an average-case delay of 240 us, and containing the worst-case delay within 1390 us, even with the hypervisor and under memory aggression.

IV. FPGA MANAGEMENT

Targeting the future evolution of modern MPSoC platforms, SPHERE also addresses *dynamic function exchange* capabilities (also known as dynamic partial reconfiguration) of modern heterogeneous devices, particularly those based on FPGA fabrics [14]. These capabilities are already consolidated in datacenters, where they are meant to extend multi-tenancy support from standard server machines to accelerators like GPU and FPGA cards. We however anticipate that the technical approach adopted for the server segment will eventually be adapted to the embedded area, allowing multiple hardware/software tasks with various levels of criticality in terms of both determinism and trustworthiness to co-exist and be dynamically loaded to the MPSoC platform. We therefore build on a common concept used in datacenter-oriented FPGAs, the device *shell*. This concept essentially refers to a statically configured portion of the FPGA, providing a fixed interface between the host's system bus (typically PCIe in server machines) and a large dynamically reconfigurable area of the FPGA device. This area is divided in one or more regions that can host multiple accelerators uploaded to the device through a dynamically reconfiguration process. The shell is in charge of mediating the accelerator reconfiguration triggered by the host and acts as a sandbox for the accelerators, which are not directly exposed to the physical system resources, including the FPGA-attached memory, e.g., a Double Data Rate (DDR) DRAM or High-Memory Bandwidth (HBM) banks.

The FPGA shell concept was considered by SPHERE for extension to the MPSoC domain, where the coupling between the host system, the FPGA, and the system memory is of course different from server-class machines, but the presence of multiple mixed-criticality hardware tasks (either dedicated accelerators or soft cores), being time-multiplexed in the FPGA fabric, creates similar needs in terms of reconfigurability, manageability, isolation, and security.

In the setting foreseen by SPHERE, depicted in Figure 3, the shell contains the internal reconfiguration port (the Internal Configuration Access Port, ICAP, in Xilinx FPGAs) and a custom reconfiguration controller. Besides interfacing the configuration port to the host, the controller supports

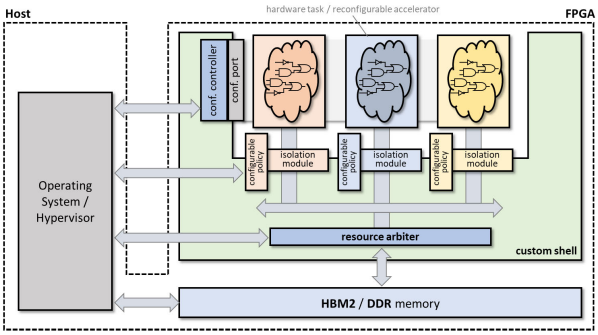


FIGURE 3. Shell architecture within the hardware-reconfigurable portion of an MPSoC platform.

a few additional functions. It can apply custom encryption/decryption/integrity checks to the partial bitstreams and features readback capabilities to extract the contents of all stateful components in the reconfigurable region, effectively taking a snapshot of the accelerator execution state at a particular point in time. This low-level mechanism underpins a variety of high-level operations that are crucial for management and reliability. In fact, SPHERE uses the readback capabilities to support checkpointing of accelerator execution state and hence preemption/context switch of hardware tasks. These two uses share fundamentally the same underlying technical approach. We first identify the notion of state in FPGA tasks, which boils down to the information bits physically held by flip-flops and on-/off-chip RAM. A basic approach consists in reading back the whole configuration bitstream from the dynamic region, without identifying the user data bits, which indeed form a significantly smaller subset than the whole bitstream. Moreover, even some of the user data bits may be unnecessary to preserve the application state for subsequent restoring. To reduce overheads, we define an application-level notion of execution state, i.e. the subset of physical state of relevance for saving/restoring the context. For that, we introduce a design flow and run-time support allowing the expression and extraction of application-level execution state. In this flow, the designer can optionally indicate which components are strictly needed for maintaining the execution state by explicitly instantiating special components from a dedicated support library (registers, configurable memories, etc.). Based on the naming convention used in the support library, low-level mapping information is extracted automatically after the place-and-route stage of the FPGA implementation flow (namely, by analyzing the `.ll` file in Xilinx Vivado toolflow). The logic to selectively extract, handle, and restore the state is derived from this mapping information. Following the design specification, the user is completely unaware of the process, enabling a transparent and automated flow. The availability of mapping information, which decouples the logic components in the design from their physical location after a specific implementation run, has also interesting implications in that it enables portability across different implementations of the same design, or possibly even different design versions having the same

application-level execution state, as long as the corresponding mapping information is made available.

Supported by the host side management routines, the reconfiguration logic can generate a partial configuration bitstream at runtime augmented with a previously saved task's state, so that writing a bitstream to the dynamic region effectively results in resuming the processing from the point in time when the task was preempted. As mentioned above, the low-level access mechanism to read/resume execution state is the FPGA internal reconfiguration port. We also considered the JTAG debug port as an access mechanism, which could limit or even avoid the resource overhead incurred by the FPGA shell, but it would perform poorly in terms of performance and provide no support for the additional security-related functions that are handled on-chip by the proposed shell-based solution. In the chosen setup, thus, the custom shell embraces the internal configuration port, which is a Xilinx ICAP primitive in our prototype, it implements the management logic in the static region, and exposes an AXI(Advanced eXtensible Interface)-lite control interface to the host-side software.

As an important remark, note that the above low-level mechanism can enable a full range of scenarios in addition to preemption. It can also be useful for supporting process migration across different implementations, as an underlying mechanism for reliability, or for checkpointing of long or critical FPGA tasks [15]. We thus plan to fully integrate the flow with the software layers in the MPSoC and make the low-level mechanisms accessible to, and controlled by the operating system/hypervisor. To this end, SPHERE will also aim at the integration with the FRED framework [16], [17] to enable time-predictable virtualization of the FPGA area by means of dynamic function exchange.

A. FPGA-BASED SECURITY SERVICES

The shell concept is not only concerned with the reconfiguration process, though. In fact, for its positioning in the system, the shell can also be in charge of arbitrating the access to shared resources from the different hardware tasks. Among other uses, this arbitration can offer interesting opportunities, as an isolation mechanism aimed at establishing performance guarantees exposed to the tasks. In addition, concerning security, the arbitration functions can be further extended to establish confidentiality, authentication, and integrity conditions. In fact, as shown in Figure 3, the shell contains one *isolation module* for each hardware task, which act as a gate between the task and the rest of the system. From the application's perspective, this module can support on-line encryption/decryption of bus transactions [18], relying on a high-throughput hardware stream cypher [19], which establish a form of confidentiality of information flows outside the perimeter of the hardware task and the (trusted) shell region. More importantly, from the system's perspective, the isolation modules can serve as firewalls monitoring the traffic patterns generated by the hardware tasks. This monitoring can be as simple as checking address bounds and access conditions, complementing existing MPSoC-level features like MMU/MPU and TrustZone, but can also apply smarter policies, for example detecting ill-behaved AXI transactions or even known patterns in the accelerator traffic that are flagged as unsafe. Such policies can be stored in a configuration memory that the shell, controlled by the host-side operating system, can dynamically change or update, allowing improved configurability and reactivity to possible threats.

V. VIRTUALIZED TSN COMMUNICATIONS

The SPHERE framework requires a communication network able to handle multiple traffic flows, internal and external to the heterogeneous platforms, with different constraints. In this context, the TSN family of standards offers a rich set of protocols and combinations to design network architectures that provide clock synchronization, fault-tolerance, and support for different traffic classes with different temporal requirements [20].

Many works addressed the key role of Ethernet and TSN for automotive communications [21]–[23], pointing to the novel features introduced in the IEEE 802.1Q standard [24], which enables Ethernet switches to support the delay and jitter requirements imposed on in-vehicle communications by modern car applications, such as automated driving.

In particular, in the SPHERE framework, three types of traffic flows are considered:

- **Real-time Control Traffic**. This is the highest priority traffic, consisting of small messages that require deterministic transmission times, zero jitter, and delivery guarantees. This traffic is typically transmitted according to a time-driven schedule, such as x-by-wire commands.
- **Periodic Real-time Traffic**. It includes periodic messages of different sizes, such as video streams, Lidar point clouds, and audio streams, which require bounded latencies and delivery guarantees.
- **Best-effort Traffic**. This is the lowest priority traffic, served with no guarantees on latencies and delivery, It includes messages for configurations, logs, GUI non-critical information, etc.

The TSN standards provide mechanisms that are suitable for managing the transmissions of flows belonging to any of these traffic types. In particular, the real-time control traffic can be transmitted using the so-called enhancements for scheduled traffic, defined in the IEEE 802.1Qbv-2015 amendment, now enrolled in the IEEE 802.1Q-2018 standard. This protocol provides novel queue management mechanisms applied to the egress queues of every switch port to enable/disable frame transmission from the queues. These mechanisms are based on *transmission gates*, which open or close according to the timings set in a gate control list that is cyclically scanned. Transmission gates can be used to guarantee temporal isolation for scheduled traffic, i.e., a high-priority traffic class that requires frame transmission based on a known timescale (i.e., time-driven transmissions). Time-driven operations leverage on the

common notion of time that is provided by the IEEE 802.1AS-2011 standard, and can also benefit from the recent IEEE 802.1AS-2020 standard that, among other things, provides reliable clock synchronization. However, when using the enhancements for scheduled traffic, transmission scheduling is a crucial aspect for the system performance and, depending on the number and types of flows to deal with and on their characteristics, may be a challenging task. In the literature, several approaches for frame scheduling have been proposed. For instance, the Satisfiability Modulo Theories (SMT) solver [25] is able to find a schedule solution by minimizing the delivery latencies or the number of needed egress queues, etc. However, the computation time needed to generate the schedule varies from some minutes to a few hours, depending on the network complexity (number of nodes/switches, number of flows, etc.). In [26], a novel heuristic scheduler is introduced for the so-called Stream Reservation (SR) classes introduced by the Ethernet Audio Video Bridging (AVB) standards and now rolled into the IEEE 802.1Q-2018 standard. It provides shorter schedule computation times, i.e., in the order of seconds, but it cannot guarantee zero jitter transmissions. However, such a scheduler guarantees a bounded interference between ST frames scheduled in the same queue at the same time.

Periodic real-time traffic can be mapped onto the SR traffic classes, handled according to the Stream Reservation Protocol (SRP) and frame transmissions undergo traffic shaping using the Credit-Based Shaper (CBS). In particular, the IEEE 802.1Q-2018 standard defines the parameters of two SR classes, i.e.:

- **SR Class A**: it is the highest priority for SR classes, providing a maximum delivery latency of 2 ms over seven hops.
- **SR Class B**: it provides a maximum delivery latency of 50 ms over seven hops.

SR classes can also be used for transmitting sporadic traffic. However, to handle the transmission of multiple SR flows according to the SRP on a single platform, like in the case of the SPHERE framework, transmissions have to follow specific rules that need to be handled by the platform. For instance, SR frames belonging to the same flow have to be transmitted to the Ethernet port spaced at least of an interval defined for each class, that is 125 μs for SR class A and 250 μs for SR class B. ST frames, instead, have to be transmitted to the Ethernet port at specific time instants (according to the schedule).

Finally, best-effort flows are transmitted according to a static priority scheduling without traffic shaping and timing guarantees on message delivery.

In a real-time context, like the one addressed by the SPHERE framework, response-time analysis techniques are required to assess whether the time constraints associated with the different types of traffic flows can be guaranteed. Recent works addressed the response-time analysis of time-sensitive networks, and some of them also encompass the enhancements for scheduled traffic [27]–[29]. Scheduled Traffic (ST) is a high priority traffic class in which frames are transmitted according to a precise time schedule. The ST frame transmissions are handled by the gate mechanism to avoid interference from other classes.

In the SPHERE project, communications are handled not only at the network level, where transmissions are scheduled according to the protocols mentioned in this section, but also at the framework level. In fact, heterogeneous platforms with virtualization may have multiple virtual machines and a lower number of Ethernet ports and/or TSN interfaces. For this reason, in the following a suitable solution that enables TSN transmissions of multiple virtual machines running on the same platform will be discussed.

A. FPGA-BASED TSN SWITCH

To implement TSN protocols on a programmable SoC, we selected the Multiport-TSN Switch (MTSN) IP core by SoC-e [30], an all-in-one solution that can be adapted and optimized depending on the application, from a simple two-port end-point to a complex multiport switch. The IP uses the Gigabit Ethernet MAC (GEM) of the SoC to configure up to 16 internal/external MII (media-independent interface) / GMII (gigabit MII) / (reduced GMII) RGMII ports. MTSN targeted devices are the Xilinx Zynq-7000 SoC and the Zynq Ultrascale+ MPSoC.

The main quality of the IP is the high configuration capability, not only of the number of ports, but also of the settings like IP, MAC address MAC table, MDIO (Management Data Input Output), frame storage queues, and more. In addition to this, it offers the widest and most up-to-date support to the TSN standards used in the automotive and industrial domains [20], [31], including the following IEEE standards:

- **IEEE 802.1AS(rev)** for time synchronization.
- **IEEE 802.1Qav**, for the credit-based shaper applied to the Stream Reservation traffic classes.
- **IEEE 802.1Qbv** for scheduled traffic.
- **IEEE 802.1Qci** for per-stream filtering and policing.
- **IEEE 802.1CB** for frame replication and elimination for reliability.
- **IEEE 802.1Qbu and 802.3br** for frame preemption.

Starting from the only available vendor reference design, which targets a custom board with the Xilinx Zynq Ultrascale+ ZU3EG MPSoC, SPHERE addresses a prototypical implementation for the Xilinx ZCU102 evaluation board [32]. Since this board has a single Ethernet port, an FMC (FPGA Mezzanine Card) Ethernet module provides four additional ports.

Note that, if using an Ethernet FMC on a different board, we have to change how to handle the timing constraint and use the recommended PIN for the Ethernet FMC.

B. TSN TRANSMISSION SCHEDULING IN VIRTUALIZED HETEROGENEOUS PLATFORMS

The underlying computing platform typically offers a limited number of TSN-enabled Ethernet ports, which can be lower than the number of VMs. This entails contention on

the Ethernet port or the need for exclusive access from one virtual machine. Moreover, as TSN transmissions have to follow specific scheduling rules, (e.g., two consecutive frame transmissions of the same SR flow have to be spaced at least by a minimum time interval) a scheduling algorithm has to be also applied to schedule the transmissions among multiple VMs.

The SPHERE project envisages a specific software component (called *TSN Manager*) that runs the transmission scheduling algorithm, whose aim is transmitting the frames belonging to different flows and traffic classes while maintaining the properties and the behavior defined for each traffic class in the IEEE 802.1Q standard.

In particular, the goal of the TSN Manager is to transmit the frames belonging to specific flows to the Ethernet port/s at specific times. In particular, ST flows are jitter-sensitive, i.e., ST frames have to be transmitted to the Ethernet port at precise time instants and the transmissions to the Ethernet port of SR frames has to follow specific rules. Consequently, the TSN Manager, which is a task itself, has to be scheduled with real-time guarantees. A recent work by Leonardi *et al.* [33] investigated three different approaches to handle TSN real-time transmissions in a heterogeneous platform with virtualization, also presenting a preliminary assessment of their performance. As shown in Figure 4 *(a)*, *(b)* and *(c)*, these approaches run at the application level, at the virtual machine level and at the hypervisor level, respectively.

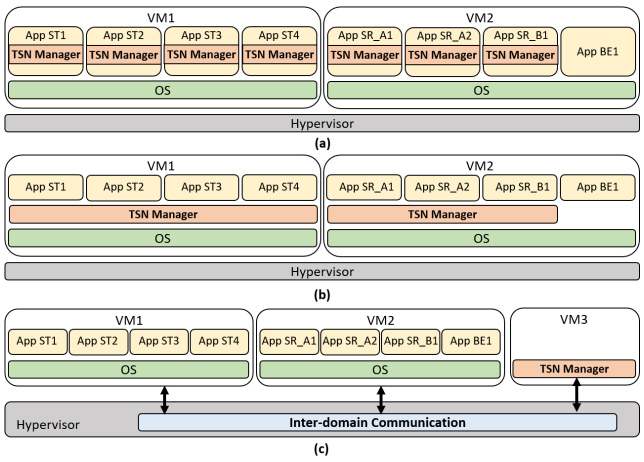


FIGURE 4. TSN Manager at (a) application level; (b) VM level (c); hypervisor level.

The results presented in [33] showed that the application-level transmission architecture offers a flexible solution, thanks to the per-flow scheduling granularity. However, suitable scheduling policies have to be used to limit jitter. The VM-level transmission architecture provides the application with transparent transmission scheduling. However, using this approach, the TSN Manager task is scheduled regardless of the priority of the flows to be transmitted, thus losing the per-flow scheduling granularity. Finally, the hypervisor-level transmission architecture also offers scheduling with per-flow granularity, like the application level, but provides better performance. However, this is at the expense of an increased hypervisor complexity, which is not desirable for safety and security purposes. Based on the findings of [33], the SPHERE framework relies on a VM-level architecture, but reserving a dedicated VM for the TSN Manager, as illustrated in Figure 5.

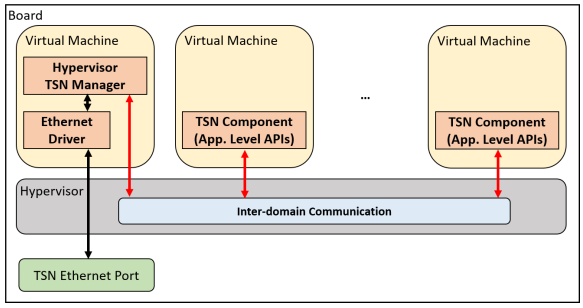


FIGURE 5. Archtecture of the SPHERE TSN Manager at the hypervisor level.

Each virtual machine communicates with the TSN Manager through a suitable API that allows the applications to transmit/receive the frames belonging to a stream, register/deregister a stream, and configure the network parameters. The API provided to the virtual machine can be used by the application to access the TSN manager through a suitable inter-domain communication mechanism.

As discussed in Section III-D, the predictable I/O mechanisms provided by SPHERE allows for a seamless integration of a special-purpose I/O scheduler. As shown in Figure 2, the TSN packet scheduler discussed in this section can be used by the TSN dispatching domain to handle I/O requests targeting the TSN switch, either in isolation (if the TSN switch is the only device handled by the platform) or in conjunction with other scheduling policy (in a hierarchical setting). For example, a platform supporting TSN as well as other I/O devices may prioritize the access to Ethernet (TSN) giving preference to Ethernet frames transmission requests.

VI. CASE STUDY: AUTONOMOUS DRIVING APPLICATION

The use case considered by the SPHERE project is a self-driving car able to follow a set of waypoints. The developed solution provides an end-to-end software stack capable of properly analyzing the surrounding environment, detecting ego vehicle position on a pre-loaded map, and providing the proper drive-by-wire control signals (throttle, steering, and brake). The car is able to navigate through a predetermined map of waypoints, promptly reacting to unexpected events in a highly dynamical environment, e.g., for avoiding obstacles. This use case integrates multiple software components, ranging from perception to control, that are becoming very relevant in the automotive/autonomous driving domain, and hence it is often adopted as possible baseline/benchmark to researchers. The tasks involved in the case study and their relations are illustrated in Figure 7, which is discussed with

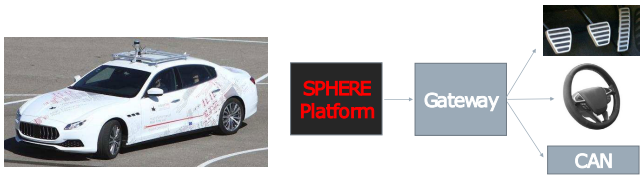


FIGURE 6. Vehicle and interface between SPHERE subsystem and vehicle DBW actuation gateway.

more details in Section VI-C. The figure specifies the name and the period (in milliseconds) of each software component.

As per project goals, safety and non-safety critical tasks are managed to execute and communicate in isolation using the SPHERE framework. Further details about the use case are described in the following subsections.

A. SENSOR DESCRIPTION

The vehicle mounts different sensors properly installed and integrated to ensure a complete coverage of its near surroundings. Specifically, the following sensors are installed:

- 5x Sekonix cameras [34], connected via Gigabit Multimedia Serial Link (GMSL);
- Ouster OS0 64-120m lidar [35], connected via Ethernet;
- GPS+IMU module, connected via Ethernet;
- 2x Continental ARS420 radars [36], connected via Ethernet.

1) VEHICLE INTERFACE

The target vehicle is a high-end passenger car used as a research platform for projects related to *autonomous driving* and *advanced driving assistance systems* (AD/ADASes). It is capable of drive-by-wire control and actuation by means of electrical signals on a Controller Area Network (CAN) bus driven by one (or more) centralized domain controllers and managed by vehicle electronic control units (ECUs). They are:

- steering control;
- shift control (from automatic to rear and vice versa);
- accelerator/brake control;
- power emergency ON/OFF button;
- steering encoder;
- encoder speed and odometry.

B. APPLICATIVE SOFTWARE ARCHITECTURE

The applicative software provides the proper throttle, steering, and brake signals to drive a vehicle through a predetermined map of waypoints that are defined in global coordinates while avoiding obstacles, like pedestrians or other vehicles. In this sense, the vehicle runs multiple tasks at different priorities, namely, to control vehicle mission (i.e., waypoint navigation), but also perform emergency maneuvers, if needed.

Our system undergoes a standard design for autonomous driving applications, which typically include the following components:

- **Sensing**, to gather information from the environment using different sensors.
- **Perception**, to detect and track the position of road-users (like pedestrians or vehicles).
- **Localization**, to determine the position of the car on the given environmental map.
- **Global** (aka mission) **planner**, to update the position and take decisions in the whole path that the car must follow.
- **Local planner**, to update the car position on the map and take decisions in the short time. This task computes the trajectory to be followed considering the current speed and angle between the vehicle and the target. The path shall be followed while avoiding obstacles.
- **Low-level actuation and control**, to provide the proper throttle and steering controls that must be effectively employed by the car to reach the target position.

1) DETECTION

Camera frames are fed into a detection task. This task is responsible for detecting and classifying the objects in the road. The plan is to use a state-of-the-art convolutional neural network (CNN) to perform object detection and classification. The CNN used is YOLO (You Only Look Once) version 3 [37], [38], re-trained from scratch on the Berkeley BDD100K dataset. The inference is performed on the FPGA and the output is composed of bounding boxes with an associated class. We currently support the following categories, but our system can easily be-retrained and customized: *Pedestrians, Cars, Trucks, Bus, Motorbikes, Bicycles, Riders, Traffic lights, Traffic signs*.

2) LOCALIZATION

A precise localization of the vehicle on a given map is crucial for navigation. This task has a high computational cost, therefore it should be as light and responsive as possible to always identify the vehicle position on the map. This task is in charge of matching the features extracted from the sensor with those stored on the map in a pre-mapping phase, but at the same time, it also has to provide the location of the car in the map. For this reason, this task is split into two subtasks: localization (best effort) and position filtering (critical). The best-effort localization is in charge of matching the current point cloud with the complete one (i.e., the one created in the mapping phase), matching the corresponding features. In our case, localization is performed only via LiDAR using the Generalized Iterative Closest Point (GICP) method, which adopts a probabilistic model to determine the position of the car on the given environmental map. This task is very computational expensive. Once the localization is obtained, the position is saved in a shared memory buffer to make it available for the filtering subtask. The position filter is implemented with an extended Kalman filter (EKF) running in the safety-critical partition. Specifically, the Kalman filter algorithm estimates the poses of the ego vehicle and objects repeating two stages: prediction and correction. The resulting

output is a matrix that corresponds to the estimated ego vehicle position. Finally, a *Particle Filter* localization algorithm is implemented to estimate the position of the vehicle in the map; this algorithm uses a Monte Carlo probabilistic model to determine the position of the car on the given map [39]. Such a position is merged with the motion estimation coming from the vehicle odometry, to obtain an accurate and predictive estimate of the vehicle's position. A nice property of the particle filter algorithm is that it is highly data-parallel, and thus it can greatly benefit from the embedded reconfigurable FPGA accelerators available in our onboard computing platform.

3) GLOBAL PLANNER

The global planner is the module that takes care of the whole mission, which is typically pre-loaded together with the map – for this reason, it is also often referred to as ''static planner''. Its purpose is to compute the high-level control signals (acceleration/speed and steering angle) to be sent to the vehicle to go from ''point A'' to ''point B'' in the map, under an optimal trajectory, assuming a priori knowledge of the environment and obstacles. As said, this task does not need to be executed very frequently, hence it has been implemented as a best effort low-criticality component.

4) LOCAL PLANNER

The main purpose of this component is to define the trajectory to follow while avoiding obstacles in a short time lapse. The local planner is one of the most important modules of the application, because it is responsible for the behavior of the vehicle and should be highly reactive to changes in the surrounding environment, such as moving objects. This task has been split into two parts: one in charge of trajectory sampling (best-effort, low criticality task) and one in charge of path following and collision avoidance (critical, highest priority task). The trajectory sampling task has to generate all the possible trajectories that the car can follow and select the most suitable one. Each trajectory is defined as a spline, that is, a line built through polynomial interpolation that represents, at each point, the position and orientation that the car will have to follow. The spline can be enriched with additional information, such as speed to hold, stop, priorities, etc. The path chosen is the closest to the optimal path while avoiding possible obstacles. The rest of the local planner is performed on the real-time partition. The path following task receives data from the trajectory sampling one and follows the chosen path with the Stanley Steering Control Algorithm. This method adjusts the current pose of a vehicle to match a reference pose that corresponds to the path computed in the previous phase. The collision check is computed on the real-time partition, using the occupancy grid given by the LiDAR, thanks to which the car speed can be adjusted to avoid obstacles.

5) ACTUATION

Actuation refers to the low-level control of the vehicle, in charge of sending commands to the car via CAN bus. This activity is implemented as a safety-critical periodic task with a period in the order of a millisecond. In particular, this task is implemented by two proportional–integral–derivative (PID) control loops, one acting to the throttle and one for controlling the car speed. Both output values are sent via the CAN bus in high-priority frame slots.

6) OTHER TASKS

Two additional tasks are present in the use case: a data collector task, which receives data from all the tasks described above, and a graphical user interface, which displays the data of interest. Both functions are implemented as best-effort tasks with low priority, running on the GNU/Linux partition. The GUI can also be deployed on a separate computing board, such as a tablet.

C. SOFTWARE PARTITIONING

1) TASK ANALYSIS

All the tasks involved in the AD stack are represented as a directed acyclic graph [40] illustrated in Figure 7, where each node of the graph is modeled as a time-triggered periodic task with a relative deadline equal to its period. The results produced by the tasks are exchanged through shared memory buffers to guarantee that the most recent data are always available to the other tasks for reading.

2) HARDWARE AND SYSTEM REQUIREMENTS

Each node of the graph in Figure 7 reports the name of the task and its corresponding period in milliseconds. The green-dashed circle represents the workload executed on the FPGA, while the black ones represent best-effort tasks that run on the CPU.

FPGA tasks can either consist of computational activities purely implemented in hardware or be triggered by software tasks to accelerate a portion of computation. In the first case, the FPGA task is periodically triggered by a timer, still implemented in hardware. In the second case, a software task first passes data to the FPGA accelerator, then triggers the acceleration request, and finally gets the produced results and completes on the CPU. Therefore, the periodicity is guaranteed by the execution paradigm since software tasks are periodically scheduled on the CPU.

To guarantee the timing constraints of the application, tasks are separated into best-effort tasks and critical tasks. The former runs on a best-effort operating system, like GNU/Linux, while the latter executes on an RTOS, as Erika v3. Real-time tasks are represented as red-dotted nodes.

3) SPHERE SYSTEM FRAMEWORK

The application configuration described so far relies on the whole SPHERE framework. Firstly, its real-time properties are ensured by the RTOSes ecosystem, which in turn leverages the strong partitioning mechanisms provided by the SPHERE hypervisors used in the project. Secondly, the TSN virtualization support is the key enabler for the

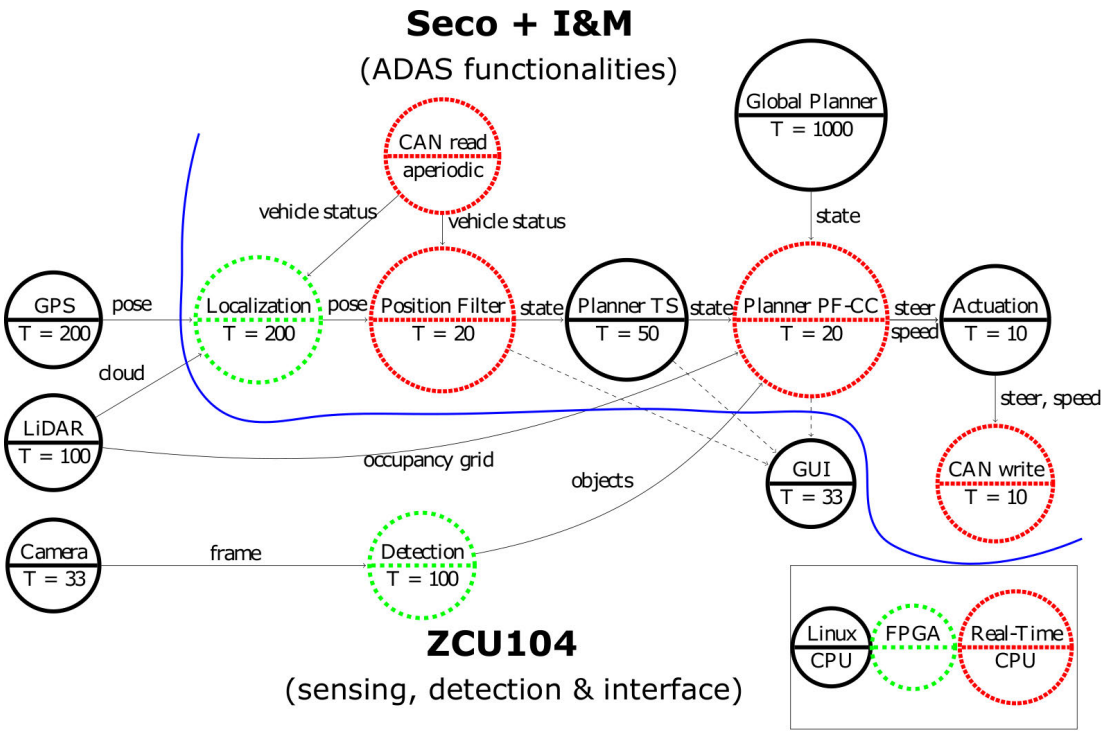


FIGURE 7. Application graph with hardware and software partitioning.

mixed-criticality communication between the two hardware platforms, among which the software stack is split, as explained by the next subsection.

D. TASK PARTITIONING

To proper balance the workload of the system, the software components are partitioned into two different hardware platforms:

- **SECO+I&M**. It is composed by a SECO SMB71 SMARC module [41], including a ZU4-class Xilinx Zynq UltraScale+ MPSoC platform, hosted on an Ideas and Motion autonomous driving application SMARC carrier board [42]. It provides the edge functionalities and interfacing for ADAS software components.
- **ZCU104**. It is an evaluation board from Xilinx with a ZU7 MPSoC [43]. It is in charge of the highly demanding object detection jobs. It also hosts the TSN switch and the relocatable FPGA design that has been presented in VI.A (not depicted in the figure).

These two platforms exchange information (depicted in Figure 7 with black dashed and solid edges) through a single TSN link, which manages the messages with a prioritization mechanism. The messages passed from the ZCU104 to the SECO+I&M platform through the TSN switch are: (1) the objects detected by the neural network, which are given to the path planning component; (2) the GPS information; (3) the LiDAR point cloud to the localization component; and 4) the LiDAR occupancy grid. On the other hand, the information produced by the SECO+I&M (i.e., the position of the car within the map) is sent to the GUI to be displayed. The proposed partitioning allows establishing a prioritized data exchange. In particular, the data with higher priority are those provided by the sensors, being the point cloud, the detected objects, and the occupancy grid the most critical ones (solid lines), whereas the GUI data are those with the lowest priority (dashed lines).

The innermost partitioning level, inside the single SoC, is managed by the hypervisor. The RTOS and Linux VMs are hosted on different CPU core sets of the SECO+I&M platform, where the hypervisor guarantees the isolation properties that are needed to obtain the real-time constraints described above. Tasks with different criticality can thus safely execute together without interference and without sacrificing performance.

VII. CONCLUSION

This work presented SPHERE, a framework to abstract the hardware complexity of systems composed of multiple heterogeneous system-on-chips and simplify the management of their computational resources. The paper discussed the use of hypervisor technology to virtualize computational resources and isolate the behavior of different subsystems running on the same platform, and presented new techniques to enable advanced isolation and virtualization capabilities. The management of time-sensitive networks in the presence of traffic flow with different temporal constraints has also been discussed together with techniques to manage the FPGA. The framework has been realized upon the Xilinx Ultrascale+ by Xilinx, employing a TSN switch by Soc-e. The paper finally presented a case study focused on autonomous driving used to validate the SPHERE framework. Future work will report on the experimental evaluation of the framework, e.g.,

by evaluating the system load for the presented task set, its schedulability analysis, and measuring the delays due to the framework. Furthermore, the consideration of additional requirements, e.g., related to system dependability and fault tolerance, is a research direction that will be addressed in the future due to its relevance for the considered application.

REFERENCES

[1] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, "A survey on cache management mechanisms for real-time embedded systems," *ACM Comput. Surveys*, vol. 48, no. 2, pp. 1–36, Nov. 2015.

[2] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 239–252.

[3] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The XtratuM approach," in *Proc. Eur. Dependable Comput. Conf.*, 2010, pp. 67–72.

[4] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 1–14.

[5] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2018, pp. 1651–1657.

[6] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 55–64.

[7] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "AXI HyperConnect: A predictable, hypervisor-level interconnect for hardware accelerators in FPGA SoC," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jul. 2020, pp. 1–6.

[8] *Clare Software Stack*. Accessed: Feb. 6, 2021. [Online]. Available: http://clare.santannapisa.it

[9] J. Kiszka and Other Community Contributors. (2020). *Jailhouse: Linux-Based Partitioning Hypervisor*. [Online]. Available: https://github.com/siemens/jailhouse

[10] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no VM exits! (Almost)," 2017, *arXiv:1705.06932*. [Online]. Available: http://arxiv.org/abs/1705.06932

[11] M. Solieri, T. Kloda, M. Bertogna, M. Sojka, and M. Baryshnikov, *D5.3: Integrated Schedulability Analysis*, document Deliverable of the HERCULES project 12 2018.

[12] M. Torquati, "Single-producer/single-consumer queues on shared cache multi-core systems," 2010, *arXiv:1012.1824*. [Online]. Available: http://arxiv.org/abs/1012.1824

[13] D. Casini, A. Biondi, G. Cicero, and G. Buttazzo, "Latency analysis of I/O virtualization techniques in hypervisor-based real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Jun. 2021.

[14] A. Alkamil and D. G. Perera, "Towards dynamic and partial reconfigurable hardware architectures for cryptographic algorithms on embedded devices," *IEEE Access*, vol. 8, pp. 221720–221742, 2020.

[15] H.-G. Vu, T. Nakada, and Y. Nakashima, "Efficient multitasking on FPGA using HDL-based checkpointing," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Eds. Cham, Switzerland: Springer, 2018, pp. 590–602.

[16] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 1–12.

[17] M. Pagani, A. Balsini, A. Biondi, M. Marinoni, and G. Buttazzo, "A linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration," in *Proc. 30th IEEE Int. Syst. Chip Conf. (SOCC)*, Sep. 2017, pp. 96–101.

[18] E. M. Benhani, C. M. Lopez, and L. Bossuet, "Secure internal communication of a trustzone-enabled heterogeneous soc lightweight encryption," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2019, pp. 239–242.

[19] L. Jiao, Y. Hao, and D. Feng, "Stream cipher designs: A review," *Sci. China Inf. Sci.*, vol. 63, no. 3, pp. 1–25, Mar. 2020.

[20] L. Lo Bello and W. Steiner, "A perspective on IEEE time-sensitive networking for industrial communication and automation systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1094–1120, Jun. 2019.

[21] J. Lee and S. Park, "Time-sensitive network (TSN) experiment in sensor-based integrated environment for autonomous driving," *Sensors*, vol. 19, no. 5, p. 1111, Mar. 2019.

[22] L. L. Bello, "Novel trends in automotive networks: A perspective on Ethernet and the IEEE audio video bridging," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–8.

[23] J. Klaus-Wagenbrenner, "Zonal ee architecture: Towards a fully automotive Ethernet-based vehicle infrastructure," Tech. Rep., 2019. [Online]. Available: https://ieee802.org/1/files/public/docs2019/dg-zinner-automotive-architecture-evolution-0319-v02.pdf

[24] *IEEE Standard for Local and Metropolitan Area Networks-Bridges and Bridged Networks*, IEEE Standard 802.1Q-2018 May 2018.

[25] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 183–192.

[26] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Porto, Portugal, Apr. 2020, pp. 27–29.

[27] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello, "Schedulability analysis of Ethernet audio video bridging networks with scheduled traffic support," *Real-Time Syst.*, vol. 53, no. 4, pp. 526–577, Jul. 2017.

[28] J. Lv, Y. Zhao, X. Wu, Y. Li, and Q. Wang, "Formal analysis of TSN scheduler for real-time communications," *IEEE Trans. Rel.*, early access, Oct. 8, 2020, doi: 10.1109/TR.2020.3026689.

[29] L. Lo Bello, M. Ashjaei, G. Patti, and M. Behnam, "Schedulability analysis of time-sensitive networks with scheduled traffic and preemption support," *J. Parallel Distrib. Comput.*, vol. 144, pp. 153–171, Oct. 2020.

[30] SoC-e. (2020). *1G MTSN—Multiport TSN Switch IP Core*. [Online]. Available: https://soc-e.com/mtsn-multiport-tsn-switch-ip-core

[31] D. Hallmans, M. Ashjaei, and T. Nolte, "Analysis of the TSN standards for utilization in long-life industrial distributed control systems," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Vienna, Austria, Sep. 2020, pp. 190–197.

[32] *Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit*, Xilinx, San Jose, CA, USA, 2020.

[33] L. Leonardi, L. L. Bello, and G. Patti, "Towards time-sensitive networking in heterogeneous platforms with virtualization," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Vienna, Austria, Sep. 2020, pp. 1155–1158.

[34] *Sekonix Cameras*. Accessed: Feb. 19, 2021. [Online]. Available: http://sekolab.com/products/camera/

[35] *Ouster OS1 Mid-Range Lidar Sensor*. Accessed: Feb. 19, 2021. [Online]. Available: https://ouster.com/products/os1-lidar-sensor/

[36] *Continental Advanced Radar Sensor 430*. Accessed: Feb. 19, 2021. [Online]. Available: https://www.continental-automotive.com/en-gl/Trucks-Buses/Vehicle-Chassis-Body/safety-topics/Product-Portfolio/Advanced-Radar-Sensor-%E2%80%93-ARS430-CV

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[38] M. Verucchi, G. Brilli, D. Sapienza, M. Verasani, M. Arena, F. Gatti, A. Capotondi, R. Cavicchioli, M. Bertogna, and M. Solieri, "A systematic assessment of embedded neural networks for object detection," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 937–944.

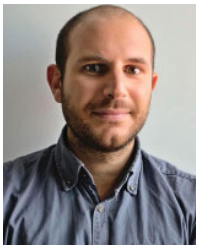
[39] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 4372–4378.

[40] F. Wurst, D. Dasari, A. Hamann, D. Ziegenbein, I. Sanudo, N. Capodieci, M. Bertogna, and P. Burgio, "System performance modelling of heterogeneous HW platforms: An automated driving case study," in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Kallithea, Greece, Aug. 2019, pp. 365–372.

[41] *Seco SMARC Module SM-b71*. Accessed: Feb. 19, 2021. [Online]. Available: https://www.seco.com/en/products/sm-b71

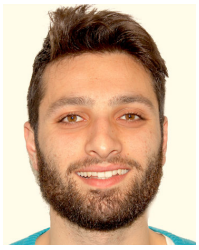
[42] *Ideas & Motion*. Accessed: Feb. 19, 2021. [Online]. Available: https://www.ideasandmotion.com/

[43] *ZYNQ Ultrascale+ MPSOC ZCU104 Evaluation Kit*. Accessed: Feb. 19, 2021. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html



ALESSANDRO BIONDI (Member, IEEE) is Assistant Professor at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. He graduated *(cum laude)* in Computer Engineering at the University of Pisa, Italy, within the excellence program, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

DANIEL CASINI (Member, IEEE) is Postdoctoral Researcher at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. He graduated *(cum laude)* in Embedded Computing Systems Engineering, a Master degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa, and received a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna of Pisa (with honors), working under the supervision of Prof. Alessandro Biondi and Prof. Giorgio Buttazzo. In 2019, he has been visiting scholar at the Max Planck Institute for Software Systems (Germany). His research interests include software predictability in multi-processor systems, schedulability analysis, synchronization protocols, and the design and implementation of real-time operating systems and hypervisors.



GIORGIOMARIA CICERO is a Senior Research Fellow at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. He graduated *(cum laude)* in Embedded Computing Systems Engineering, a Master degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa. In 2015, he has been visiting trainee at the European Space Agency (ESTEC, Netherlands). His research interests include software predictability in multi-processor systems and heterogeneous platforms, system-level cyber-security hardening techniques, and design and implementation of realtime operating systems and hypervisors.

NICCOLÒ BORGIOLI is a Ph.D. student at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. He graduated *(cum laude)* in Embedded Computing Systems Engineering, a Master's Degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa. His research interests include the design and implementation of real-time operating systems and hypervisors, cyber-physical systems, machine learning, and cybersecurity.

GIORGIO BUTTAZZO (Fellow, IEEE) is full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa, received a M.S. degree in Computer Science at the University of Pennsylvania, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He is Editor-in-Chief of *Real-Time Systems*, Associate Editor of the *ACM Transactions on Cyber-Physical Systems*, and IEEE fellow since 2012. He has authored 7 books on real-time systems and more than 300 papers in the field of real-time systems, robotics, and neural networks, receiving 11 best paper awards.

GAETANO PATTI (Member, IEEE) received the M.S. and the Ph.D. degrees from the University of Catania, Catania, Italy, in 2013 and 2017, respectively. He is a Postdoctoral Researcher with the Department of Electrical, Electronics, and Computer Engineering, University of Catania. Since 2012, he has been working in the field of real-time industrial networks, automotive networks, wireless sensor and actuator networks (WSANs), powerline communications, and networks for mobile robot applications. He coauthored more than 30 papers published in the proceedings of international conferences and in important international journals. Dr. Patti is currently a Member of the IEEE Industrial Electronics Society Technical Committee on Factory Automation (Subcommittee on In-Vehicle Embedded Systems).

LUCA LEONARDI (Member, IEEE) received the M.S degree *(summa cum laude)* in computer engineering and the Ph.D. degree in systems, energy, computer and telecommunications engineering from the University of Catania, Catania, Italy, in 2016 and 2020, respectively. He is a Postdoctoral Researcher with the Department of Electrical, Electronics, and Computer Engineering, University of Catania. Since 2015, he has been researching in the field of realtime wireless communications, Wireless Sensor Networks (WSNs), Low Power Wide Area Networks (LPWANs), and Industrial Internet of Things (IIoT). He coauthored about 15 papers published in the proceedings of international conferences and in important international journals.

LUCIA LO BELLO (Senior Member, IEEE) (Ph.D.) is tenured Associate Professor with the Department of Electrical, Electronic and Computer Engineering, University of Catania, Italy. She was Guest Professor at the University of Malardalen, Sweden (2014), and Visiting Researcher with the Department of Computer Engineering, Seoul National University, Korea (2000- 2001). In the year 2008 she was the recipient of the IEEE Industrial Electronics Society Early Career Award. She authored or coauthored more than 170 technical papers in the area of wireless sensor networks, automotive communications, industrial networks for Industry 4.0, and real-time embedded systems. She is responsible for the University of Catania of several international and national projects and research grants funded by companies. She is Senior member of the IEEE and the current Secretary of the IEEE Industrial Electronics Society.

MARCO SOLIERI (Member, IEEE) is a postdoctoral researcher of the HiPeRT Lab at the Università di Modena e Reggio Emilia (IT), where he leads a group of around 12 students and junior researchers. His research interests are in operating systems and hypervisors for real-time and embedded systems, with a strong accent to industrial transfer for automotive and automation verticals. He received in 2016 a double Ph.D. in computer science from both the Université Sorbonne Paris Nord and the Università di Bologna, and has been also with the Université de Paris and the University of Bath.

PAOLO BURGIO (Member, IEEE) got a Ph.D in Electronics Engineering jointly between the University of Bologna and the University of Southern-Brittany, in 2013. His research topics are next-generation predictable systems based on heterogeneous many-cores and GPGPUs, with an eye on compilers, and parallel programming models. Since 2014 he joined HiPeRT Lab at Univ. of Modena, where he currently coordinates the project on autonomous driving systems. He is currently involved in the Prystine and Fractal EU projects.

IGNACIO SANUDO OLMEDO (Member, IEEE) received his Ph.D. in Computer Science at the University of Modena in 2018. He is currently a Post-Doctoral Researcher at the High-Performance Real-Time (HiPeRT) Lab at the same university. His research interests include safety, reliability in hard real-time systems and self-driving vehicles.

ANGELO RUOCCO is a bachelor student in computer science at the University of Modena and Reggio Emilia. His research is focused on hypervisors and runtimes for embedded systems.

LUCA PALAZZI is a bachelor student in computer science at the University of Modena and Reggio Emilia. His research is focused on hypervisors and networking for embedded systems.

MARKO BERTOGNA (Senior Member, IEEE) is Full Professor at the University of Modena and Reggio Emilia, where he founded the HiPeRT Lab. His main research interests are in High-Performance Real-Time systems, especially based on multi- and many-core devices, Autonomous Driving and Industrial Automation systems. In 2008, he received a Ph.D. in Computer Sciences from the Scuola Superiore Sant'Anna of Pisa. He has authored more than 100 papers, receiving 10 Best Paper Awards in first level international conferences and journals. He coordinated multiple EU and industrial projects, securing more than 10 MEuro in funding for his research group. He served in more than 60 program committees of international conferences in embedded and realtime systems, chairing or co-chairing 8 events. He is Senior Member of the IEEE, and Stakeholder Member of HiPEAC. He is CEO and founder of the academic spinoff HiPeRT Srl.

ALESSANDRO CILARDO (Senior Member, IEEE) is currently an associate professor at the University of Naples Federico II. He received a five-year degree in Electronics Engineering, *magna cum laude*, in 2003, and a Ph.D. degree in Computer Science in November 2006. He is the single or main author of around 75 peer-reviewed papers published in leading scientific journals and conferences, including various IEEE and ACM transactions, as well as top conferences like DATE and FPL. His research focuses on digital design methodologies and the application of programming paradigms and tools from the parallel computing domain to electronic system-level design. A further research activity in the area of computer arithmetic targets the application domain of security and cryptography-related processing. He is involved in a number of funded projects at both the national level and the European level (7FP and H2020 projects). He is a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE) and the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC).

NICOLA MAZZOCCA (Senior Member, IEEE) is currently a Professor of High- Performance and Reliable Computing at the Computer and System Engineering Department of the University of Naples Federico II, Italy. He owns an MSc Degree in Electronic Engineering and a Ph.D. in Computer Engineering, both from the University of Naples Federico II. He authored over 200 papers in international journals, books, and international conferences in the field of computer architecture, reliable and secure systems, distributed systems, and performance evaluation in high-performance systems. His research activities include methodologies and tools for design/analysis of distributed systems; techniques for modelling and analysis of distributed heterogeneous systems and communication networks; secure and real-time systems; distributed control applications; models and tools for configuration and performance evaluation of distributed, heterogeneous systems and communication networks; dedicated parallel architectures.

ANTONINO MAZZEO (Senior Member, IEEE) is an Emeritus Professor with the University of Naples Federico II, Italy, where he is a teacher in computer architecture. He led major research programs in conjunction with international universities, research agencies (CNR, ASI, and EC), and technology leading industries in Italy and abroad. He has wide experience in the field of complex systems modeling, embedded systems, general and special purpose parallel architectures, and security and performance evaluation.