Virtualizing Real-Time Processing Units in Multi-Processor Systems-on-Chip

Marcello Cinque, Gianmaria De Tommasi, Sara Dubbioso, Daniele Ottaviano
Dept. of Electrical Engineering and Information Technology
Federico II University of Naples
Naples, Italy
{macinque,detommas,sara.dubbioso}@unina.it, da.ottaviano@studenti.unina.it

***Abstract*—Emerging Multiprocessor System-on-Chip (MPSoC) platforms, featuring asymmetric multi-processing (AMP), are becoming the underlying tool to develop real-time embedded mixed-criticality systems. These type of systems are key enablers for the emerging Industry 4.0 and Industrial Internet of Things revolution. Virtualization technologies can even more boost the opportunities brought by MPSoCs, providing unique isolation and flexibility features in next generation embedded devices. However, the management of hardware accelerators, such as real-time processing units (RPUs) available on today's MPSoCs, is still an open challenge. In this paper, we propose a virtualization-based architectural solution to RPU sharing in modern MPSoCs. The aim is to isolate the execution on RPUs of many tasks offloaded by several different VMs running on traditional application processors on the same chip. The practical implications of such architecture are discussed in the context of a case study regarding the real-time plasma vertical stabilization control in the ITER[1] experimental fusion power plant.**

***Index Terms*—Mixed Criticality, Real-Time virtualization, MPSoC, Real-time Processing Unit, Fusion engineering**

I. Introduction

High level of integration in system on chip is made possible by the development of increasingly advanced hardware production technologies. This explains the possibility to implement heterogeneous components on a single piece of silicon, making the emerging Multi-Processor Systems on Chip (MPSoCs) [1] capable of guaranteeing performance, scalability, programmability, and reconfigurability requirements that are simply out of reach for traditional Programmable Logic Controllers (PLCs) used in industrial environments.

As an example, the Xilnx's MPSoC named Zynq® UltraScale+™ is a widely used MPSoC [2], providing 64-bit processor scalability while combining real-time control with soft and hard engines for graphics, video, waveform, and packet processing. This is made possible due to the presence of numerous hardware components and heterogeneous processor units on the same hardware, such as Application Processing Units (APUs, e.g., quad-core ARM Cortex-A5), Real-Time Processing Units (RPUs, e.g., dual-core ARM Cortex-R5F), re-configurable hardware (e.g., 16nm FinFET+ Programmable Logic), and Graphical Processing Units (GPUs, such as, ARM Mali-400MP2).

[1]ITER is a Nuclear Facility INB-174. The views and opinions expressed herein do not necessarily reflect those of the ITER Organization.

On the software side, virtualization is a key technology to get the most from recent MPSoCs. While virtualization has revolutionized general-purpose computing and cloud applications in the past decade [3], ensuring the reduction of operation costs, server consolidation, flexible system configuration and elastic resource provisioning, the adoption of virtualization approaches in embedded systems has been considered only in recent years.

Traditionally, the pain of virtualization in the embedded space was caused by the lack of appropriate hardware resources to make the solution easy to implement while providing satisfactory performance. The ARMv8 architecture, included in new MPSoCs, such as the above-mentioned Zynq UltraScale+, enables true hardware accelerated virtualization to alleviate implementation problems.

There are many reasons for using virtualization technologies on an MPSoC. First, the hypervisor (or Virtual Machine Monitor, VMM) can carefully manage the system loading providing processors to remain as fully loaded as possible for performance and timeliness specifications. That is a common challenge for embedded systems but is not easy to achieve it without the supervision work of an hypervisor, that can assign spare resources to new virtual machines (VMs), without affecting the ones already deployed. In addition, hypervisors can help to streamline application development and operation, making it possible to apply the same technologies and know-how of typical cloud computing setups to embedded systems. Finally, virtualization improves fault-tolerance trough transparend redundancy. During the execution of a critical task run by a real-time operating system (RTOS), the hypervisor can monitor the system detecting failures and can then either restart the RTOS or start up a passive or active back-up instance, to minimize or eliminate downtime of critical services.

A relevant challenge in this context is the need for isolation and partitioning. Isolation requirements are imposed by industrial safety standards, to make virtualized environments not interfere each other while running simultaneously on the same board. This is in line with the concept of Mixed-Criticality System (MCS) [4], enabling to run hard real time (and safety-critical) tasks together with compute intensive non-real-time (and non-critical) tasks on the same physical hardware.

While virtualization solutions are available to realize MCSes on standard embedded boards, i.e., based on symmetric multi-

processing (SMP) (such as, ARM Cortex-A), the advent of MPSoCs exacerbate isolation issues. Suffice to think about the use of shared and heterogeneous hardware *accelerators*, i.e., RPUs, GPUs, and the programmable logic. In other terms, it is still unclear how multiple VMs, running on APUs, can simultaneously and transparently use accelerators, without interfering each other. In this direction, first attempts to virtualize the programmable logic or the communication between processors are emerging in hypervisors, as we discuss in section II. However, to the best of our knowledge, no solutions to date are available regarding the sharing of RPUs among multiple running VMs, while facing isolation problems. So, if two VMs are willing to (simultaneously) offload real-time processing to the RPUs, the resulting behavior can be unpredictable, without the proper coordination and abstraction provided by the hypervisor.

In this paper, we propose an architectural solution to this problem, based on the use of the Xen hypervisor and OpenAMP, a framework for the communication among heterogeneous processors. The basic idea is to intercept, at hypervisor level, the requests for RPU usage coming from VMs hosted on the APUs, and transparently orchestrate them via OpenAMP. On the RPU side, we envision the use of an RTOS to run the OpenAMP counterpart of the real-time hypervisor and orchestrate real-time tasks. We reason about possible implications regarding to the use of MPSoCs, hypervisors, and RPUs, in large-scale real-time processing infrastructures in the compelling scenario of nuclear fusion power real-time control of the ITER tokamak. In particular, we deal with the use case of plasma vertical stabilization control, which requires to run multiple control loops, with different criticality levels, on the same hardware.

The rest of the paper is organized as follows. In section II we reiview the state of the art regarding the use of virtualization technologies in MPSoCs. Section III describes the proposed architectural solution, whereas Section IV presents the ITER case study. Conclusive remarks are provided in Section V.

II. Related Work

The recent trend of using virtualization on embedded systems and Internet of Things has seen a proliferation of solutions. These depend on the particular domain-specific constraints and, in most cases, cannot rely on solutions born for cloud environments. Hence, either completely new platforms have been proposed, or existing ones have been completely re-targeted to the new scenario. Despite these efforts, only a few solutions are looking into the opportunity provided by MPSoCs.

An example is Xen, one of the most adopted open-source hypervisors in cloud computing environments, recently ported to Xilinx's MPSoCs [2]. Xen is an open-source type-1 or bare-metal hypervisor, which makes it possible to run many instances of an operating system or indeed different operating systems in parallel on a single machine (or host). Xen is the only type-1 hypervisor that is available as open source, and it is used as the basis for several different commercial and open-source applications. In proof of this, Amazon Web Services alone runs million Xen instances. While Xen's traditional architecture has been x86-compatible, recent hosts development has made it a robust solution on ARM architectures as well. Xen takes full advantage of ARMv8's underlying virtualization hardware, and provides support to ARM's memory management and partitioning. Recent Xen versions include the Real-Time Deferrable Server (RTDS) scheduler, to provide guaranteed CPU capacity to guest VMs [5]. Also, Xen includes the support for integrating Programmable Logic devices, such as FPGA cards [6], making it possible to share the same hardware device across multiple VMs. However, this solution is only available for server machines in cloud environments. In addition, no support is still available in XEN to share RPUs in an asymmetric multiprocessing environment.

Another example is represented by hypervisors specifically targeting embedded systems, such as Jailhouse [7]. It is a partitioning hypervisor, based on Linux, with a strong focus on isolation guarantees to VMs, which are called *cells* in the Jailhouse jargon. A cell can be seen as a strongly isolated domain, with its own static split of CPU, memory, I/O devices, etc. Regarding MPSoCs, Jailhouse provides support to access FPGA-based accelerators from cells, using a framework called OpenAMP [8], which includes mechanisms, such as Remoteproc and RPMsg (see next section for more details), to enable the communication between asymmetric cores in a multiprocessing environment. Despite these efforts, also Jailhouse does not explicitly address the problem of RPU sharing.

III. The Proposed Architectural Solution

The aim of our research is to investigate how RPUs resources can be transparently shared between VMs that run over APUs, using the underlying hypervisor. A typical scenario would be the presence of two (or more) VMs, willing to offload real-time computation on the RPU. On classical Linux-based deploys, the user would simply send commands to the RPU, expecting results in due time and assuming to be the only "client" of the RPU. The question is: if more VMs are present, using RPUs, *how can we keep the same user-experience, in terms of RPU usage, while assuring isolation in terms of request/response load and processing to be conducted on RPUs?*

A. Current practice: OpenAMP

To answer the question, the first aspect to be analyzed is how practitioners use RPU on Asymetric Multi-Processing (AMP) architectures, as we aim to provide developers with the same experience, even when running the code within a VM.

AMP architectures typically entail a combination of dissimilar software environments such as Linux, a real-time operating system (RTOS), or bare-metal running on homogeneous or heterogeneous processing cores present in an MPSoC, all working in concert to achieve the design goals of the end application. Using hypervisors, such as Xen, it is possible to

run several OSes as VMs on APUs, but it is not possible to run Xen on top of an RPU. The latter typically run a bare-metal application or at least an RTOS that is not controlled by a hypervisor. To effectively deal with the complexities of managing life cycle of several different operating systems on dissimilar processors, and to provide an enabling Inter Processor Communications (IPC) infrastructure for offloading compute workload, new and improved software capabilities and methods are being studied [9]. The most recent technique resides in using OpenAMP [8] providing the software components needed to enable the development of software applications for Asymmetric Multiprocessing (AMP) systems.

As anticipated in section II, OpenAMP uses two main framework components already present in Linux Kernel from 2011 for managing and massaging between heterogeneous processors: remoteproc (Remote Processor) and RPMs (Remote Processor Messaging). The former is a framework that allows a Linux master to control and manage the life cycle of remote processors and their associated software contexts to enable control of the reset, load, execute and reboot states of the processors and cores, while the latter is a Messaging framework that provides inter-processor communication (IPC) between kernel drivers and remote processors in AMP environments.

The RPMsg in LinuxAMP implicitly assumes that Linux will always be the master operating system and does not support Linux as remote OS in an AMP configuration. Further, the remoteproc and RPMsg APIs are available from Linux kernel space only and there is no equivalent API or library usable with other OSs and run-times. A typical setup can be observed in Fig. 1.

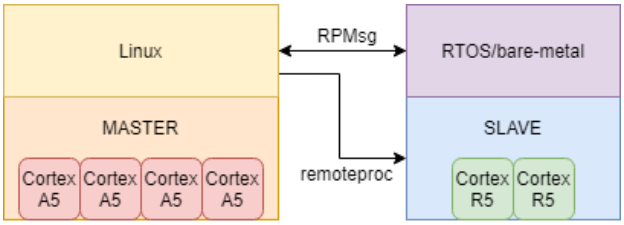

Fig. 1. LinuxAMP typical setup.

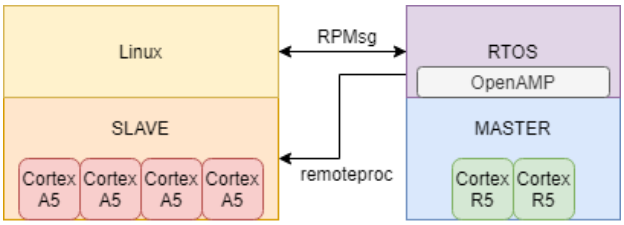

Fig. 2. OpenAMP possible setup.

The OpenAMP framework was introduced to expand the scope of the original Linux AMP framework providing standalone library written in C language that implements remoteproc and RPMsg functionality usable with RTOS or Bare-metal software environments, with API level compatibility and functional symmetry to its Linux counterpart. In this way it is possible to create another type of setup as represented in Fig. 2.

While these setups are the current practice to offload processing to RPUs (the Cortex R5 processors in the figure), this is still not enough to assure isolation and transparency to VMs running on APUs (the Cortex A5 processors in the figure).

B. The proposed solution: XEN over OpenAMP

To complete the answer to the question above, we propose to combine the virtualization paradigm with the possibility offered by OpenAMP of communication between heterogeneous processors, with the aim of create a mechanism for sharing RPU's computation resources between VMs.

As target hypervisor, we assume to use XEN for many reasons: (i) it is open-source and thus modifiable; (ii) it has already been ported on ARM and on MPSoCs such as Xilinx' Ultrascale+; (iii) it already implements real-time extensions useful for mixed-criticality systems; and (iv) it is based on Linux, hence it can be easily integrated with OpenAMP.

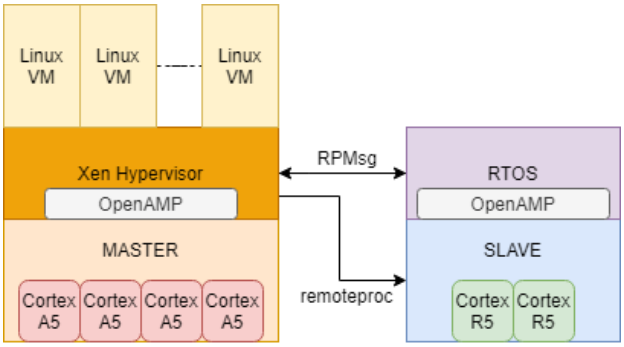

Fig. 3. Xen over OpenAMP setup.

To achieve isolation and resource sharing of the RPU's computation resources, the framework provided by OpenAMP can be used from within the hypervisor for supervised management of heterogeneous compute resources, allowing the hypervisor to supervise interactions between the VMs running on it and the software running on RPU (see Fig. 3). The idea consists to implement a new framework that wraps the one offered by OpenAMP, adding algorithms for real-time control of requests (in terms of bandwidth regulation, with reservation approaches) coming from VMs. If the request sent by a VM is schedulable by the RTOS running on RPUs, then it will be sent through RPMsg otherwise it will be deferred, ensuring that the RPU will never be overloaded. It is worth noting that applications run in VMs can still use the same API to communicate with the RPU as if they were running bare-metal on APUs, hence achieving transparency for developers.

On the RPU side, the approach for sharing the resource consists in creating a specific firmware for the RPU containing not only the RTOS files but also the code of all the tasks that VMs, running on APUs, may request during their execution. The firmware will be inserted into the OS root file system which will be loaded on the APUs part. Immediately

after system startup, together with the creation of VMs, the firmware previously created will be loaded on the RPU via remoteproc by the hypervisor. Doing so, once the system is started, each virtual machine can request the execution of a real time task. This request is captured by the hypervisor which must demand for the current load on the RPU, using RPMsg, so as to decide whether the requested task can be performed without the risk of exceeding the time constraints of the other tasks currently running on RPU. If so, the hypervisor will be able to use the primitives offered by OpenAMP to start the requested task on RPU and manage the communication with the requesting VM, which will remain unaware of all the underlying communication. On the other hand, if the requested task turns out to be infeasible, the requesting VM would receive an error message to avoid overloading the RPU. It is worth pointing out that the constraint to deploy all the needed tasks in advance on the RPU is not a strong requirement. Typically, in embedded systems, tasks which will be performed are well known before the execution, especially in a safety critical environments with strict temporal requirements. In addition, if new tasks or improved versions of existing tasks need to be added, it is always possible to stop the execution and reload the RPU with a new firmware.

Using the APIs offered by OpenAMP is also possible an alternative deployment, making the RPU act as master. In this scenario, the RTOS can be regarded as a safety-critical monitor for VMs running on APUs. The RTOS will handle critical system functionality requested from VMs and will manage several Linux contexts handling non-critical system functions. Upon failure of one the Linux-based subsystem, the RTOS can simply re-boot the failed subsystem without causing any adverse effects to critical system functions.

IV. The case of the ITER real-time vertical stabilization systems

Tokamaks are one of the most promising experimental devices aimed at proving the feasibility of energy production by means of nuclear fusion on Earth [12]. In a tokamak, a fully ionized gas of hydrogen ions, called *plasma*, is confined by magnetic fields and heated to temperatures up to hundreds millions of degrees [13]. At such high temperatures, collisions between ions can overcome the Coulomb repulsive forces, resulting in fusion reactions [14]. Since the mid 70s, many International projects have been successfully established to build and operate tokamak machines all around the world. ITER [15], which is currently under construction in France, represents the most ambitious one, and includes the joint efforts of EU, India, People's Republic of China, Russia, South Korea and USA. ITER will be the first magnetic confinement device to produce a net surplus of fusion energy and its first plasma is envisaged in 2025.

In large experimental plants such as ITER, there are some common requirements that have to be taken into account during the design of the real-time infrastructure [16], [17]. In particular, on the hardware side, such an infrastructure should:

- cope with the unavoidable hardware obsolescence and maintenance requirements that are expected during the life cycle of an experiment spanning several decades, whilst attempting to be as *hardware independent* as possible;
- manage the expected increase in computational requirements by being scalable;
- allow sharing of the resources between the processing nodes (e.g. to share the plant measurements and the outputs of each processing node).

Moreover, on the software side, it is crucial to have an architecture that:

- supports and facilitates the test and validation, by establishing strict and well defined boundaries between the application algorithm and the interfaces with other plant systems;
- supports model-based development in order to be able to validate software components against plant models and minimize the risks and commissioning/debugging efforts when developing complex plant control systems;
- supports the development of model-free and data-driven control algorithms that include different loops, with different sampling times, and different requirements in terms of reliability.

The latter requirement calls for an architecture based on a mixed-criticality system (MCS) to implement integrated control systems with different levels of criticality. Possible control approaches that belong to this category are those ones based on *Extremum Seeking* [18] and on *Reinforcement Learning* [19]. Hybrid approaches, where model-based adaptive control schemes are augmented with neural networks that are continuously trained by a loop with lower priority, also calls for mixed-criticality architectures.

An example of advanced control system for the ITER tokamak that may rely on a mixed-criticality real-time infrastructure, is the Vertical Stabilization (VS) one, which stabilizes the plasma column in the vacuum chamber and counteracts all the relevant disturbances. Indeed, high performance plasmas currently run on tokamaks have an elongated poloidal cross-section that make them vertically unstable. Therefore, an active VS system is needed to run the discharges (for more details, the interested reader may refer to [10]).

Model-based approaches have been proposed in literature to solve the VS problem. Despite the different proposed solutions, the performance of any existing VS system strongly depends on the so called plasma *growth-rate* γ. Therefore, a possible approach would be to adapt the control gains as function of γ. However, the real-time estimation of γ is based on the plasma equilibrium reconstruction [11], which is still a computationally demanding task, if compared with the time scale the VS system should react in [20]. Therefore such plasma equilibrium algorithms should run as low-priority task in a MCS.

Moreover, nowadays computational capability are paving the way for data-driven control approaches to solve the VS

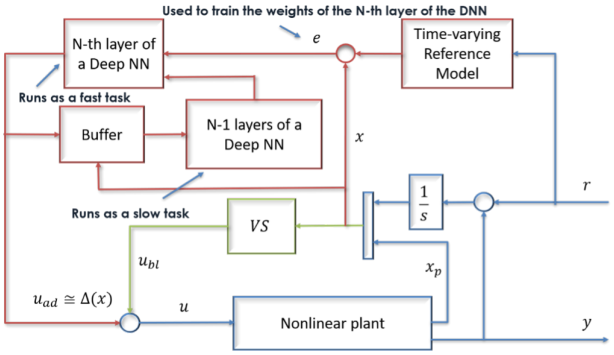

Fig. 4. Deep Model Reference Adaptive Vertical Stabilization System.

problem that also call for MCS. One possible architecture is the one shown in Fig. 4, which is based on a scheme originally presented in [21]. Here a *Model Reference Adaptive Control* is augmented by a *Deep Neutral Networks* that identifies in real-time the significant nonlinearities and uncertainties. More in details, the update of the weights for the neural part of the proposed architecture is performed in real-time at different sampling times: the faster loop deals with the update of the weights of the outer layer, which runs at the same sampling time as of the baseline controller. on the other hand, the weights of the inner layers are adapted at a slower sampling rate, by using data batches for the update. Therefore, this architecture includes at least three different control loops, with different level of criticality, being the baseline controller the most critical one. There is then the need to guarantee, at the implementation level, temporal separation and fault isolation of the tasks, to avoid that a problem or a delay in a low criticality task, as the adaptation of the inner layers of the deep neural network, can affect a high criticality one. In addition, the proposed system may need to run in isolation with respect to other plasma control systems components deployed on the same machinery.

As a consequence, on our virtualized MPSoC architecture, the baseline controller could be seen as the critical real-time task to be offloaded to the RPU, while the outer control loops can be deployed on two temporally separated VMs, running on APUs. Our architecture also enables the possibility to run testing and commissioning sessions of new versions of controllers "in parallel" with stable and fully tested ones, on the same hardware and on the same data in real-time. In this way, it is possible to rapidly commission new software versions, while falling back to the stable controller in case of malfunction, so to fully exploit the computing power of the hardware deployed in the tokamak. In this scenario, the new software will run the updated version of the baseline controller on the RPU, concurrently with the previous stable version. The proposed architecture would thus transparently manage the sharing of the RPU on behalf of controllers running in VMs, minimizing interference and without requiring developers to bother about resource sharing.

V. Conclusive Remarks

In this paper we presented an architectural proposal, based on virtualization and processor communication mechanisms on asymmetric multi-processors, to virtualize Real-Time Processing Units in modern Multi-Processor Systems on Chip. The architectural solution enables new usage scenarios of hardware accelerators, while assuring transparency and isolation. Current efforts are being devoted to the implementation of the solution on real machinery. Future work will address the development, deployment, and test of the ITER vertical stabilization controller, as envisioned in the previous section.

References

[1] W. Wolf, A. A. Jerraya and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008, vol. 27, pp. 1701–1713.

[2] Xilinx, Inc., "Enabling Virtualization with Xen Hypervisor on Zynq UltraScale+ MPSoCs (White Paper)," Xilinx, 2016.

[3] M. GarcÌa-Valls, T. Cucinotta, and C. Lu, "Challenges in Real-time Virtualization and Predictable Cloud Computing," *Journal of Systems Architecture* 60, no. 9, pp.726-740, 2014.

[4] M. Cinque, R. Della Corte, A. Eliso, A. Pecchia, "RT-CASEs: container-based virtualization for temporally separated mixed-criticality task sets," 10.4230/LIPIcs.ECRTS.2019.5.

[5] S. Xi, M. Xu, C. Lu, L. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in Xen, " in *Proceedings of the IEEE EMSOFT confernece*, 2014.

[6] W. Wang, M. Bolic, and J. Parri, "pvFPGA: accessing an FPGA-based hardware accelerator in a paravirtualized environment," in *Proc. IEEE CODES+ ISSS*, 2013.

[7] Siemens AG, "Jailhouse hypervisor source code", https://github.com/siemens/jailhouse, accessed on 10 June 2021.

[8] github.com,https://github.com/OpenAMP/open-amp, accessed on 5 May 2021

[9] F. Baum and A. Raghuraman, "Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs," in *Proc. of the 8th European Congress on Embedded Real Time Software and Systems*, Jan. 2016.

[10] G. De Tommasi, "Plasma magnetic control in tokamak devices," *Journal of Fusion Energy*, vol. 38, no. 3-4, pp. 406-436, 2019.

[11] N. Bao et al., "A Real-Time Disruption Prediction Tool for VDE on EAST," *IEEE Transactions on Plasma Science*, vol. 48, no. 3, pp. 715–720, 2020.

[12] "European Research Roadmap to the Realisation of Fusion Energy," https://www.euro-fusion.org/fileadmin/user_upload/EUROfusion/Documents/2018_Research_roadmap_long_version_01.pdf, 2018 (accessed on 6 June 2021).

[13] "EAST hit new mark in quest for fusion reactor," http://english.ipp.cas.cn/news/202106/t20210604_271631.html, accessed on 6 June 2021.

[14] J. Wesson and D. Campbell, *Tokamaks*. Oxford University Press, 2011.

[15] "ITER website," https://www.iter.org/, accessed on 6 June 2021.

[16] M. Cinque *et al.*, "Management of the ITER PCS Design Using a System-Engineering Approach," *IEEE Transactions Plasma Science*, vol. 48, no. 6, pp. 1768–1778, 2020.

[17] G. De Tommasi *et al.*, "Real-Time Systems in Tokamak Devices. A Case Study: The JET Tokamak," *IEEE Transactions on Nuclear Science*, vol. 58, no. 4, pp. 1420–1426, 2011.

[18] G. De Tommasi, S. Dubbioso, A. Mele, A. Pironti, "Stabilizing elongated plasmas using extremum seeking: the ITER tokamak case study," in *Proc. of the 29th Mediterranean Conference on Control and Automation*, Bari, Italy, Jun. 2021.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[20] R. Albanese *et al.*, "ITER-like vertical stabilization system for the EAST Tokamak," *Nuclear Fusion*, vol. 57, no. 8, p. 086039, 2017.

[21] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. of the 58th IEEE Conference on Decision and Control*, pp. 4601–4608, Nice, France, Dec. 2019.