IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 November 2025, accepted 20 November 2025, date of publication 26 November 2025,
date of current version 4 December 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3637300

RESEARCH ARTICLE

HyperFPGA: A Scalable Reconfigurable Platform for Computing Paradigms Research

WERNER FLORIAN SAMAYOA[1,2], MARIA LIZ CRESPO[1], SERGIO CARRATO[2], AGUSTIN SILVA[3], MAYNOR GIOVANNI BALLINA ESCOBAR[1,2], AND ANDRES CICUTTIN[1]

[1]Multidisciplinary Laboratory (MLab), The Abdus Salam International Centre for Theoretical Physics, 34151 Trieste, Italy
[2]Dipartimento di Ingegneria e Architettura (DIA), Universitá degli Studi di Trieste, 34127 Trieste, Italy
[3]Instituto de Investigaciones CientÍficas y Tecnológicas, Mar del Plata B7608FDQ, Argentina

Corresponding author: Werner Florian Samayoa (werneroswaldo.floriansamayoa@phd.units.it)

This work has been partially supported by the STEP and TRIL programmes of the International Centre for Theoretical Physics (ICTP), which are gratefully acknowledged.

ABSTRACT In recent years, the limitations of von Neumann-based supercomputers in terms of energy efficiency and performance scalability have stressed the need for alternative computing paradigms. This paper introduces the HyperFPGA, an open SoC-FPGA cluster designed for experimental research on novel supercomputing architectures and paradigms. Unlike existing platforms, the HyperFPGA offers flexibility across a wide range of configurations, from electrical standards for data transmission to distributed computing models. Additionally, it is equipped with multiple power consumption monitors to independently measure different computational and hardware aspects, separating data movement from computation. The paper details the design and implementation of the HyperFPGA and demonstrates its basic functionality and scalability through a custom computational solution for the non-attacking n-queens problem, a well-known distributed computing problem. These preliminary tests demonstrate the ability of the HyperFPGA to efficiently handle computationally intensive and distributed tasks, showcasing its potential for fine-grained reconfigurable supercomputing solutions addressing a broad spectrum of scientific and engineering problems. The platform is available for open collaborative projects focused on energy-efficient supercomputing and computational physics.

INDEX TERMS FPGA, SoC, heterogeneous computing, supercomputing, reconfigurable computing.

I. INTRODUCTION

Current supercomputing platforms show a decrease in improvement rates [1], [2], in contrast to the projections of Moore's law and Dennard scaling. This slowdown could be attributed to several factors. First, the physical limits of transistor downscaling have nearly been reached. Second, growing complexity and operational frequency lead to higher power consumption and an increase in dark silicon within the chips. In addition, branch prediction and out-of-order execution mechanisms often cause inefficient memory copy operations, resulting in cache misses over half of the time, having a negative impact on both performance and power consumption [3], [4].

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

Several strategies have been tested over the years to overcome these issues. One of them is processing-in-memory [5], [6]. It involves bringing the processing circuits and memory as close as possible to minimize power consumption and latency. This idea can be traced back to 1989 with the programmable active memory (PAM) on Perle-0 [7], the first multiple field-programmable gate array (FPGAs) research platform to study alternative computing architectures. This kickstarted the research on scalable custom computers, leading to platforms such as Janus [8] and ARUZ [9], among many others [10].

The reconfigurable fabric of FPGAs continues to enable researchers to experiment with new computing architectures. Additionally, the increase in memory resources, clock speed, and flexibility of modern FPGAs makes them attractive substitutes for central processing units (CPUs) and graphics

processing units (GPUs) for some applications. Furthermore, modern devices such as system-on-chip FPGA (SoC-FPGAs) are of particular interest given their great potential with respect to heterogeneous microarchitectures [10].

The diversity of approaches complicates the comparison of architectures due to hardware-specific factors, such as communication protocols, distributed memory systems, interconnection topologies, and interoperability. These aspects are crucial in identifying the most efficient and adaptable architectures for future reconfigurable supercomputing, driving the development of more versatile experimental clusters.

Vendor dependence, which originates at the hardware level due to vendor-specific silicon, often extends to the software level in the form of closed-source tools. This issue affects not only FPGAs but also graphics processing units, both of which heavily rely on proprietary tools, forcing researchers to resort to workarounds to bypass framework restrictions [11].

It is also important to note that, as demonstrated over the years, custom networks and protocols remain highly relevant for reconfigurable computing applications [9], [12], [13], [14], [15], [16], [17] that usually require ad hoc hardware, limiting research capabilities.

To overcome these challenges, multiple systems have sought to provide researchers with the ability to explore computing and network architectures. NetFPGA was introduced in 2007 [18], [19] to explore the opportunities that FPGAs provide as smart network interface cards (NICs). Inspired by NetFPGA, Enzian [20] extended this approach by presenting an open, heterogeneous, scalable node with ample hardware design space, facilitating comparative studies across different platforms.

Motivated by these limitations, in this work we introduce the HyperFPGA, a scalable, reconfigurable FPGA cluster aimed at advanced research in heterogeneous computing. The platform emphasizes openness, modularity, and high scalability, incorporating customizable interconnects and real-time power monitoring, enabling detailed energy analysis at the node and system levels.

The main contributions of this paper are:

- The design and implementation of the HyperFPGA cluster architecture, a scalable, modular, and heterogeneous platform that supports fine-grained reconfigurable computing.
- A customizable interconnection network based on high-speed I/Os enables experimentation with communication protocols across electrical and logical layers, as well as domain-based monitoring of power consumption for independent insights into communication and computation power usage, which is crucial for optimizing energy consumption in distributed applications.
- The introduction of an open-source development framework composed of vendor-independent tools, drivers, physical interconnects, and an orchestration layer, aimed at fostering a community of researchers and developers.
- The validation of HyperFPGA's functionality and capabilities is demonstrated through a simple example that illustrates domain-specific power monitoring tools, data distribution mechanisms, and computing orchestration. The example also showcases how data distribution enables parallel problem-solving and how the platform manages the configuration of all FPGAs and controls the data flow.

The remainder of this paper is organized as follows. Related works are discussed in detail in Section II. Section III presents the design decisions from a conceptual perspective, laying out the main characteristics that make the HyperFPGA unique. In Section IV, we describe how the concepts were implemented, leading to a detailed discussion of the hardware and software of the HyperFPGA in Subsections IV-A and IV-B, respectively. Section V presents an implementation of the N-Queens problem, which serves as a demonstration of the functionalities and scalability of the HyperFPGA. A discussion of future work and possible improvements to the system is presented in Section VI. Finally, in Section VII, we present the main conclusions of this work.

II. RELATED WORKS

Since the 1980s, several FPGA clusters have been developed to explore scalable and reconfigurable architectures for research and practical applications.

Early efforts, such as the Perle-0 [7], focused on demonstrating the computing potential of scalable reconfigurable computing using systolic arrays. Alternatively, the RAWFabric [21], consisting of 64 FPGAs soldered directly onto a large printed circuit board (PCB), focused on maximizing performance for CPU emulation, with no particular interest in further scalability.

Subsequently, with the creation of FAST [22], PLATO [23], BEE [12], [24], and Maxwell [25], scalability became an important factor. The reason for this was the desire to address larger and more complex problems. At this point, most of the clusters were specialized for a given application field.

In 2012, Formic [26] demonstrated that cost-effective clusters could be constructed using homogeneous boards. In the same year, with the rise of commercial FPGA boards, multiple clusters such as Bluehive [27] and Superdragon [28], among others [11], [29], [30], [31], were built, each with its own new set of tools, such as the Phoenix framework [32] and the Janus toolbox [33]. Since the creation of the *RawFabric* compiler [21], multiple tools have been developed to accommodate new programming models, cluster implementations, and targeted applications [34], [35], [36], [37]. Similarly, other high-level synthesis frameworks have been developed using C / C++ [38], Java [39], and MATLAB [40], extending to deep learning frameworks such as hls4ml [41], [42], [43], [44], [45].

All of these works made valuable contributions in the areas of development tools, architectures, and programming models; however, the network aspect remained mostly unchanged

due to the decision to use standard solutions, such as gigabit Ethernet or Infiniband. In contrast, other clusters focused on specific applications, such as RAPTOR [46], BiCoSS [16], and ARUZ [9], opting to implement custom networks at the hardware level. This allowed for better performance while raising numerous questions with respect to network customization. Some of these were addressed in [47], where the performance of indirect and direct interconnections was compared, demonstrating that the performance was closely related to the size of the packet and, ultimately, the intended application. These findings add to the relevance of custom topologies and network stacks, highlighting the need for experimental platforms that enable joint research on networks and architectures.

More recently, Enzian [20] was introduced as an open research platform for hybrid CPU/FPGA systems. It was designed to provide researchers with a comprehensive tool for investigating a wide range of hardware design possibilities by taking advantage of the unique hardware that binds a server-grade CPU to an FPGA through a CPU memory coherency port. The platform provides instruments to monitor power consumption and enables research that is not possible with existing commercial hybrid systems.

The HyperFPGA builds on its predecessors by maintaining an open-source approach in a hybrid system, extending the research space to electrical standards, and offering a common platform for the validation of novel parallel computational paradigms and network infrastructure. This is possible by exposing not only gigabit transceivers but also I/Os for node interconnection, allowing for custom protocols up to the electrical level. Additionally, the HyperFPGA provides a vendor-independent approach that enables customizing node resources without the need for major interventions by using SoMs, something that is not present on any other platform of this type.

III. HyperFPGA DESIGN PHILOSOPHY

The design concept of the HyperFPGA closely aligns with the guidelines discussed in [48] and has been inspired by previous research [49] as well as multiple platforms, including Enzian [20], ESSPER [50], and BEE [51], as previously mentioned. The development of the HyperFPGA was guided by principles aimed at maximizing the experimental space and system's reconfigurability. Its fundamental characteristics are described as follows:

- **Modular, open, and scalable hardware:** These features enable users to customize and adapt the system to their specific requirements, which is crucial for exploring different architectures, implementing diverse applications, and accommodating evolving hardware needs. In addition, modularity eases maintenance by isolating issues to specific components and promotes the reuse of existing hardware designs and modules.
- **Flexible network:** In the same way that hardware is optimized for a given purpose, interconnection architectures significantly impact distributed systems performance. The HyperFPGA strives to be a research platform; for this reason, it is not designed around any specific protocol, but rather exploits the multiple I/Os provided by FPGAs to enable experimenting with networks at different levels, from electrical standards to logical protocols.
- **Open, programmable, and modular development framework:** This framework allows users to tailor tools and workflows according to their experimental needs, which is essential for exploring various computing strategies and paradigms. A programmable development environment supports the integration of hardware and software components through standardized and open interfaces, enabling seamless interaction between the FPGA-based hardware and the software stack. All components must be built according to the open-source philosophy to favor a collaborative environment.
- **Instrumentation:** The HyperFPGA provides the instruments to obtain detailed information on the power consumption of the system by separating the power domains of the platform. This makes it possible to measure the consumption of the CPU, FPGA, and their interconnections independently. This allows independent measurement of the CPU, FPGA, and their interconnections, aiding researchers in developing precise design exploration tools and optimizing implementations based on power consumption for computation and communication separately.
- **Research capabilities overall:** All design decisions aim to maximize the platform's research potential and accessibility by prioritizing flexibility, openness, and ease of use. Although HyperFPGA may not excel in performance for specific specialized tasks compared to dedicated solutions, it offers unparalleled versatility. This enables experimental exploration across a wide spectrum of areas, including novel computational models and paradigms, energy consumption profiling, and the development of new languages, design strategies, and automation tools tailored for fine-grained reconfigurable supercomputing.

IV. IMPLEMENTATION DETAILS

We introduce a scalable platform based on identically shaped nodes that are interconnected to form a 2D mesh (thoroughly studied in [52] and [53]), effectively mimicking the routing interconnection pattern of an FPGA; hence the name HyperFPGA. Additionally, the interconnection can be modified to form a 2D torus [48] by using cables and adapters to connect the opposite sides of the cluster. Figure 1 shows the cluster in its current state, consisting of 16 nodes.

A. HARDWARE

Each HyperFPGA node is made up of a carrier board, as shown in Figure 2, which is a square PCB that measures 15×15 cm^2 and a SoM, based on SoC-FPGA. On each

FIGURE 1. HyperFPGA cluster in its current state, consisting of 16 nodes, six Ultrascale+ MPSoC ZU4EG and ten Ultrascale+ MPSoC ZU3EG.

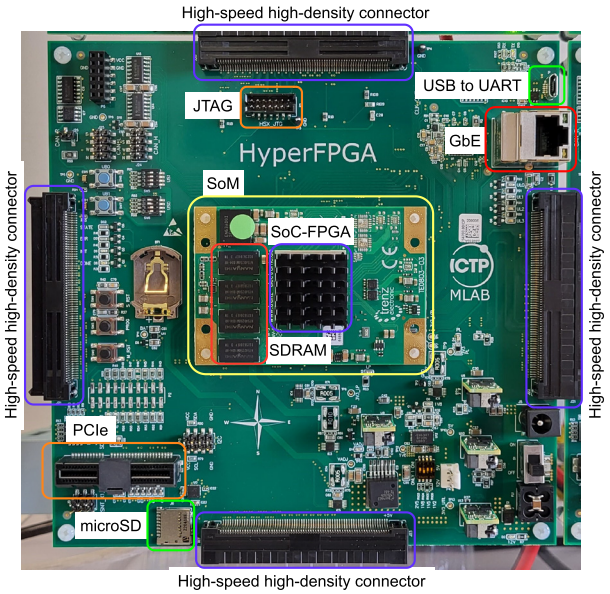


FIGURE 2. HyperFPGA carrier with a mounted System-on-Module, main components and interfaces are highlighted and labeled.

side, there is a Hirose FX18 high-speed connector [54] with 140 pins, which provides a physical interconnection between adjacent carrier boards. For debugging, the JTAG and UART SoM ports are available through a dedicated connector and micro USB, respectively. A gigabit Ethernet port allows integrating the system in commodity networks. Four Samtec Searay 160-pin connectors [55] at the center of the board provide interface signals with the SoM. A peripheral component interconnect express (PCIe) x4 port is also available to increase the CPU capabilities. Finally, a microSD card slot allows persistent storage.

Each SoM [56] features an AMD Ultrascale+ SoC-FPGA, combining a quad-core A53 ARM processor, a dual-core R5 ARM real-time processor, and an Ultrascale+ FPGA. This open SoM format is supported by at least two vendors (Trenz [56] and Sundance [57]), offering commercial devices equipped with various SoC-FPGAs from the AMD Ultrascale+ and Microchip PolarFire FPGA [58] families.

1) POWER AND MONITORING

Each carrier board has a barrel plug that connects to a 12 V external power supply. The main power is filtered and limited through a resettable fuse at 15 A. The 12 V feeds five onboard power converters that supply each of the power domains present in the system. Additionally, the 12 volts provide power directly to the PCIe connector.

The following SoM power domains are individually monitored through I2C using the INA228 [59] current and power monitoring chip:

- Low power peripherals and ARM R5 CPU
- Full power, consisting of GTHs and ARM A53 CPU
- FPGA programmable logic
- Adjustable voltage for FPGA input/output buffers

By dividing the power domains for computing and data communication, it is possible to gather detailed information about the power consumption of specific computational activities during execution. One of the primary advantages is the capability to measure power usage in real-time, offering valuable insights for profiling and optimizing computational solutions to enhance energy efficiency.

2) CARRIER BOARD INTERCONNECTION

The carrier board uses four high-speed connectors that provide 28 pairs of low-voltage differential signals with varying maximum speeds. With these signals, a variety of possible electrical standards can be implemented.

Up to 4 GTH lanes are available per connector, providing a point-to-point connection of up to 16.3 Gbps. Given that these rely on proprietary and vendor-specific hard IP cores, porting implementations becomes a complicated task. To alleviate this, solutions such as the Kyokko IP [60] offer an open-source wrapper for transceivers, resulting in a portable and interoperable interface between Intel and AMD FPGAs.

Figure 3 shows the three HyperFPGA networks. ARM processors are connected through a switched gigabit Ethernet network available for user interaction. The four board-to-board connectors and the PCB provide the infrastructure for a CAN network to interconnect all processors. At the same time, these connectors allow one to take advantage of the large number of GPIOs present in the FPGAs; 18 high-performance (HP) and 6 high-density (HD) pairs are available on each connector, along with the GTHs. Each HP and HD pair provides additional throughput of 1.2 Gbps and 500 Mbps at double data rate (DDR), respectively. The maximum total throughput for each FPGA is 388 Gbps, depending on the available SoM resources.

B. SOFTWARE

Software is crucial for interacting, managing, monitoring, and developing applications for the cluster. Due to the unique characteristics of each reconfigurable supercomputing system, custom tools are often necessary. This was the case with RAPTOR [46], which consisted of a complete software stack, including a graphical user interface. However, similar solutions tend to restrict users to paradigms related to the architecture or target application. To avoid this, we focus on providing the minimum interfaces to guarantee the safe operation of the cluster, such as the CPU-FPGA communication channel and its high-level abstraction. Furthermore, the programming paradigm remains completely at the discretion of the user, as the base services provide configuration, monitoring, and safety measures. This fully programmable framework enables the HyperFPGA to function as a versatile and generic testbed for heterogeneous computing applications.

The HyperFPGA software stack, shown in Figure 4, is open and modular, allowing users to adapt it according to their experiments. It differs from other similar frameworks, such as Octoray [61], by relying on custom tools in the hardware definition, boot, and hardware abstraction layers to provide open and standard interfaces that allow the integration of existing tools into the upper layers to maximize vendor independence and guarantee safety. From the ground up, the stack begins with hardware definition tools and facilities that configure the peripherals on the carrier board and SoM for their use. This layer is also accessible to users to implement their application-specific digital circuits. On top of this, the boot tools implement the required security checks and hardware configurations for an operating system (OS) to run on the ARM CPUs. To interface the OS and user space with the hardware, the hardware abstraction layer offers a custom script that generates the required files to identify the user-created hardware in the FPGA, and a custom communication block (ComBlock) [62] with its Linux driver, which provides a safe interface with the FPGA.

A management layer consisting of *JupyterHub* facilitates all administrative tasks. And at the top of the stack, the programming context consists of *JupyterLab* along with *IPython Parallel*, which provides the tools for CPU cluster management. The entire stack depends on open tools wherever possible.

1) BOOT

The boot process is vital for ensuring a reliable system startup and proper configuration. The HyperFPGA, in line with its philosophy, utilizes a custom Linux-based OS developed with Petalinux. Due to the potential file size exceeding the available flash memory, the microSD card port on the carrier is used to host boot files, the file system, and user space, providing ample storage and cost-effective replaceability.

Figure 5 shows the HyperFPGA boot process, which begins with the Platform Management Unit (PMU) and Configuration Security Unit (CSU) processors booting with default binaries. If successful, the CSU configures the ARM R5 with the First Stage Boot Loader (FSBL) from the on-chip memory (OCM), ensuring that the GTHs on the SoM have a valid clock. Afterward, the ARM Trusted Firmware (ATF) runs on ARM A53 processors, enabling OS deployment. U-boot then loads the custom Linux kernel, and the OS (Debian 11) boots with network access. The FPGA retains the boot configuration until a user applies a bitstream to the Configuration RAM (CRAM).

The boot sequence provides flexibility, with options such as hosting a minimal boot on SoC-FPGA flash and using the SD card as a backup. Alternatively, a file transfer protocol (FTP) server could host the OS for added resilience. Although successful on other platforms [63], [64], the controlled and accessible HyperFPGA environment prioritizes completion of board initialization before remote interaction, delaying control transfer to the user until then.

2) HARDWARE ABSTRACTION

Any OS provides interfaces to interact with hardware; Linux is not the exception. In this regard, the device tree supports the compatibility and portability sought by the HyperFPGA, offering dynamic hardware descriptions for Linux-based operating systems. It enables interaction without hard-coding hardware platform details by using devicetree overlays.

To take advantage of this tool, a custom Python script *XSA2Bit* was developed to translate the Vivado XSA files into devicetree overlays. This facilitates transparent OS updates with new FPGA functionalities, minimizing user intervention. The script also automatically instantiates ComBlock devices, enhancing the interface between the FPGA and CPU. The ComBlock abstraction simplifies the use of SoC-FPGA buses, promoting portability and separation of CPU and FPGA programming activities. The use of kernel drivers ensures memory protection and enables driver development with common file operations. Detailed advantages of the ComBlock utilization are discussed in [65].

3) MANAGEMENT

This is the first point of interaction between the user and the HyperFPGA. It consists of authentication, authorization, and accounting services. To simplify the development of the platform, we decided to rely on *JupyterHub*. Given that all nodes already run Linux on ARM cores, the implementation was greatly simplified.

JupyterHub provides several authentication methods, and its open-source nature simplifies customization. Currently, the system is deployed on a local network using the *native Linux authenticator*. Users can request to be registered, and the administrator can enable the account without storing sensitive information. Once logged in, *JupyterHub* relies on spawners to redirect users to their respective *JupyterLab* sessions. By modifying the behavior of the spawner, it is possible to run different configurations to customize user interaction within the system.

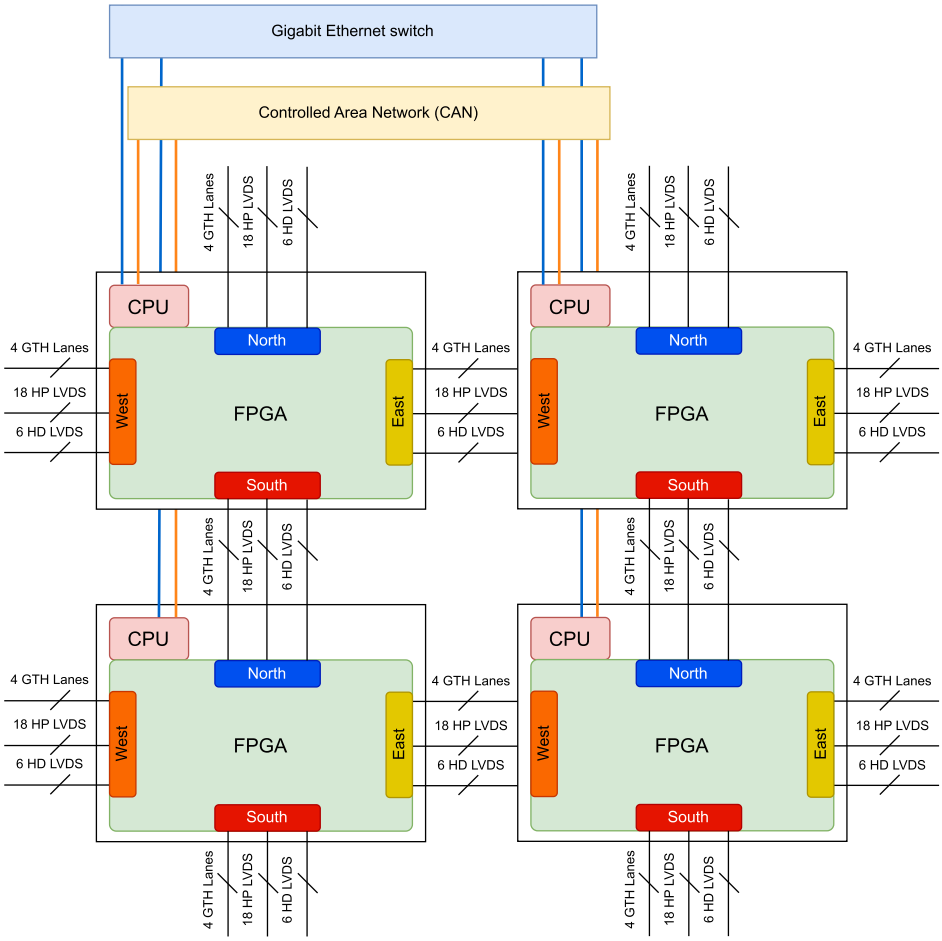


FIGURE 3. Schematic view of four HyperFPGA carriers hosting SoC-FPGAs interconnected. The CPUs interconnect through a Gigabit Ethernet switch and a controller area network (CAN), while the FPGAs are directly attached through their I/Os.

Context	Tools
Programming	JupyterLab, IPython Parallel
Management	JupyterHub
Hardware abstraction	XSA2Bit, ComBlock driver
Boot	Yocto (Petalinux), U-boot
Hardware definition	Vivado, Application-agnostic BSP

FIGURE 4. HyperFPGA software stack showing the tools and their application context.

Figure 6 shows the diagram that corresponds to this cluster implementation. It shows the hub as the only gateway for interaction with the cluster, but also as a centralized file system for the user's configuration files, experiment results, and applications. Management tasks are primarily limited to this part of the system and consist of enabling user accounts, upgrading system-wide libraries, securing data storage, and assigning nodes to users, among other responsibilities. For this implementation, the spawner starts a *JupyterLab* session on the hub server, and communication is forwarded to a given cluster of nodes through Ethernet.

JupyterHub flexibility has also proven to be beneficial for user interaction on individual boards. By implementing the *SSHSpawner* [66], users are further restricted to a specific node of the cluster, which can be useful for teaching activities [67]. The *SSHSpawner* takes advantage of the network by initiating a remote *JupyterLab* session on a node and then forwarding it to the hub server, as shown in Figure 7. In both cases, the gateway to the hardware is a common server that can easily implement all safety measures that require minimal programming skills.

4) PROGRAMMING

Programming, in general, involves two main stages: the configuration and implementation of algorithms. During configuration, the hardware architecture, initial conditions, and algorithm parameters are set, while implementation translates the logic into an activity. Although the configuration may be relatively straightforward in classical computing, this

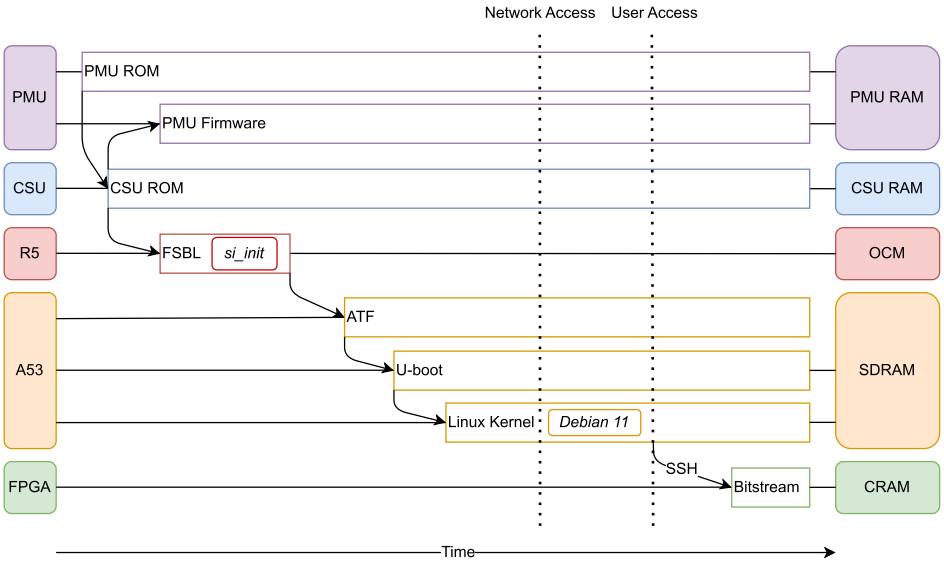


FIGURE 5. HyperFPGA boot sequence showing hand-off points.

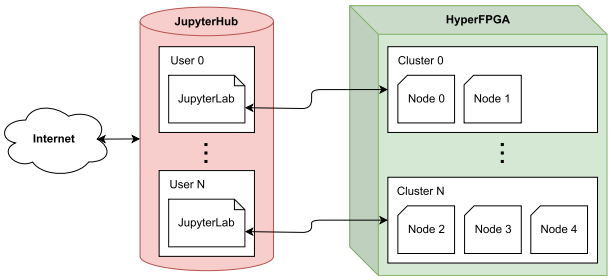


FIGURE 6. User interaction block diagram with *JupyterHub* centralized management and remote clusters instantiated using HyperFPGA nodes; spawner initiates local *JupyterLab* session on the hub where notebooks can be created, modified, and executed.

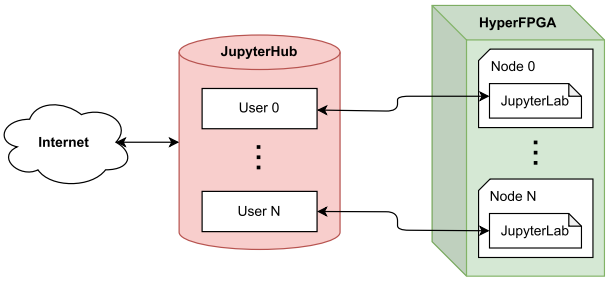


FIGURE 7. Block diagram of the infrastructure configured for user interaction with individual nodes through *SSHSpawner*. Users are authenticated in the hub and the spawner initiates a remote session that is forwarded to the hub for user interaction.

is not the case for reconfigurable platforms, where the hardware architecture may be very different for each problem and even between implementations. At this point, only a flexible framework can adapt accordingly to allow users to implement their applications without having to make major modifications to the underlying layers.

Figure 8 shows the current workflow and tools required to develop and implement an application for the HyperFPGA cluster. To begin with, HDL code can be automatically generated or written directly in VHDL or Verilog, which are the only two languages supported for hardware synthesis in the case of Vivado [68]. The SoC-FPGA implementation methodology follows the principles described in [69], where the ComBlock [62] is used as a bridge between the reconfigurable logic of the user and the CPU. By abstracting the interface details, the ComBlock provides a compatibility layer consisting of registers, FIFOs, and dual-port RAM that facilitates the portability of the design.

The HyperFPGA platform configuration is provided in a custom board support package (BSP) format, compatible with Vivado. Once the user design has been integrated and synthesized for the SoC-FPGA, it can be exported to be processed by our custom Python script: *XSA2Bit*. This script compiles the bitstream and generates the corresponding device-tree overlay. Once the project files are prepared, users can take advantage of a *Jupyter* environment, including *JupyterHub* [70] for management and *JupyterLab* for application development.

In the described cluster scheme using *JupyterHub*, users have the option to interact with the system through a shell terminal or *JupyterLab* notebooks. The user's home folder contains a `nodes.json` file detailing cluster node information, crucial for configuring and monitoring computations in FPGAs.

Two folders are created within the user account: *bitstreams* for FPGA configuration and device tree overlays targeting specific FPGAs, and *comblock_driver* for Linux kernel drivers that facilitate interaction with the ARM A53 quad-core and the FPGA via the ComBlock. The drivers have normal and debugging variants, allowing users to inspect the

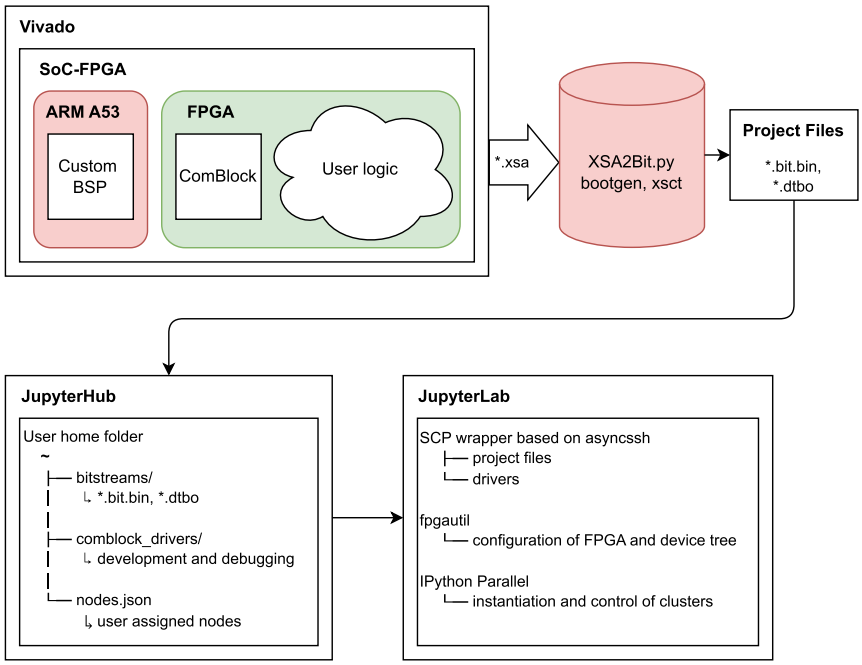


FIGURE 8. HyperFPGA workflow for application development.

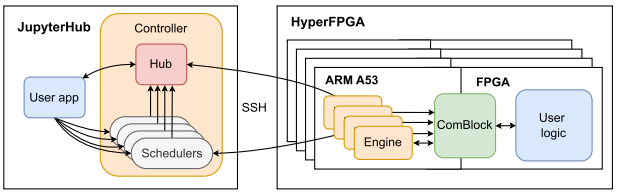


FIGURE 9. The HyperFPGA application framework implementation consists of the *IPython Parallel* controller, hub, schedulers and engines that integrate with the ComBlock.

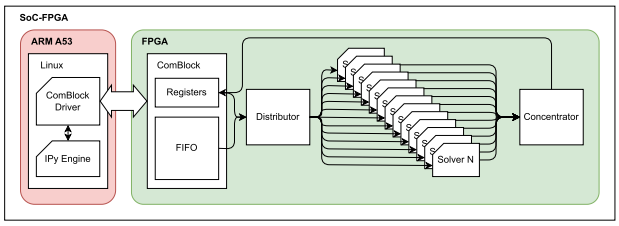


FIGURE 10. N-Queens problem block diagram of a single node showing the solvers, partial solutions distributor, partial results concentrator, and ComBlock.

kernel-ComBlock interactions. Once a device tree overlay is applied, the kernel driver generates device files for user-space interaction with the ComBlocks in the FPGA.

The application framework is based on *IPython Parallel* [71], which follows a message-passing paradigm. It provides functions that facilitate the instantiation, interaction, and management of CPU clusters. The message passing paradigm is implemented via functions to share Python objects and perform computations, such as *broadcast*, *barrier*, and *map*. These functions, combined with the ComBlock drivers, allow for a heterogeneous computing flow in which the ARM and FPGAs can effectively cooperate.

Figure 9 shows the *IPython Parallel* architecture, consisting of a controller that acts as the interface between the user application on JupyterHub and the engines running on the HyperFPGA nodes. The controller contains a hub that manages connections, schedulers, task requests, and results. The schedulers provide a fully asynchronous interface with the engines. At the node level, the engines interface with the user logic through the ComBlock module and the Linux driver.

C. COMPARATIVE SUMMARY OF PLATFORM CAPABILITIES

Table 1 summarizes the main features of the HyperFPGA in comparison with representative reconfigurable computing platforms. The comparison emphasizes key aspects such as architecture, scalability, interconnect design, power monitoring, and openness, providing a concise view of HyperFPGA's unique advantages and research potential [72].

Unlike integrated CPU–FPGA systems such as Enzian, ARUZ, ESSPER, and AWS F2, which primarily target system-level workloads, **HyperFPGA** introduces a modular SoC–FPGA architecture that enables distributed and parallel experimentation for computational problems.

Overall, HyperFPGA provides greater flexibility and accessibility for experimental hardware research. Its reconfigurable interconnect allows users to define custom protocols and topologies, with the flexibility to study the scalability, workload sensitivity, and performance trade-offs beyond fixed-architecture clusters.

TABLE 1. Comparative table of similar FPGA platforms.

Feature	HyperFPGA	Enzian [20]	ARUZ [9]	ESSPER [50]	AWS F2 [73]
Hardware resources	SoC-FPGA module	CPU + FPGA	2922 SoC-FPGAs 23040 FPGAs	8 CPUs 32 FPGA	CPU + 8 FPGAs
Scaling (no discontinuities at increasing size)	Yes	Yes	No	No	No
Network	Point-to-point, in-fabric switch	Point-to-point, in-fabric switch	Point-to-point	Point-to-point, in-fabric switch, switched	Switched
Electrical standards flexibility	Yes, LVDS and LVCMOS	No	No	No	No
User access level	IO level	System	Application	Application	Application
Fine-grained power monitoring	4 Power domains	Multiple power domains	No	1 Power domain	None
Open source	Hardware and Software	Software	N/A	Software	Software
Device upgradability	Yes	No	No	Yes	N/A

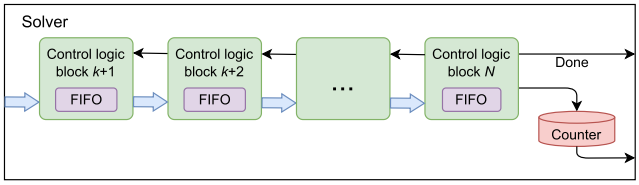


FIGURE 11. N-Queens solver internal block diagram showing $N - k$ control logic blocks with their input FIFOs. The last control logic increments the solutions counter and generates the *Done* flag.

V. TEST APPLICATION: N-QUEENS PROBLEM

The N-Queens problem is a combinatorial problem that has been extensively studied [74]. It consists of determining the total possible placements of N queens on an N by N chessboard for which no queen can attack another. The number of total solutions increases approximately with the factorial size of the chessboard.

Its inherent computational complexity has made the N-Queens problem a classical benchmark in combinatorial optimization. The exploration of its vast solution space can be naturally parallelized, making it particularly suitable for evaluating parallel search algorithms. Moreover, its well-defined constraints and exponential growth of possible configurations make it an effective experimental benchmark for testing tree exploration architectures, pipelined designs, and pruning strategies in reconfigurable hardware, providing an ideal test case for our development framework.

A. IMPLEMENTATION

This particular problem tests the scalability of HPC platforms and has been solved in numerous ways. We use a backtracking algorithm [75] to optimize the implementation and reduce the computational time required to find all solutions. A key benefit of using backtracking is that it follows a depth-first search strategy, which facilitates distributing different branches of the search tree across the cluster nodes for parallel processing.

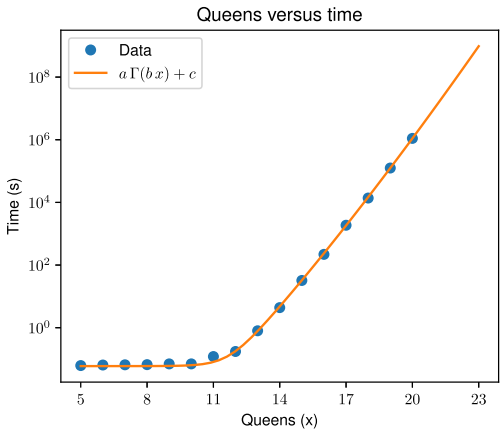


FIGURE 12. This plot shows how the N-Queens problem execution time grows with real values from the experiments using a single solver running at 100 MHz. Additionally, a model using the Gamma function (Γ) was used to fit the data, where $a = 7.595 * 10^{-7}$, $b = 0.802$, $c = 0.0595$.

In this case, we calculated part of the problem at the hub. This consists of finding the valid positions of the queens for the first k columns of the chessboard in the CPU, as this is the least computationally intensive part of the problem. Given that it is impossible to anticipate how long it will take to evaluate each partial solution, we maintain a continuous data distribution to the nodes to maximize efficiency. Inside the nodes, we implement the backtracking algorithm using VHDL templates that are customized by passing parameters such as the size of the board and the number of columns (k) to calculate the partial solutions at the hub.

Figure 10 shows the FPGA block diagram within each node. The logic is managed through read and write operations

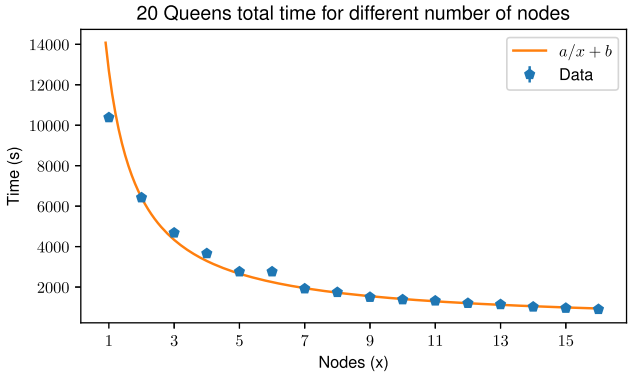


FIGURE 13. N-Queens problem execution time for a board size of $N = 20$ and different numbers of nodes, from 1 to 16. A simple model allows estimating further improvements, where $a = 1.253 * 10^4$, $b = 156.1$.

to the ComBlock resources. The ComBlock FIFO holds the partial solutions assigned to the node, while a distributor reads from this FIFO and sequentially transfers the partial solutions to each solver in a round-robin fashion.

The block diagram of an individual solver can be seen in Figure 11. Inside each solver, the backtracking algorithm is implemented by unrolling the columns of the chessboard into $N - k$ logic control blocks, each with its own input FIFO to minimize dead time. Each logic control block takes a partial solution from the preceding block and checks for valid solutions in its designated column. Any valid solution is concatenated into the partial solution and fed to the next block. When the last block finds a valid solution, the solver increments its solutions counter. Once no more partial solutions are available for processing and the solver has completed its assigned computations, a *done* flag is raised. Finally, when the concentrator detects that all solvers have raised the *done* flag, it accumulates the values of all counters and updates the ComBlock registers.

B. RESULTS

During the algorithm development phase, the implementation of the solver was tested on a single SoC-FPGA development platform for different chessboard sizes. The results obtained give us an idea of how the problem grows. Figure 12 shows how the execution time increases with the number of queens in a factorial way [74]. This plot was drawn using a single solver running at 100 MHz.

As expected, when partial solutions are distributed to more nodes, the total execution time decreases, confirming that the platform allows one to easily exploit the algorithm's parallelization. In this case, the size of the chessboard N was fixed at 20, and up to 16 nodes were used to understand how increasing the number of solvers affected the execution time. Figure 13 shows how the execution time changes when increasing the number of nodes.

Another important factor is the number of solvers that fit into a determined FPGA. For this particular test, two different SoMs were used. Six Ultrascale+ MPSoC ZU4EG fitted 28 solvers at 400 MHz, and ten Ultrascale+ MPSoC ZU3EG with 23 solvers at 300 MHz, which are equivalent to 1362 solvers at 100 MHz.

To illustrate the performance gain, we estimated the speedup of our reconfigurable implementation for $N = 15$, which resulted in more than 10^5 times faster compared to a Python implementation executed on a single thread of an Intel i7 CPU at 3.2 GHz with 32 GB of DDR3 RAM at 2.4 GHz. This improvement is expected to increase significantly with N.

The power consumption profile of the implementation, shown in Figure 14, was obtained by taking samples from the three relevant power domains at different stages of the implementation. The Adjustable Voltage power domain that corresponds to the FPGA I/Os was left out, given that the implementations did not make use of these resources. The defined stages were as follows:

1) Initialization: This stage encompasses the bring-up of the platform up to the point when network access is available.
2) Configuration: Includes the instantiation of the IPyParallel engines and the implementation of the bitstream in the FPGA.
3) Execution: It is the actual computing stage. In this case, it consists of the feed of partial solutions to the solvers in the FPGA and the polling of the done flag.

Once the boot sequence is completed, the system reaches the initialization stage. At this point, the OS runs on the ARM CPUs of the system without user intervention; the FPGA has no configuration loaded, the FPGA I/Os are static, and the consumption of the Low power domain is limited to the peripherals of the OS, given that the real-time processing unit is not being used. It can be appreciated that the CPUs present minimal consumption, as only the essential tasks of the OS are running.

The configuration stage is the initial interaction point. In this stage, the user creates the remote engines on the node and configures the FPGA. These two major changes are reflected in the increase in consumption of the Full power domain and Programmable logic power domain. With the configuration complete, the application can begin execution. It consists of a continuous polling of the solutions counter from the engines, which translates to increased power consumption in the Full power and Low power domains. Although no computational resources are engaged in the Low Power domain at this stage, the ComBlock is accessed via the low-power AXI ports associated with it, accounting for the observed increase in consumption. This demonstrates that the power monitoring system can effectively track energy usage within the SoC-FPGA device, providing valuable insights into internal power dynamics.

VI. FUTURE WORKS

Having established the feasibility of reconfiguring, programming, and interacting with the HyperFPGA cluster, we will

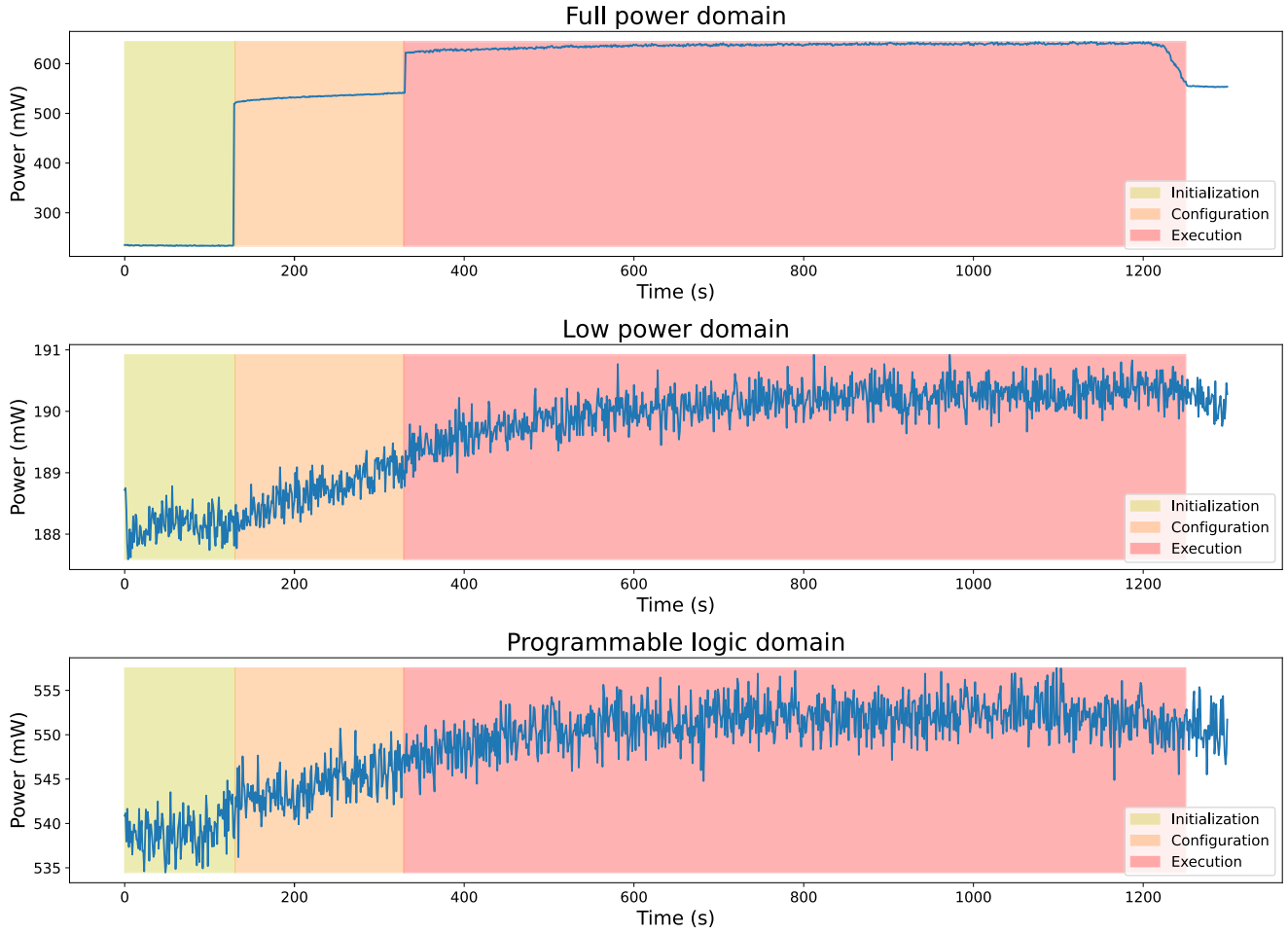


FIGURE 14. Instant power consumption in the relevant power domains of the HyperFPGA for the 20 Queens test. The FPGA I/Os is not shown given that they were not used for this implementation.

proceed to experiment with more complex problems and make the platform available to the scientific community for experimental research and development activities. An aspect that has not been discussed in detail in this paper is the FPGA-specific network. This is one of the most relevant parts of the system, as significant improvements will come from taking advantage of the seemingly scalable FPGA fabric that results from interconnecting the HyperFPGA nodes. To achieve this, we will integrate high-speed protocols to take advantage of the high bandwidth and flexibility of the available I/Os. A generic distributed memory approach, such as the one described in [76], could be assigned to the cluster to begin exploring direct communication between adjacent FPGAs and collaboration among all FPGAs in the cluster.

Another important aspect is determining the best way to describe distributed heterogeneous algorithms. Important contributions have been made showing the potential of high-level synthesis solutions for FPGA clusters [77], [78] that are worth analyzing in depth. At the same time, design space exploration tools have evolved [79], providing a better estimate of the resources needed for a particular circuit and offering fully adaptive solutions for transparent energy optimizations [80]. These could be extended to identify optimal strategies for segmenting a problem and distributing partial tasks among a given number of nodes.

VII. CONCLUSION

In this work, we present the HyperFPGA cluster, a hardware platform and related software utilities for experimentation with heterogeneous and fine-grained reconfigurable computing architectures. We show that it is possible to build a flexible cluster on a reconfigurable and programmable framework that allows for research on diverse computational paradigms and related design and programming strategies. The platform was developed on the principles of openness, modularity, and scalability as prescribed in [48].

All HyperFPGA hardware fabrication files and schematics are openly available for anyone to modify and replicate. In the same spirit, the custom Linux OS, the configurations of *JupyterHub*, the ComBlock Linux drivers, and every piece of software that was developed are provided in the appropriate

repositories. This, along with the documentation, will help the community take advantage of the numerous benefits of the HyperFPGA.

The hardware modularity was achieved by exploiting commercial off-the-shelf SoMs with an open pinout for vendor interoperability.

We demonstrated the feasibility of implementing and scaling fine-grained parallel computing algorithms in the HyperFPGA by using *JupyterLab* and the predefined interconnection layer consisting of the ComBlock and Linux kernel driver. By implementing a backtracking algorithm to solve the N-Queens problem on the 16 nodes, it was demonstrated that one can take advantage of the scalability of the cluster. A model of the speed-up was obtained that shows the potential of the hardware and the efficiency of the development environment.

The HyperFPGA system offers unique research capabilities, driven by its flexibility and detailed dynamic power monitoring features. Its physical layer, enabled by directly interconnected FPGAs, facilitates experimentation and development of communication systems, such as networks and protocols. Furthermore, independent power monitoring discriminates between the computing and communication domains, offering valuable insights for profiling and optimizing the energy consumption of implemented computing architectures. When combined with its programmable framework, these features render HyperFPGA an ideal platform for exploring and validating innovative paradigms in fine-grained reconfigurable supercomputing, heterogeneous clusters, and parallel programming.

ACKNOWLEDGMENT

The authors would like to thanks Bruno Valinoti and Luis Guillermo GarcÍa Ordóñez for sharing their valuable experience in designing, testing, and manufacturing PCBs based on SoMs.

ONLINE RESOURCES

All files used to develop the HyperFPGA platform and software are freely available in the following repositories:

- HyperFPGA source files:
  - https://gitlab.com/ictp-mlab/hyperfpga-linux
  - https://gitlab.com/ictp-mlab/hyperfpga-hw
  - https://gitlab.com/ictp-mlab/xsa2bit
- N-Queens problem implementation files:
  - https://gitlab.com/ictp-mlab/n-queens

REFERENCES

[1] M. Feldman. (2017). *TOP500 Meanderings: Sluggish Performance Growth May Portend Slowing HPC Market*. [Online]. Available: https://www.top500.org/news/top500-meanderings-sluggish-performancegrowth-may-portend-slowing-hpc-market/

[2] J. Koomey, Z. Schmidt, and S. Naffziger. (Dec. 2019). *Supercomputing Performance and Efficiency: An Exploration of Recent History and Near-Term Projections*. [Online]. Available: https://www.amd.com/en/system/files?file=documents/Supercomputing-Performance-Efficiency.pdf

[3] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. Twenty-Third Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2018, pp. 316–331, doi: 10.1145/3173162.3173177. [Online]. Available: https://doi-org.ezproxy.cern.ch/10.1145/3173162

[4] B. Dally. (2015). *Challenges for Future Computing Systems*. [Online]. Available: https://shorturl.at/bipvB

[5] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system," *IEEE Access*, vol. 10, pp. 52565–52608, 2022, doi: 10.1109/ACCESS.2022.3174101.

[6] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, and J. S. Vetter, "A survey on processing-in-memory techniques: Advances and challenges," *Memories-Mater., Devices, Circuits Syst.*, vol. 4, Jul. 2023, Art. no. 100022, doi: 10.1016/j.memori.2022.100022.

[7] P. Bertin, D. Roncin, and J. Vuillemin. (Jun. 1989). *Introduction To Programmable Active Memories*. Paris Research Laboratory. [Online]. Available: https://www.hpl.hp.com/techreports/Compaq-DEC/PRL-RR-3.pdf

[8] F. Belletti et al., "Janus: An FPGA-based system for high-performance scientific computing," *Comput. Sci. Eng.*, vol. 11, no. 1, pp. 48–58, Jan. 2009, doi: 10.1109/MCSE.2009.11. [Online]. Available: https://ieeexplore.ieee.org/document/4720223/

[9] R. Kiełbik, K. Hałagan, W. Zatorski, J. Jung, J. Ulański, A. Napieralski, K. Rudnicki, P. Amrozik, G. Jabłoński, D. Stożek, P. Polanowski, Z. Mudza, J. Kupis, and P. Panek, "ARUZ— Large-scale, massively parallel FPGA-based analyzer of real complex systems," *Comput. Phys. Commun.*, vol. 232, pp. 22–34, Nov. 2018, doi: 10.1016/j.cpc.2018.06.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465518302182

[10] Y. K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "In-depth analysis on microarchitectures of modern heterogeneous CPU-FPGA platforms," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–20, Feb. 2019, doi: 10.1145/3294054. [Online]. Available: https://api.semanticscholar.org/CorpusID:67871716

[11] T. Boku, N. Fujita, R. Kobayashi, and O. Tatebe, "Cygnus–World first multihybrid accelerated cluster with GPU and FPGA coupling," in *Proc. 51st Int. Conf. Parallel Process.*, Aug. 2022, pp. 1–8, doi: 10.1145/3547276.3548629.

[12] K. Kuusilinna, C. Chang, M. J. Ammer, B. C. Richards, and R. W. Brodersen, "Designing BEE: A hardware emulation engine for signal processing in low-power wireless applications," *EURASIP J. Adv. Signal Process.*, vol. 2003, no. 6, pp. 502–513, May 2003, doi: 10.1155/s1110865703212154.

[13] A. T. Markettos, P. J. Fox, S. W. Moore, and A. W. Moore, "Interconnect for commodity FPGA clusters: Standardized or customized?" in *Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2014, pp. 1–8, doi: 10.1109/FPL.2014.6927472. [Online]. Available: http://ieeexplore.ieee.org/document/6927472/

[14] A. P. Davison and Human Brain Project Collaboration. (2023). *About the BrainScaleS Hardware? HBP Neuromorphic Computing Platform Guidebook (WIP)*. [Online]. Available: https://electronicvisions.github.io/hbp-sp9-guidebook/PM/pm_hardware_configuration.html

[15] S. Lyberis, G. Kalokerinos, M. Lygerakis, V. Papaefstathiou, I. Mavroidis, M. Katevenis, D. Pnevmatikatos, and D. S. Nikolopoulos, "FPGA prototyping of emerging manycore architectures for parallel programming research using formic boards," *J. Syst. Archit.*, vol. 60, no. 6, pp. 481–493, Jun. 2014, doi: 10.1016/j.sysarc.2014.03.002. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S138376211400054X

[16] S. Yang, J. Wang, X. Hao, H. Li, X. Wei, B. Deng, and K. A. Loparo, "BiCoSS: Toward large-scale cognition brain with multigranular neuromorphic architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2801–2815, Jul. 2022, doi: 10.1109/TNNLS.2020.3045492.

[17] T. T. Nguyen, K. T. Pham, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Effective switchless inter-FPGA memory networks," *J. Parallel Distrib. Comput.*, vol. 179, Sep. 2023, Art. no. 104713, doi: 10.1016/j.jpdc.2023.05.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523000837

[18] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA—An open platform for gigabit-rate network switching and routing," in *Proc. IEEE Int. Conf. Microelectronic Syst. Educ. (MSE)*, Jun. 2007, pp. 160–161, doi: 10.1109/MSE.2007.69.

[19] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014, doi: 10.1109/MM.2014.61.

[20] D. Cock, A. Ramdas, D. Schwyn, M. Giardino, A. Turowski, Z. He, N. Hossle, D. Korolija, M. Licciardello, K. Martsenko, R. Achermann, G. Alonso, and T. Roscoe, "Enzian: An open, general, CPU/FPGA platform for systems software research," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Feb. 2022, pp. 434–451, doi: 10.1145/3503222.3507742.

[21] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: Raw machines," *Computer*, vol. 30, no. 9, pp. 86–93, Sep. 1997, doi: 10.1109/2.612254. [Online]. Available: http://ieeexplore.ieee.org/document/612254/

[22] J. D. Davis, "FAST: A flexible architecture for simulation and testing of multiprocessor and CMP systems," Ph.D. dissertation, Dept. Elect. Eng., Stanford University, Stanford, CA, USA, Dec. 2006.

[23] A. Dollas, D. Pnevmatikatos, N. Aslanides, S. Kavvadias, E. Sotiriades, S. Zogopoulos, K. Papademetriou, N. Chrysos, K. Harteros, E. Antonidakis, and N. Petrakis, "Architecture and application of PLATO, a reconfigurable active network platform," in *Proc. 9th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Mar. 2001, pp. 101–110.

[24] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2 a high-end reconfigurable computing system," *IEEE Design Test Comput.*, vol. 22, no. 2, pp. 114–125, Feb. 2005, doi: 10.1109/MDT.2005.30.

[25] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, "Maxwell—A 64 FPGA supercomputer," in *Proc. 2nd NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Aug. 2007, pp. 287–294, doi: 10.1109/AHS.2007.71. [Online]. Available: http://ieeexplore.ieee.org/document/4291933/

[26] S. Lyberis, G. Kalokerinos, M. Lygerakis, V. Papaefstathiou, D. Tsaliagkos, M. Katevenis, D. Pnevmatikatos, and D. Nikolopoulos, "Formic: Cost-efficient and scalable prototyping of manycore architectures," in *Proc. IEEE 20th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2012, pp. 61–64, doi: 10.1109/FCCM.2012.20.

[27] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive—A field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Proc. IEEE 20th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2012, pp. 133–140, doi: 10.1109/FCCM.2012.32. [Online]. Available: https://ieeexplore.ieee.org/document/6239804/

[28] G. Tan, C. Zhang, W. Wang, and P. Zhang, "SuperDragon," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 4, pp. 1–22, Oct. 2015, doi: 10.1145/2740966.

[29] AMD-Xilinx. (2021). *Xilinx Adaptive Compute Clusters (XACC) Academia-Industry Research Ecosystem | HACC Resources*. [Online]. Available: https://www.amd-haccs.io/adapt_2021.html

[30] T. Prickett. (2018). *Forging a Hybrid CPU-FPGA Supercomputer*. [Online]. Available: https://www.nextplatform.com/2018/09/25/forging-a-hybrid-cpu-fpga-supercomputer/

[31] A. D. Ioannou, K. Georgopoulos, P. Malakonakis, D. N. Pnevmatikatos, V. D. Papaefstathiou, I. Papaefstathiou, and I. Mavroidis, "UNILOGIC: A novel architecture for highly parallel reconfigurable systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 4, pp. 1–32, Sep. 2020, doi: 10.1145/3409115.

[32] A. Pant, H. Jafri, and V. Kindratenko, "Phoenix: A runtime environment for high performance computing on chip multiprocessors," in *Proc. 17th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2009, pp. 119–126, doi: 10.1109/pdp.2009.41.

[33] M. Baity-Jesi et al., "The Janus project: Boosting spin-glass simulations using FPGAs," *IFAC Proc. Volumes*, vol. 46, no. 28, pp. 227–232, 2013, doi: 10.3182/20130925-3-cz-3023.00039.

[34] G. Chirkov and D. Wentzlaff, "SMAPPIC: Scalable multi-FPGA architecture prototype platform in the cloud," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Jan. 2023, pp. 733–746, doi: 10.1145/3575693.3575753.

[35] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "ZRLMPI: A unified programming model for reconfigurable heterogeneous computing clusters," in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2020, p. 220, doi: 10.1109/FCCM48280.2020.00051.

[36] L. Kalms and D. Göhringer, "Scalable clustering and mapping algorithm for application distribution on heterogeneous and irregular FPGA clusters," *J. Parallel Distrib. Comput.*, vol. 133, pp. 367–376, Nov. 2019, doi: 10.1016/j.jpdc.2018.02.033. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518301199

[37] J. M. de Haro, R. Cano, C. Álvarez, D. Jiménez-González, X. Martorell, E. Ayguadé, J. Labarta, F. Abel, B. Ringlein, and B. Weiss, "OmpSs@cloudFPGA: An FPGA task-based programming model with message passing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2022, pp. 828–838, doi: 10.1109/IPDPS53621.2022.00085.

[38] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the streams-C high level language," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2000, pp. 49–56, doi: 10.1109/FPGA.2000.903392.

[39] B. L. Hutchings and B. E. Nelson, "Using general-purpose programming languages for FPGA design," in *Proc. 37th Conf. Design Autom.-DAC*, 2000, pp. 561–566, doi: 10.1145/337292.337581.

[40] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Parallelization of MATLAB applications for a multi-FPGA system," in *Proc. 9th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Mar. 2001, pp. 1–9.

[41] N. Tarafdar, G. Di Guglielmo, P. C. Harris, J. D. Krupa, V. Loncar, D. S. Rankin, N. Tran, Z. Wu, Q. Shen, and P. Chow, "AIgean : An open framework for deploying machine learning on heterogeneous clusters," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, pp. 1–32, Sep. 2022, doi: 10.1145/3482854.

[42] J. Knapheide, P. Kreowsky, and B. Stabernack, "Demonstrating NADA: A workflow for distributed CNN training on FPGA clusters," in *Proc. 33rd Int. Conf. Field-Program. Log. Appl. (FPL)*, Sep. 2023, p. 363, doi: 10.1109/FPL60245.2023.00068.

[43] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1143–1158, Aug. 2020, doi: 10.1109/TC.2020.3000118.

[44] E. Mageiropoulos, N. Chrysos, N. Dimou, and M. Katevenis, "Using hls4ml to map convolutional neural networks on interconnected FPGA devices," in *Proc. IEEE 29th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2021, p. 277, doi: 10.1109/FCCM51124.2021.00062.

[45] J. Haris, P. Gibson, J. Cano, N. Bohm Agostini, and D. Kaeli, "SECDA-TFLite: A toolkit for efficient development of FPGA-based DNN accelerators for edge inference," *J. Parallel Distrib. Comput.*, vol. 173, pp. 140–151, Mar. 2023, doi: 10.1016/j.jpdc.2022.11.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731522002301

[46] M. Porrmann, J. Hagemeyer, C. Pohl, J. Romoth, and M. Strugholtz, "RAPTOR-A scalable platform for rapid prototyping and FPGA-based cluster computing," in *Advances in Parallel Computing*, vol. 19. Amsterdam, The Netherlands: IOS Press, 2010, doi: 10.3233/978-1-60750-530-3-592.

[47] A. Mondigo, T. Ueno, K. Sano, and H. Takizawa, "Comparison of direct and indirect networks for high-performance FPGA clusters," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications (ARC 2020)* (Lecture Notes in Computer Science), vol. 12083. New York, NY, USA: Springer, 2020, pp. 314–329. [Online]. Available: http://link.springer.com/10.1007/978-3-030-44534-8_24

[48] A. Cicuttin, M. Liz Crespo, K. S. Mannatunga, J. G. Samarawickrama, N. Abdallah, and P. B. Sabet, "HyperFPGA: A possible general purpose reconfigurable hardware for custom supercomputing," in *Proc. Int. Conf. Adv. Electr., Electron. Syst. Eng. (ICAEES)*, Nov. 2016, pp. 21–26, doi: 10.1109/ICAEES.2016.7888002.

[49] W. F. Samayoa, M. L. Crespo, A. Cicuttin, and S. Carrato, "A survey on FPGA-based heterogeneous clusters architectures," *IEEE Access*, vol. 11, pp. 67679–67706, 2023, doi: 10.1109/ACCESS.2023.3288431.

[50] K. Sano, A. Koshiba, T. Miyajima, and T. Ueno, "ESSPER: Elastic and scalable FPGA-cluster system for high-performance reconfigurable computing with supercomputer fugaku," in *Proc. Int. Conf. High Perform. Comput. Asia–Pacific Region*, Feb. 2023, pp. 140–150, doi: 10.1145/3578178.3579341.

[51] (Jun. 2010). *DESIGN TOOLS–BEEcube Launches BEE4, a Full-Speed FPGA Prototyping Platform–EDN*. [Online]. Available: https://www.edn.com/design-tools-beecube-launches-bee4-a-full-speed-fpga-prototyping-platform/

[52] K. M. Keung, "A study of on-chip FPGA system with 2D mesh network," Ph.D. dissertation, Dept. Elect. Comput. Eng., Iowa State Univ., 2018.

[53] U. A. Gulzari, Z. Salcic, W. Farooq, S. Anjum, S. Khan, M. Sajid, and F. S. Torres, "Comparative analysis of 2D mesh topologies with additional communication links for on-chip networks," *Comput. Netw.*, vol. 241, Mar. 2024, Art. no. 110193, doi: 10.1016/j.comnet.2024.110193. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624000252

[54] (2021). *FX18 High-Speed Transmission Compatible 0.8mm Pitch Board-to-Board Connector*. [Online]. Available: https://www.hirose.com/ja/product/series/FX18#

[55] (2021). *SEARAY High-Density Open-Pin-Field Arrays*. [Online]. Available: https://www.samtec.com/high-speed-board-to-board/high-density-arrays/searay/

[56] S. Kunath. (2023). *Trenz TE0803 TRM*. [Online]. Available: https://wiki.trenz-electronic.de/display/PD/TE0803+TRM

[57] (2023). *SOM3–MPF300T*. [Online]. Available: https://www.sundanceedsp.com/som/polarfire/som-pf2-system-on-module-based-on-polarfire-mpf300t-1fcg784e/%7D

[58] (2023). *PolarFire SoC MSS Technical Reference Manual*. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/ReferenceManuals/PolarFire_SoC_FPGA_MSS_Technical_Reference_Manual_VC.pdf

[59] (2015). *INA228 Bi-Directional Current/Power Monitor With I2C Interface*. [Online]. Available: https://www.ti.com/product/INA228

[60] A. Tomori and Y. Osana, "Kyokko: A vendor-independent high-speed serial communication controller," in *Proc. 11th Int. Symp. Highly Efficient Accel. Reconfigurable Technol.*, Jun. 2021, pp. 1–6, doi: 10.1145/3468044.3468051.

[61] Z. Al-Ars, J. Petri-Koenig, J. Hoozemans, L. Dierick, and H. P. Hofstee, "OctoRay: Framework for scalable FPGA cluster acceleration of Python big data applications," in *Proc. SC Workshops Int. Conf. High Perform. Comput., Netw., Storage, Anal.*, Nov. 2023, pp. 539–546, doi: 10.1145/3624062.3624541.

[62] ICTP-Mlab and INTI-CMNT. (2021). *The Core Comblock*. [Online]. Available: https://gitlab.com/rodrigomelo9/core-comblock

[63] N. Dzemaili, "A reliable booting system for Zynq ultrascale+ MPSoC devices," Ph.D. dissertation, Dept. Elect. Eng., HU University of Applied Sciences, Utrecht, The Netherlands, 2021. [Online]. Available: https://cds.cern.ch/record/2763095

[64] M. Fuchs, L. E. Ardila-Perez, T. Mehner, and O. Sander, "Split boot—True network-based booting on heterogeneous MPSoCs," *IEEE Trans. Nucl. Sci.*, vol. 70, no. 6, pp. 1157–1163, Jun. 2023, doi: 10.1109/TNS.2023.3237968.

[65] K. S. Mannatunga, L. G. G. Ordóñez, M. B. Amador, M. L. Crespo, A. Cicuttin, S. Levorato, R. Melo, and B. Valinoti, "Design for portability of reconfigurable virtual instrumentation," in *Proc. X Southern Conf. Program. Log. (SPL)*, Apr. 2019, pp. 45–52, doi: 10.1109/SPL.2019.8714446.

[66] NERSC. (Sep. 2016). *Sshspawner*. [Online]. Available: https://github.com/NERSC/sshspawner

[67] M. L. Crespo, F. Foulon, A. Cicuttin, M. Bogovac, C. Onime, C. Sisterna, R. Melo, W. Florian Samayoa, L. G. García Ordóñez, R. Molina, and B. Valinoti, "Remote laboratory for E-learning of systems on chip and their applications to nuclear and scientific instrumentation," *Electronics*, vol. 10, no. 18, pp. 21–26, Sep. 2021, doi: 10.3390/electronics10182191. [Online]. Available: https://www.mdpi.com/2079-9292/10/18/2191

[68] (2021). *Vivado Design Suite User Guide*. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug973-vivado-release-notesinstall-license.pdf

[69] M. L. Crespo, A. Cicuttin, J. Dondo, and F. Rincón, "Reconfigurable virtual instrumentation based on FPGA for science and high-education," in *Field-Programmable Gate Array (FPGA) Technologies for High Performance Instrumentation*, J. Fagerberg, Ed., Hershey, PA, USA: IGI Global, 2016, ch. 5, pp. 99–123.

[70] P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing, "Binder 2.0–reproducible, interactive, sharable environments for science at scale," in *Proc. Python Sci. Conf.*, 2018, pp. 113–120, doi: 10.25080/majora-4af1f417-011.

[71] F. Perez and B. E. Granger, "IPython: A system for interactive scientific computing," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 21–29, May 2007, doi: 10.1109/mcse.2007.53. [Online]. Available: https://ipython.org

[72] W. F. Samayoa, "HyperFPGA: SoC-FPGA cluster architecture for supercomputing and scientific applications," Ph.D. dissertation, Dept. Eng. Archit., University of Trieste, Trieste, Italy, 2024. [Online]. Available: https://hdl.handle.net/11368/3068428

[73] (2017). *Amazon EC2 F1 Instances*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[74] OEIS Foundation. (2023). *Number of Ways of Placing N Nonattacking Queens on an N X N Board*. [Online]. Available: https://oeis.org/A000170

[75] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/bf01386390.

[76] W. Florian, B. Valinoti, L. G. García, M. Cervetto, E. Marchi, M. L. Crespo, S. Carrato, and A. Cicuttin, "An open-source hardware/software architecture for remote control of SoC-FPGA based systems," in *Applications in Electronics Pervading Industry, Environment and Society (ApplePies 2021)*. New York, NY, USA: Springer, 2022, pp. 69–75, doi: 10.1007/978-3-030-95498-7_10.

[77] S. Lee, J. Kim, and J. S. Vetter, "OpenACC to FPGA: A framework for directive-based high-performance reconfigurable computing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2016, pp. 544–554, doi: 10.1109/IPDPS.2016.28.

[78] Y. Osana, T. Imahigashi, and A. Tomori, "OpenFC: A portable toolkit for custom FPGA accelerators and clusters," in *Proc. 8th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Nov. 2020, pp. 185–190, doi: 10.1109/CANDARW51189.2020.00045.

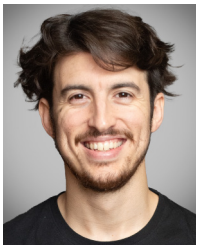
[79] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90429–90455, 2022, doi: 10.1109/ACCESS.2022.3201107. [Online]. Available: https://ieeexplore.ieee.org/document/9864576/

[80] M. G. Jordan, G. Korol, T. Knorst, M. B. Rutzig, and A. C. S. Beck, "Energy-aware fully-adaptive resource provisioning in collaborative CPU-FPGA cloud environments," *J. Parallel Distrib. Comput.*, vol. 176, pp. 55–69, Jun. 2023, doi: 10.1016/j.jpdc.2023.02.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523000308

WERNER FLORIAN SAMAYOA received the B.S. degree in electronics engineering from the University of San Carlos, Guatemala, in 2018. He is currently pursuing the Ph.D. degree in industrial and information engineering with the Universitá degli Studi di Trieste, under a Joint-Supervision Program with the Multidisciplinary Laboratory (MLab), The Abdus Salam International Center for Theoretical Physics. His research interests include scalable and reconfigurable supercomputing.

MARIA LIZ CRESPO is currently a Research Officer with The Abdus Salam International Centre for Theoretical Physics (ICTP) and an Associate Researcher of Italian National Institute for Nuclear Physics (INFN), Trieste, Italy. She is also coordinating the research and training program of the Multidisciplinary Laboratory (MLab), ICTP. She has organized several international schools and workshops on fully programmable systems on chip for nuclear and scientific instrumentation. She is the co-author of more than 100 scientific publications in prestigious peer-reviewed journals. Her main research interests include advanced scientific instrumentation for particle physics experiments and experimental multidisciplinary research.

SERGIO CARRATO received the master's degree in electronic engineering and the Ph.D. degree in signal processing from the University of Trieste, Trieste, Italy. He then worked with Ansaldo Componenti and Sincrotrone Trieste, in the field of electronic instrumentation for applied physics. He joined the Department of Electronics, University of Trieste, where he is currently an Associate Professor of electronic devices.



AGUSTIN SILVA is currently pursuing the Ph.D. degree in computational modeling. He is an Electronics Engineer. He has been actively involved in FPGA and Quantum Computing, since 2017, areas central to both his undergraduate and graduate research, resulting in multiple scientific publications. Currently, he holds a postdoctoral position with the Technology Innovation Institute, Abu Dhabi, where his work focuses on designing RFSoC FPGA-based control electronics for superconducting quantum computers.

MAYNOR GIOVANNI BALLINA ESCOBAR received the degree in electronic engineering from the Universidad de San Carlos de Guatemala. He is currently pursuing the joint Ph.D. degree with the International Centre for Theoretical Physics (ICTP) and the University of Trieste, Italy. His research focuses on heterogeneous computing in MPSoC-FPGA and scientific instrumentation, particularly in the development of Cherenkov detectors. He is also involved in advanced instrumentation projects at the Universidad de San Carlos de Guatemala.

ANDRES CICUTTIN received the degree in physics from the National University of La Plata, Argentina, in 1992, and the Laurea degree in fisica from the University of Trieste, Italy, in 1993. He is currently a Technical Assistant with the Multidisciplinary Laboratory, The Abdus Salam International Centre for Theoretical Physics, and an Associate Researcher with Italian National Institute for Nuclear Physics (INFN). He has organized and directed numerous international workshops on programmable logic devices for scientific instrumentation and high education.