# OPEN-CFR: Open-source Co-design Framework for Redundancy with DPR in COTS FPGA SoCs

Francesco Restuccia*
*University of California San Diego*
La Jolla, CA, USA
frestuccia@ucsd.edu

Biruk Seyoum*
*Columbia University*
New York, NY, USA
biruk@cs.columbia.edu

Alexander Redding
*University of California San Diego*
La Jolla, CA, USA
alredding@ucsd.edu

Zhenghua Ma
*University of California San Diego*
La Jolla, CA, USA
zhm007@ucsd.edu

Guy Eichler
*Columbia University*
New York, NY, USA
guyeichler@cs.columbia.edu

Luca Carloni
*Columbia University*
New York, NY, USA
luca@cs.columbia.edu

Ryan Kastner
*University of California San Diego*
La Jolla, CA, USA
kastner@ucsd.edu

***Abstract*—Commercial-off-the-shelf (COTS) Field Programmable Gate Array (FPGA) systems-on-chip (SoCs) are flexible platforms combining a Processing System (PS) featuring high-performance embedded-class processors with a configurable FPGA subsystem for the deployment of hardware accelerators and ad-hoc custom peripherals. Flexibility combined with demonstrated high performance and high energy efficiency make COTS FPGA SoCs particularly attractive for space applications. One of the main challenges in using COTS FPGA SoCs in space applications is the susceptibility of the FPGA technology to Single Event Upsets (SEU) caused by ionizing particles. This paper presents OPEN-CFR, an Open-source Co-design Framework for redundant execution of hardware accelerators on COTS FPGA SoC platforms. From a high-level description of the target system, OPEN-CFR provides: (i) an automatically generated hardware shell supporting voting and detection, specifically tailored for the interface of a target hardware accelerator, (ii) a full Dynamic Partial Reconfiguration flow for fast recovery after SEU, and (iii) a generated software runtime executable for the setup and runtime management of the full redundant system. OPEN-CFR automates the creation of the full FPGA design, providing as output the bitstreams and software executable for the target FPGA SoC platform. We evaluate the performance of OPEN-CFR and compare it with state-of-the-art solutions on realistic experimental scenarios and on a use-case scenario deploying the HLS4ML framework on popular COTS FPGA SoCs from the ZYNQ family from AMD-Xilinx.**

## I. INTRODUCTION

Field-Programmable Gate Arrays (FPGA) are popular platforms for the implementation of space applications [1], [2], [3]. They are currently in use for enabling communications, sensors, instruments, and systems in applications ranging from low Earth orbit satellites to space missions and deep-space exploration [4], [5]. Multiple vendors provide *space-grade* FPGA platforms that are typically based on anti-fuse or flash technologies. However, traditional space-grade FPGA platforms have significant limitations compared to commercial-off-the-shelf (COTS) SRAM-based FPGA platforms. These limitations include: (i) deployment in older technology nodes, which leads to lower performance and worse energy efficiency; (ii) being significantly expensive; (iii) the single programming capability of anti-fuse-based FPGAs, which lacks the flexibility of reprogrammable COTS FPGAs; (iv) flash-based FPGAs generally not matching the performance levels of COTS SRAM-based FPGAs.

Among the commercially available SRAM-based FPGA platforms, FPGA systems-on-chip (SoCs) combine a programmable FPGA fabric with a Linux-capable processing system, typically deploying multiple hard silicon ARM-based processors and peripherals [6], [7]. FPGA SoC platforms support a great level of flexibility, enabling the deployment of fully-featured Linux-based applications, enriched with custom functionalities deployed in the FPGA fabric as hardware accelerators or custom peripherals. The Dynamic Partial Reconfiguration (DPR) feature adds another level of flexibility to these platforms, enabling the partial runtime reconfiguration of the FPGA fabric while keeping the rest of the FPGA subsystem running. The features enabled by FPGA SoCs are particularly relevant in modern applications requiring hardware acceleration, for instance, when complex deep neural network (DNN) algorithms must be efficiently computed [8], [9], [10], [11].

Several research works have focused on the use of COTS FPGA in space applications. However, the application of COTS FPGAs in space applications comes with its own set of challenges [12] – one of the major difficulties is

*These authors contributed equally to this work the susceptibility of SRAM-based COTS FPGA platforms to Single-Event Upsets (SEUs) caused by space radiation. SEUs are capable of corrupting the configuration memory of an SRAM-based FPGA, thus altering the implemented function. This can lead, in the worst-case scenario, to the generation of incorrect results, which can potentially cause fatal consequences. Clearly, this scenario should be avoided by the majority of critical applications.

Redundant execution is a popular and effective approach to improving the reliability of systems executing under the effect of SEUs – it consists of deploying and synchronizing the execution of multiple replicated instances of the same accelerator. According to the required functionalities, redundant execution can provide different features, ranging from single error detection in dual modular redundancy (DMR), single error detection and correction in triple modular redundancy (TMR), and possibly more advanced features when deploying more than three instances of the replicated module.

The research community widely investigated redundant execution on FPGAs (Section II). On the commercial side, multiple vendors provide commercial tools for the deployment of redundant designs [13], [14]. The industrial solutions focus on generic RTL developments, lacking features for an automated full-system generations for FPGA SoCs. Moreover, most of the industrial solutions are closed-source, preventing modifications and eventually improvements, thus limiting the contributions that can be provided by the research community. Finally, on the research community side, a holistic open-source co-design framework that supports the automatic generation of redundant designs along with the software runtime, and a flexible Dynamic DPR flow targeting COTS FPGA SoC platforms is not yet available.

**Contribution:** This paper proposes OPEN-CFR: an Open-source Co-design Framework for Redundancy utilizing the DPR capability in COTS FPGA SoCs. OPEN-CFR is a co-design framework supporting the integration, development, and execution of redundant hardware-accelerated applications on commercial FPGA SoC platforms. OPEN-CFR is based on: (i) a configurable *hardware shell* composed of two IPs supporting the synchronization, voting, detection, and correction of runtime execution faults on TMR; (ii) a full DPR flow for the deployment and placement of the multiple instances of the target accelerator in multiple reconfigurable slots to enable fast recovery; (iii) a *software runtime* for the configuration, runtime management, and recovery (triggering DPR of the slots when a fault is detected by the hardware shell); and (iv) a set of scripts to automate the generation and integration of the hardware shell, the full RTL of the design (including different IPs), the DPR flow, and the software runtime. OPEN-CFR takes as inputs a configuration file (a high-level description of the system), the target accelerator source files in IP-XACT format (without requiring the RTL, in order to support third-party accelerator IPs), and software code to execute on the processors in the processing system (PS). OPEN-CFR generates the static bitstream, the dynamic (partial) bitstreams, and the software runtime executable for the configuration and management of the whole system. We experimentally evaluate the performance and area impact of OPEN-CFR on realistic designs and on the HLS4ML flow for the acceleration of DNN algorithms on COTS FPGA SoCs, comparing the measured results with a state-of-the-art commercial tool for the Xilinx-AMD FPGA SoC platforms.

Open-CFR is open-source [15]. This baseline solution can be leveraged by the broad research community – we hope with that to stimulate further research contribution on the topic. We discuss this fundamental matter in more detail in Section V.

## II. Related Works

Several previous works have focused on redundant execution to increase the reliability of a system executing in harsh environments, such as outer space.

Le et al. [12] recently described the features and challenges in designing space systems based on FPGA technologies. Su et al. [16] proposed a methodology aimed at leveraging the programmable logic in FPGA SoC platforms for supporting the redundant execution of the Cortex A53 cores integrated in the PS of Xilinx-AMD Zynq Ultrascale+ platforms. Doran et al. [17] proposed a methodology for supporting synchronization in lockstep execution of cores/processors, putting forward the concept of post-processing resource release to facilitate the implementation of redundant core/processor arrays. Kastensmidt et al. [18] investigated how to place the voter's logic into a redundant design, showcasing how the reliability of the system depends on the placement of voters in the design. Lazaro et al. [19] proposed a methodology to support redundant communication in low-speed peripherals. Such a solution is limited to the RTL design and on AXI-lite communication interfaces.

Osterloh et al. [20] provided an extensive survey of the DPR features available in space-grade FPGAs. Pilotto et al. [21], Heiner at al. [22], Bolchini et al. [23], and Straka et al. [24] investigated methodologies for leveraging dynamic partial reconfiguration to recover from SEU in redundant designs.

Barbirotta et al. [25] presented a methodology leveraging open-source RISC-V architectures to support replicated thread execution. Sarraseca et al. [26] proposed an open-source implementation and SoC integration of a RISC-V lockstep core based on Gaisler's NOEL-V core for space domain [27]. Barbirotta et al. [28] investigated the integration of co-processors in a fault-tolerant RISC-V core, evaluating advantages and weaknesses. Wilson et al. [29] investigated the SEU sensitivity in popular RISC-V open-source processors through an automatic fault injection setup leveraging DPR.

The research community also spent considerable efforts in providing frameworks supporting various functionalities. Jacobs et al. [30] proposed a reconfigurable fault-tolerance framework for FPGAs, enabling the dynamic adjustment of system redundancy and fault mitigation from the radiation incurred at different orbital positions. Lee et al. [31] extended LegUp [32], proposing a framework for generating TMR designs for FPGA platforms from C programs. Xin et al. [33] presented multiple methodologies aimed at implementing

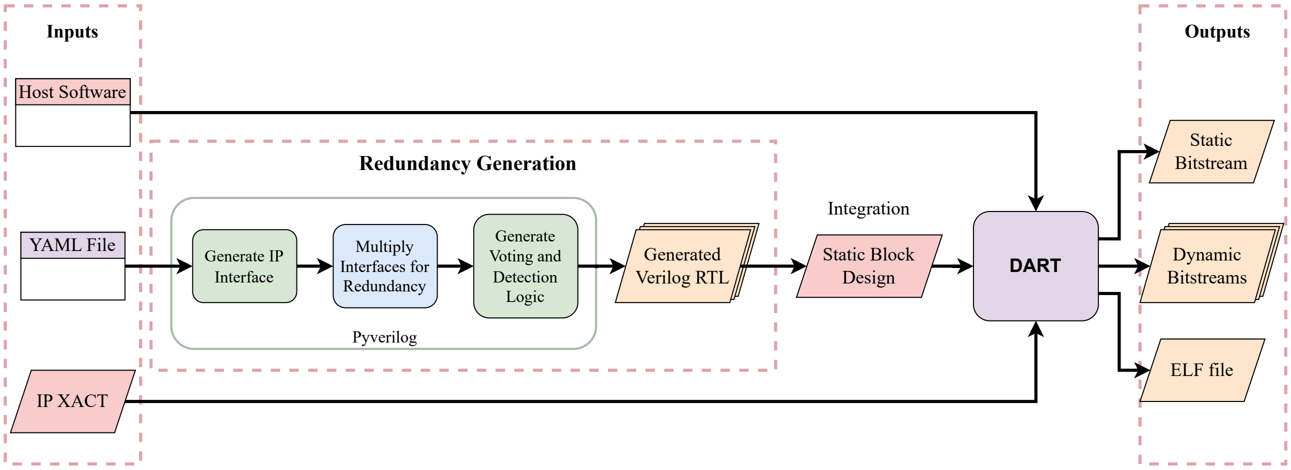

Fig. 1. The internals of the OPEN-CFR framework building flow.

TMR designs in FPGA platforms. Nema et al [34] proposed a framework, based on RTL simulations, for finding vulnerable components in hardware designs through fault injection and fault propagation tracking. Shreejith et al. [35] proposed an approach to redundancy specifically focused on safety-critical automotive systems, leveraging DPR and a custom bus controller.

On the industry side, AMD-Xilinx provides the TMR Tool, a collection of IPs for the deployment of redundant applications on most of their commercial FPGA platforms. Synopsys provides the Synplify tool for the development and implementation of high-reliability designs for FPGA platforms, including the option of generating redundant execution. While effective, these tools do not provide support for DPR integration, a software runtime, nor an automatic tool for the generation of redundant designs. Moreover, since the IPs provided by the TMR tool are closed-source, they prevent any direct modification, enrichment, or improvement.

A co-design open-source framework for supporting the implementation of redundant applications on COTS FPGA SoCs, integrated with a full DPR flow for fast recovery is still missing.

## III. THE OPEN-CFR FRAMEWORK

This section describes the OPEN-CFR framework and is organized as follows: Section III-A describes the framework flow and its internals, from the provisioning of the input files to the generation of the bitstreams and software runtime. Section III-B provides an overview of the hardware and software architecture of a system deployed with OPEN-CFR. Section III-C describes the system runtime functionalities – we provide a discussion on potentially interesting extensions and future work in Section V.

We assume a typical hardware acceleration flow in FPGA SoCs, where the hardware accelerators or custom IPs are deployed into the FPGA fabric, enriching the execution of the high-performance embedded processors deployed in the PS of the FPGA SoC. For maximum flexibility, we take as reference one of the most popular standard interfaces used for hardware accelerators in modern FPGA SoCs, composed of (see Figure 2): (i) a data AXI-full manager port (M) [36]. This port is leveraged by the accelerator to autonomously fetch data and write results to/from the main DRAM memory subsystem in the PS; (ii) a configuration AXI-lite subordinate port (S) [36]. This port is accessed by the processors in PS for the configuration of the accelerator and triggering the start of the execution. In simple low-performance hardware accelerators lacking a data port, the configuration port can also be leveraged by the processors for feeding the accelerator with the data to execute on and to read its produced results; (iii) an interrupt signal, notifying the processors when the execution of the accelerator is done, thus the availability of fresh results produced by the accelerator. It is worth mentioning that in addition to the native interfaces we described above, OPEN-CFR can be easily extended to support the integration of accelerators using other standard interfaces or custom interfaces.

OPEN-CFR does not rely on the RTL of the integrated accelerators – we assume that the RTL might not be available (as it can happen, for instance, when integrating third-party IPs). This feature ensures seamless support for a wide range of IPs, including those internally developed, sourced from open-source repositories, or obtained from third-party vendors with encrypted RTL. OPEN-CFR leverages the interface of the accelerator IP to detect when a fault hits one of the replicated instances of the accelerator, by monitoring and detecting at runtime any mismatch in the results produced by the replicated instances of the accelerator.

### A. *The* OPEN-CFR *Framework Flow*

Figure 1 shows a schematic representation of the OPEN-CFR co-design flow. OPEN-CFR takes as input: (i) the OPEN-CFR YAML configuration file, (ii) the IP-XACT representation of the hardware accelerator, and (iii) the C/C++ software configuration code for the accelerator and provides as outputs: (i) the static bitstream to be deployed on the FPGA static part, including the reconfiguration logic, the redundancy logic,

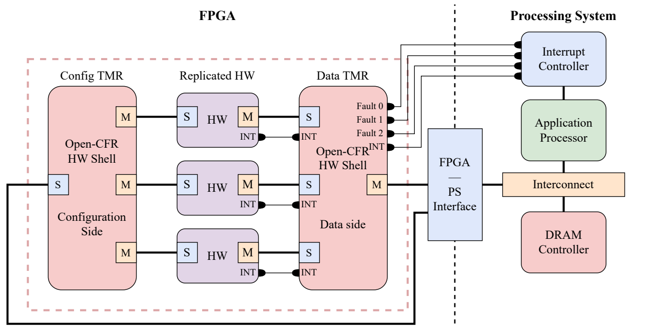

Fig. 2. The OPEN-CFR sample hardware architecture on a commercial FPGA SoC.

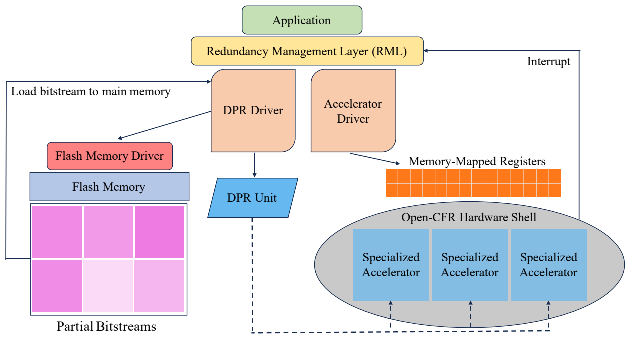

Fig. 3. The software architecture of the OPEN-CFR framework.

and the system interconnect, (ii) the partial bitstreams, to be deployed in the DPR reconfigurable slots, and (iii) the executable software runtime, running in the processors in the PS and in charge of setting up and manage the whole system.

The first step of the OPEN-CFR flow is generating a representation of the accelerator interface, as described by the YAML configuration file, in a Verilog Abstract Syntax Tree (VAST) using Pyverilog [37]. The second step is multiplying the VAST accelerator interface to obtain triple modular redundancy. The third step combines voting and detection logic that is generated for the interface signals on the redundant accelerators as a VAST module – the hardware shell is generated, which allows the processor to access the redundant replicas of the hardware accelerator as a single unit. The fourth step is to convert the generated VAST logic to Verilog RTL using the Pyverilog code generation tool. In the final step, the generated hardware shell is provided as an input to DART [38], an open-source tool integrated with OPEN-CFR to automate the DPR design flow on AMD-Xilinx FPGA SoCs.

While DART provides the capability to generate the full and partial bitstreams of a traditional DPR design along with runtime support for baremetal and Linux applications, we augmented it to support redundancy. Our modifications include: (i) a seamless duplication of the runtime reconfigurable accelerator IPs, (ii) the automatic insertion of the hardware shell into the design and automation of the connection with the accelerators, (iii) automatic generation of the configuration address-map and device tree for a transparent execution from software, and (iv) finally, the generation of the full and partial bitstreams of the system in a single push-button flow.

## *B. The Architecture of An* OPEN-CFR*-Generated System*

*1) Hardware architecture:* Figure 2 shows a simplified representation of the hardware architecture of a OPEN-CFR-generated system on a COTS FPGA SoC platform. The left side of Figure 2 shows the FPGA subsystem, while the right side represents the PS. As depicted in the figure, OPEN-CFR creates three identical instances of the target accelerator $HW$, named $HW_1$, $HW_2$, and $HW_3$. The DART tool [38] for DPR allocates a reconfigurable slot for each replica of the accelerator. Each replica is deployed on a DART slot. On the left, the configuration ports of the replicated accelerators are connected to the configuration side of the OPEN-CFR-generated hardware shell for redundant execution, each to a manager AXI-lite port. In turn, configuration side of the shell exports a subordinate interface directly connected to the PS and used by the software runtime for the configuration of the replica accelerators. In addition to configuring the system, the configuration side of the hardware shell enforces synchronous execution of the replicas, votes their provided handshakes, and provides fault detection functionalities during configuration using interrupts to the PS, enabling a prompt response of the system in case of fault during configuration.

The data ports of the accelerator replicas are connected to the data side of the OPEN-CFR-generated hardware shell for redundancy, each to an AXI-full subordinate interface. In turn, the data side of the shell exports a manager AXI-full interface, connected to the FPGA-PS interface to provide access to the the main DRAM memory subsystem. The data side of the hardware shell implements voting on the data results produced by the replica accelerators and handshakes– the majority voting of the results is eventually written to DRAM memory. Besides the voting logic, the detection logic is also deployed on this side of the shell – any mismatch in the data produced by the accelerator replicas or handshake is detected and signaled with an interrupt to the OPEN-CFR software runtime running in the PS.

The OPEN-CFR hardware shell deploys an interrupt line for each deployed DPR slot, each signaling a fault detected in the specific slot to the software runtime running in the PS. As TMR is deployed, three interrupt lines are exported to the PS.

*2) Software architecture:* Figure 3 shows the software architecture of the OPEN-CFR framework. At the top, a software application that runs on a resource constrained device requires the assistance of a specialized hardware accelerator in order to complete the execution of a task with a higher throughput and better energy efficiency. The software application allocates the memory for the input and the output of the task, and sets the input values. The application aggregates any additional task-specific information that is needed in order to execute the task. For instance, the execution of a 2D convolution requires an array for the input, an array for the output, dimensions and weights for the kernels, padding information, pooling information, etc.

From the software application layer, the information is passed to the Redundancy Management Layer (RML). The information must be organized by the application in a software data `struct` named $RML\_t$ which includes a string with the name of the requested task, a pointer to the input array, the dimensions of the input, a pointer to the output array, the dimensions of the output, and a pointer to an array that includes additional task-specific arguments. The RML captures the $RML\_t$ and instantiates a couple of device drivers.

The first driver is the DPR driver. This driver is pre-configured by DART at design time to hold a data `struct` named $bit\_t$ that contains the names and sizes of the available partial bitstreams. Each partial bitstream includes a specialized accelerator for a specific task. The partial bitstreams are pre-loaded at design time into a flash memory that can be accessed on the FPGA board via a designated device driver. The DPR driver reads the name of the task from $RML\_t$, matches it to an entry in $bit\_t$ and requests the specific bitstream file from the flash memory driver. The flash memory driver fetches the partial bitstream and stores it in the main memory of the device to make it available for the DPR driver. Then, the bitstream is passed to the DPR unit from DART, which re-configures the available slots in the programmable logic with the partial bitstream.

The second driver is the one that instantiates the specialized accelerator. This driver receives all the information to invoke one instance of the accelerator and successfully executes the task including the information inside $RML\_t$. The driver writes the information into memory-mapped registers which correspond with the OPEN-CFR hardware shell. The shell configures all of the redundant copies of the accelerator with the same configuration. When the device driver initiates an accelerator invocation through the memory-mapped registers, the shell invokes all the underlying accelerators in parallel (see Section III-C1).

An interrupt mechanism between the OPEN-CFR hardware shell and the RML allows the shell to request a reconfiguration in a specific accelerator slot. When an interrupt is generated, the RML triggers a reconfiguration in the specific slot indicated by the shell and the partial bitstream is loaded by the DPR unit. At the end of the execution, the output of the task is stored inside the output array that was allocated by the application. The OPEN-CFR hardware shell writes the output into the main memory according to the address that was previously provided within $RML\_t$ by the application.

## C. *Runtime functionalities description*

This section describes the runtime functionalities of OPEN-CFR, involving configuration and synchronization, voting and error detection, and recovery with the DART DPR flow.

*1) Configuration and synchronization of the replicas:* As mentioned at the beginning of Section III, modern hardware accelerators are typically configured with software running in the processors embedded in the PS. Thus, when a OPEN-CFR-based design is deployed in the system, the first two steps operated after the overall setup of the platform are: (i) the first configuration of the replicas of the hardware accelerator, and (ii) the execution synchronization. In some scenarios, reconfiguration can also be required at runtime, for instance, for changing parameters in the function implemented by the accelerator or updating the memory reference pointers in the accelerator for fetching data or writing the results.

OPEN-CFR aims at full user software transparency – the synchronization of the multiple replicas of the accelerator is directly managed and abstracted by the OPEN-CFR hardware shell. The software sees a unique address space associated with the multiple accelerator replicas rather than a separate address space for each replica, meaning that the standard user software code for the interaction with a single accelerator, which is one of the inputs of OPEN-CFR, can be seamlessly deployed with no user modifications required.

The processor running software triggers a configuration by accessing the secondary interface of the OPEN-CFR-generated hardware shell. The hardware shell replicates the configuration for each replica, presenting the same data to all of the replicas at the same clock cycle. Providing the same configuration with exactly the same timing to all of the replicas, the OPEN-CFR hardware shell guarantees not only an identical configuration of the cores but also enforces execution synchronization among them, ensuring that all of the replicas: (i) share the same configuration, and (ii) start execution synchronously. Thus, the replicas run in lockstep.

The previous case described a modern accelerator exporting a configuration port, in which the start of the accelerator computation is typically triggered by setting an internal register. In the case in which the integrated accelerator does not support/require a software configuration, OPEN-CFR can support the lockstep synchronization of the replicas leveraging a ”start execution” signal, typically featured in simpler accelerators. OPEN-CFR also supports scenarios in which the accelerator does not export a data port for autonomously accessing the DRAM memory in the PS. In these cases, the interface of the hardware accelerator is simplified and limited to its configuration port and the interrupt signal (as described at the beginning of Section III). In this last scenario, the hardware shell also takes care of feeding the replicas with the same data provided by the processors at the same cycle.

*2) Voting and Error detection:* Besides configuration and synchronization, the OPEN-CFR-generated hardware shell is also in charge of voting on the data produced by the multiple replicas and of detecting faults during execution. As from the previous section, the hardware shell guarantees that the cores are configured with the same data and run in lockstep. This means that any mismatch in the data produced by the replica accelerators or any interface handshake inconsistency indicates a fault in one of the replicas.

The OPEN-CFR hardware shell operates fault detection and majority voting of the outputs. The capacity of error detection and correction depends on the deployed redundant architecture – the architecture under analysis deploying TMR support error correction of one faulty replica and error detection. When a fault is detected, the OPEN-CFR hardware shell triggers the interrupt corresponding to the DPR slot deploying the detected faulty replica. At this point, the OPEN-CFR software runtime will take care of triggering a DPR recovery of the signaled slot, as discussed next.

*3) Recovery with Dynamic Partial Reconfiguration:* Upon detecting a fault, an interrupt is triggered to notify the software runtime. The hardware shell exports an individual interrupt line for each DPR slot. Depending on the recovery policy, the software framework sends a command to the reconfiguration controller inside the FPGA fabric to perform partial reconfiguration of the faulty accelerator. The configuration controller fetches the bitstreams from the flash memory and configures the FPGA via the ICAP interface [7].

## IV. EXPERIMENTAL VALIDATION

This section describes the experiments we conducted to validate OPEN-CFR on COTS FPGA SoCs and compare it with a state-of-the-art solution. We first evaluated OPEN-CFR on synthetic tests, leveraging a commercial, closed-source DMA [39] as the integrated accelerator, and evaluating the performance impact in terms of overall execution performance (Section IV-B) and resource consumption (Section IV-C) of a OPEN-CFR-generated system, compared with a system deployed using the IPs of the TMRTool [13] provided by AMD-Xilinx. In these tests, the DMA represent the behavior of a generic hardware accelerator. Then, we evaluated our solution on a real use case scenario, implementing redundancy on the HLS4ML framework for the acceleration of DNN algorithms on FPGAs (Section IV-Es). The section starts with a description of the experimental setup.

### A. Experimental setup

We developed and deployed realistic architectures on an AMD-Xilinx Zynq Ultrascale+ ZCU104 FPGA SoC platform. The architecture under analysis in Section IV-B and Section IV-C integrates an AMD-Xilinx DMA as a hardware accelerator [39]. The DMA is replicated in three instances by OPEN-CFR– the hardware architecture mirrors the architecture reported in Figure 2; the configuration interfaces and the data port are connected to the OPEN-CFR hardware shell. The configuration port of the hardware shell is in turn connected to the PS-FPGA interface for software configuration. The data port is connected to the FPGA-PS interface, and in turn, to the central DRAM memory controller. Thanks to the transparency of OPEN-CFR from the software perspective, we leveraged the standard bare-metal software code provided by the vendor for the DMA with no modification. For comparison, we deployed a redundant application based on the same replicated DMA leveraging the hardware IPs provided in the TMRTool [13].

In Section IV-E, we evaluate OPEN-CFR on a real use-case scenario, implementing redundancy with our framework on the HLS4ML hardware-accelerated system for DNN algorithms [40].

We leverage the standard AMD-Xilinx Vivado flow for the synthesis and implementation of the designs. We kept the default 100 MHz clock configuration for the FPGA. To measure execution time performance, we leverage the AMD-Xilinx Vitis baremetal workflow under debug mode.

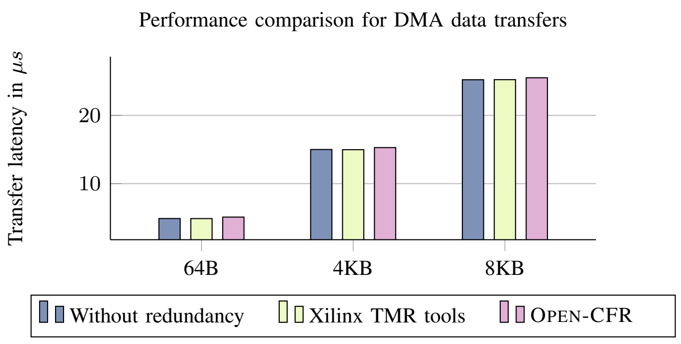

Fig. 4. Performance evaluation at the firmware level of a OPEN-CFR-generated redundant design compared with the AMD-Xilinx TMR-tool and the design without redundancy. Results are reported on a linear scale. The measured results are comparable in all of the scenarios (average reported).

### B. Execution time performance evaluation

This experiment compares the performance of a reference design deploying a single DMA with: (i) a redundant TMR design developed using the IPs of the Xilinx-AMD TMR Tool, and (ii) a OPEN-CFR-generated redundant system. We configured the DMAs through its configuration port to move different amounts of data from the central DRAM memory in PS (64 Bytes, 4 KBytes, and 8KBytes) through its data port (see Figure 2). We measure the overall completion time, from the start of the configuration to the interrupt raised by the DMA signaling the completion of the data move. We made 1000 runs for each of the designs under evaluation, measuring their respective completion times. Figure 4 reports the average measured performance, as a function of the different amounts of data moved. As expected, the results are comparable for all of the tested scenarios: the redundancy logic is fully combinatorial, thus it introduces no cycle delays. This experiment confirms that a OPEN-CFR-generated redundant architecture offers a performance that is comparable with the state-of-the-art solution for the target FPGA SoC platform.

### C. Resource consumption

In this second experiment, we evaluate the hardware resource consumption of an OPEN-CFR-generated design, comparing the resource cost in Lookup Tables (LUTs) (Figure 5) and Flip-Flops (FFs) (Figure 6) of a reference design deploying a non-redundant DMA with: (i) a redundant TMR design developed using the IPs of the Xilinx-AMD TMRtool, and (ii) a OPEN-CFR-generated redundant system. We report on the chart the cost of the overall hardware design (total) and the cost of the voting logic. The resource consumption of a single DMA and the interconnect deployed in the design are also reported for comparison purposes. The reported data are collected from the AMD-Xilinx Vivado tool used for the synthesis and implementation of the designs. For a fair comparison with the AMD-Xilinx TMR Tool, the reported resources do not include the logic for DPR.

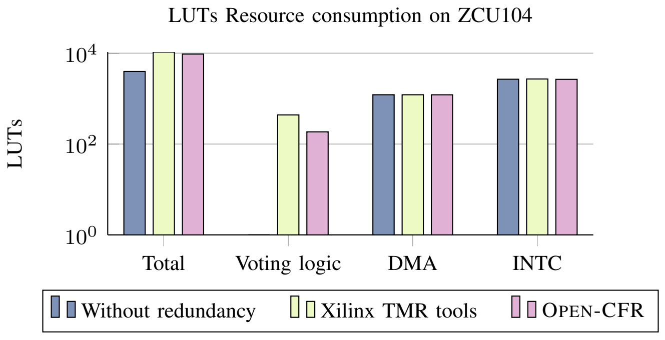

Fig. 5. LUTs resource consumption of a OPEN-CFR-generate design compared with the AMD-Xilinx TRM tool. Resource consumption for a single DMA and the Interconnect (INTC) are reported for comparison purposes. Results are reported on a logarithmic scale.

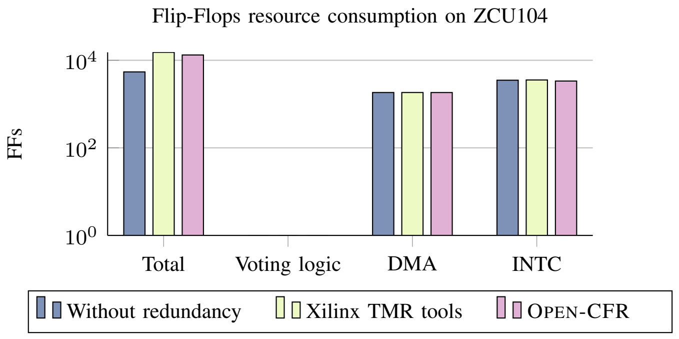

Fig. 6. Flip-Flops resource consumption of a OPEN-CFR-generate design compared with the AMD-Xilinx TRM tool. Resource consumption for a single DMA and the Interconnect (INTC) are reported for comparison purposes. Results are reported on a logarithmic scale.

Figure 5 and Figure 6 demonstrate that the impact on hardware resources, in terms of total resource consumption, of a OPEN-CFR-generated redundant system is comparable with designs deployed using the IPs from the AMD-Xilinx TMR-tool. Solely considering the voting logic, Figure 5 showcases how the OPEN-CFR-generated voting logic has lower LUT consumption concerning the IPs of the AMD-Xilinx TMR Tool. Since the redundancy logic is fully combinational, The implementation of the voting logic does not require any flip-flop, as shown in Figure 6. The experiments confirm that a redundant application generated with the OPEN-CFR flow has comparable resource consumption concerning a commercial solution.

## D. Use-case scenario: the HLS4ML framework

FPGAs are increasingly used in particle physics research because they can provide extremely high-throughput and low-latency data inference for many experiments. Scientists have developed a number of neural network architectures for particle classification, including the jet-tagger [41]. However, implementing a neural network in hardware logic from scratch is difficult and tedious. Fortunately, HLS4ML[42] enables scientists to do fast and easy prototyping of neural networks on FPGAs. HLS4ML is an an end-to-end toolchain that converts

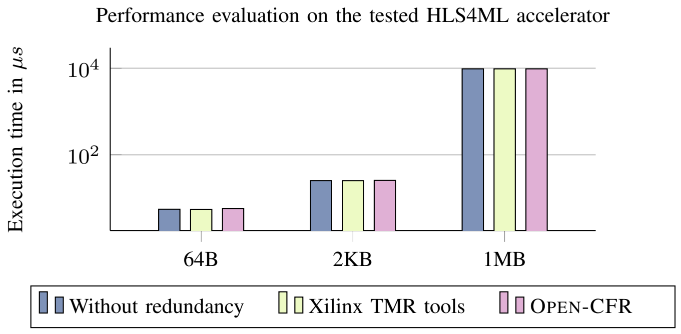

Fig. 7. Performance evaluation at the firmware level of a OPEN-CFR-generated redundant design compared with the AMD-Xilinx TMR-tool and the design without redundancy. Results are reported in logarithmic scale. The measured results are comparable in all of the scenarios (average reported).

a neural network defined using popular ML frameworks (e.g., PyTorch, TensorFlow, and Keras) to an RTL accelerator.

We leveraged the OPEN-CFR framework for implementing a redundant system featuring of a jet-tagger accelerator generated with HLS4ML workflow. Similarly to section IV-B, we compare the results for the OPEN-CFR-generated system with the TMR tool and the reference design without redundancy. Also in this case, we collected data for 1000 runs.

Figure 7 reports the average measured results, as a function of the amount of data processed by the jet-tagger accelerator. Also in this case, the performance of the OPEN-CFR-generated system is comparable with the design generated by the state-of-the-art tool. This confirms the applicability of our solution to real-world use case scenarios.

## E. Recovery-time

The last set of our experiments focuses on the runtime recovery time after fault detection for a case-study system developed using OPEN-CFR. The recovery time is defined as the total time it takes to configure the configuration controller (*pre-config*), load the bistream on the FPGA (*partial/full config*), and finally configure and invoke the accelerators (*post-config*) after a *fault* interrupt is received. We utilized the *sum_vec* and *mul_vec* accelerators from the DART IP repository [38]. In each invocation, the accelerators fetch 1024 64-bit width vectors and perform addition and multiplication respectively. We implemented the system using the two accelerators in a TMR configuration in OPEN-CFR and generated a total of seven bitstreams (six partial bitstreams for each of the accelerator and one full bitstream) targeting a 100 MHz clock. Table II provides a summary of the resource consumption of each accelerator as well as the sizes of the bitstreams. Note that, the sizes of the partial bitstreams varies widely even for similar types of accelerators. This is due to the differing floorplan sizes of the reconfigurable regions of the fabric that are automatically generated using DART [43].

For this experiment, we triggered *fault* by intentionally mis-configuring one of the accelerators with the wrong value at runtime. Figure 8 provides the results of the comparison of fault recovery for both accelerators under full reconfiguration

TABLE I
FEATURES COMPARISON BETWEEN OPEN-CFR AND THE AMD-XILINX TMR TOOL.

| | Resource consumption | Performance | Software Co-design | Full DPR Flow | Automatic Design Generation | Open Source |
|---|---|---|---|---|---|---|
| TMRTool | Comparable | Comparable | No | No | No | No |
| OPEN-CFR | Comparable | Comparable | Yes | Yes | Yes | Yes |

TABLE II
SUMMARY OF THE FPGA RESOURCE UTILIZATION AND BITSTREAM SIZES OF THE CASE-STUDY APPLICATION.

| | LUTs | BRAM | DSPs | bitstream size (KB) |
|---|---|---|---|---|
| Sum_Vec | 1219 | 5 | 0 | 817 |
| | | | | 624 |
| | | | | 737 |
| Mult_Vec | 3162 | 0 | 8 | 922 |
| | | | | 872 |
| | | | | 877 |
| Static | 13817 | 45 | 0 | 4045 |

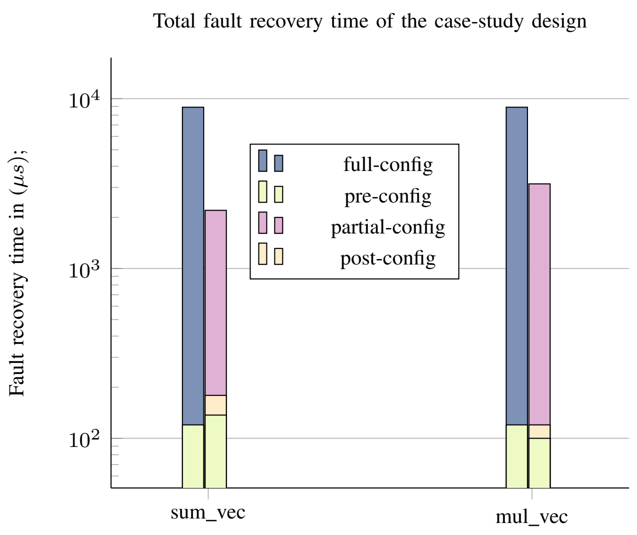

Fig. 8. Comparison of total fault recovery using a partial-reconfiguration vs full-reconfiguration approaches.

mode as well as partial reconfiguration. As expected, the full configuration time is much longer than the partial one (3.8x for *sum_vec* and 2.96x *mul_vec*). Thanks to OPEN-CFR, the reduction in recovery time allows us to build a system that is able to recover faster from SEUs.

Overall, our experiments demonstrate that OPEN-CFR delivers performance and resource consumption that are comparable to a commercial solution. It does so by using a fully open-source co-design flow that supports the automatic generation of the whole software/hardware system targeting commercial FPGA SoCs.

## V. DISCUSSION AND FUTURE WORKS

This work advances the state-of-the-art by proposing a baseline co-design framework for the generation of redundant applications on COTS FPGA SoCs. Our framework features a configurable, generated, RTL hardware shell for redundancy, supporting an integration with a mature DPR flow for recovery after fault detection, and providing a flexible software runtime. A summary of feature comparison between OPEN-CFR and the AMD-Xilinx commercial tool is reported in Table I.

We believe that the open-hardware nature of the framework can introduce interesting research directions – we hope to stimulate further research developments in the deployment of COTS FPGA SoCs in space applications. Next, we discuss some interesting avenues for future research that can be pursued leveraging OPEN-CFR.

*1) N-modular redundancy:* As the hardware shell for redundancy of OPEN-CFR is fully open source and generated by the framework, OPEN-CFR can be easily extended to support hardware N-modular redundancy by making slight modifications to the Pyverilog generation script. N-modular redundancy, combined with the DART DPR flow already integrated into OPEN-CFR, enables the exploration of advanced mechanisms for keeping the system operational and capable of solving fault(s) during the recovery phase of a replica and methodologies for synchronization of a recovered replica with the rest of the system at runtime. Given the closed-source nature of commercial tools, these explorations are generally prevented in commercial tools. This direction is interesting for strictly critical systems, in which the system must be kept operational during a recovery phase, and also enables the exploration of the functionalities/resource consumption tradeoffs and the scalability of N-modular redundancy systems in real COTS FPGA SoCs.

*2) Integration with OS kernels for real multi-tasking:* Another interesting direction is the integration of OPEN-CFR with an operative system, e.g. FreeRTOS [44] or the AMD-Xilinx Petalinux [45] distribution provided by AMD-Xilinx for their FPGA SoC platforms. This direction enables the integration of the configuration and user software associated with each redundant accelerator in software tasks, which can be, in turn, scheduled by the OS scheduler for supporting execution priorities and scheduling methodologies.

*3) Property-based verification:* The open-hardware nature of OPEN-CFR enables customization and analysis of the RTL of the redundant hardware shell. Thus, the correct functionalities of OPEN-CFR can be supported with the definition of a property-based verification [46], which can be used in complex scenarios to ensure that the implemented functionalities behave as expected and without weaknesses or bugs. This direction can provide an important additional level of assurance in strictly critical systems [47], [48].

*4) Worst-case mathematical timing analysis:* Strict timing requirements are typically mandated in hard real-time critical systems. Again, the full open-source nature of OPEN-CFR makes it possible to provide systematic timing analysis of the whole stack of the system, from hardware to software. An interesting direction is the combination of the traditional techniques for supporting software timing guarantees with the most recent mathematical approaches for fine-grained hardware timing analysis [49], [50]. This direction enables the combination of redundant execution with strong upper-bound timing guarantees for the execution, recovery, and synchronization of the system in hard real-time critical applications deployed in COTS FPGA SoCs.

## VI. Conclusion

This paper presents OPEN-CFR, an open-source co-design framework for redundancy with DPR support for fast and flexible recovery on COTS FPGA SoCs. We described the OPEN-CFR framework flow and internals and experimentally compared the features, performance, and resource consumption with a state-of-the-art tool providing IPs for implementing redundant execution on AMD-Xilinx platforms [13]. We believe that an open-hardware co-design framework for deploying redundant applications can open interesting research directions. In particular, we hope with this project to further stimulate the research in the use of COTS FPGA SoC platforms in space applications.

## Acknowledgments

This work was partially supported by the Department of Energy (A#: DE-SC0024458) and by the National Science Foundation (A#: 1764000).

## References

[1] M. Wirthlin, "High-reliability fpga-based systems: Space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015.

[2] N. Montealegre, D. Merodio, A. Fernandez, and P. Armbruster, "In-flight reconfigurable fpga-based space systems," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2015, pp. 1–8.

[3] L. Rockett, D. Patel, S. Danziger, B. Cronquist, and J. Wang, "Radiation hardened fpga technology for space applications," in *2007 IEEE Aerospace Conference*. IEEE, 2007, pp. 1–7.

[4] K. Varnavas, W. H. Sims, and J. Casas, "The use of field programmable gate arrays (fpga) in small satellite communication systems," in *International Conference on Advances in Satellite and Space Communications (SPACOMM 2015)*, no. M15-4394, 2015.

[5] A. S. Dawood, S. J. Visser, and J. A. Williams, "Reconfigurable fpgas for real time image processing in space," in *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No. 02TH8628)*, vol. 2. IEEE, 2002, pp. 845–848.

[6] *Zynq-7000 All Programmable SoC - Reference Manual*, Xilinx, 9 2016, uG585.

[7] *Zynq UltraScale+ Device - Reference Manual*, Xilinx, 12 2017, uG1085.

[8] *The Vitis AI official webpage*, AMD-Xilinx, https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[9] B. Seyoum, M. Pagani, A. Biondi, S. Balleri, and G. Buttazzo, "Spatio-temporal optimization of deep neural networks for reconfigurable fpga socs," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1988–2000, 2020.

[10] F. Restuccia and A. Biondi, "Time-predictable acceleration of deep neural networks on fpga soc platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 441–454.

[11] G. Eichler, B. Seyoum, K.-L. Chiu, and L. P. Carloni, "Mindcrypt: The brain as a random number generator for soc-based brain-computer interfaces," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 70–77.

[12] C. H. Le and L. R. Miles, "Challenges in fpga design for complex, high performance space applications," in *2023 IEEE Space Computing Conference (SCC)*. IEEE, 2023, pp. 45–50.

[13] *MicroBlaze Triple Modular Redundancy (TMR) Subsystem v1.0*, Xilinx-AMD, 2022, pG268.

[14] *The Synopsys Synplify official webpage*, Synopsys, https://www.synopsys.com/implementation-and-signoff/fpga-based-design/synplify.html.

[15] *The Open-CFR official repo*, https://github.com/alexredd99/open-cfr.

[16] H. Su, T. Lu, C. Feng, and L. Chen, "Triple module redundancy reliability framework design based on heterogeneous multi-core processor," *Procedia Computer Science*, vol. 183, pp. 504–511, 2021.

[17] H. D. Doran and T. Lang, "Dynamic lockstep processors for applications with functional safety relevance," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2021, pp. 1–4.

[18] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for sram-based fpgas," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 1290–1295.

[19] J. Lázaro, A. Astarloa, A. Zuloaga, J. Á. Araujo, and J. Jiménez, "Axi lite redundant on-chip bus interconnect for high reliability systems," *IEEE Transactions on Reliability*, 2023.

[20] B. Osterloh, H. Michalik, S. A. Habinc, and B. Fiethe, "Dynamic partial reconfiguration in space applications," in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, 2009, pp. 336–343.

[21] C. Pilotto, J. R. Azambuja, and F. L. Kastensmidt, "Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications," in *Proceedings of the 21st annual symposium on Integrated circuits and system design*, 2008, pp. 199–204.

[22] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "Fpga partial reconfiguration via configuration scrubbing," in *2009 International Conference on Field Programmable Logic and Applications*. IEEE, 2009, pp. 99–104.

[23] C. Bolchini, A. Miele, and M. D. Santambrogio, "Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*. IEEE, 2007, pp. 87–95.

[24] M. Straka, J. Kastil, and Z. Kotasek, "Modern fault tolerant architectures based on partial dynamic reconfiguration in fpgas," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE, 2010, pp. 173–176.

[25] M. Barbirotta, A. Cheikh, A. Mastrandrea, F. Menichelli, M. Ottavi, and M. Olivieri, "Evaluation of dynamic triple modular redundancy in an interleaved-multi-threading risc-v core," *Journal of Low Power Electronics and Applications*, vol. 13, no. 1, p. 2, 2022.

[26] M. Sarraseca, S. Alcaide, F. Fuentes, J. C. Rodriguez, F. Chang, I. Lasfar, R. Canal, F. J. Cazorla, and J. Abella, "Safels: An open source implementation of a lockstep noel-v risc-v core," in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2023, pp. 1–7.

[27] *The NOEL-V Official Webpage*, Gaisler, https://www.gaisler.com/index.php/products/processors/noel-v.

[28] M. Barbirotta, A. Cheikh, A. Mastrandrea, F. Menichelli, M. Angioli, S. Jamili, and M. Olivieri, "Fault-tolerant hardware acceleration for high-performance edge-computing nodes," *Electronics*, vol. 12, no. 17, p. 3574, 2023.

[29] A. E. Wilson and M. Wirthlin, "Fault injection of tmr open source risc-v processors using dynamic partial reconfiguration on sram-based fpgas," in *2021 IEEE Space Computing Conference (SCC)*. IEEE, 2021, pp. 1–8.

[30] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive fpga-based space computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 4, dec 2012. [Online]. Available: https://doi.org/10.1145/2392616.2392619

[31] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, "Tlegup: A tmr code generation tool for sram-based fpga applications using hls," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 129–132.

[32] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2, pp. 1–27, 2013.

[33] W. Xin, "Partitioning triple modular redundancy for single event upset mitigation in fpga," in *2010 International Conference on E-Product E-Service and E-Entertainment*. IEEE, 2010, pp. 1–4.

[34] S. Nema, J. Kirschner, D. Adak, S. Agarwal, B. Feinberg, A. F. Rodrigues, M. J. Marinella, and A. Awad, "Eris: Fault injection and tracking framework for reliability analysis of open-source hardware," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 210–220.

[35] S. Shreejith, K. Vipin, S. A. Fahmy, and M. Lukasiewycz, "An approach for redundancy in flexray networks using fpga partial reconfiguration," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 721–724.

[36] *AMBA® AXI™ and ACE™ Protocol Specification*, ARM, IHI 0022D.

[37] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, vol. 9040. Springer International Publishing, Apr 2015, pp. 451–460.

[38] B. Seyoum, M. Pagani, A. Biondi, and G. Buttazzo, "Automating the design flow under dynamic partial reconfiguration for hardware-software co-design in FPGA SoC," in *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 2021, pp. 481–490.

[39] *AXI CDMA, LogiCORE IP Product Guide, Document number PG034*, AMD-Xilinx.

[40] *The HLS4ML official documentation*, Fast Machine Learning Lab, https://fastmachinelearning.org/hls4ml/.

[41] M. Pierini, J. M. Duarte, N. Tran, and M. Freytsis, "Hls4ml lhc jet dataset (150 particles)," Jan. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3602260

[42] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. Di Guglielmo, P. Harris, J. Krupa *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.

[43] B. B. Seyoum, A. Biondi, and G. C. Buttazzo, "Flora: Floorplan optimizer for reconfigurable areas in fpgas," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3358202

[44] *The FreeRTOS official webpage*, FreeRTOS, https://www.freertos.org/index.html.

[45] *PetaLinux Tools Documentation Reference Guide*, AMD-Xilinx, uG1144.

[46] W. Hu, A. Ardeshiricham, M. S. Gobulukoglu, X. Wang, and R. Kastner, "Property specific information flow analysis for hardware security verification," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[47] F. Restuccia, A. Meza, R. Kastner, and J. Oberg, "A framework for design, verification, and management of soc access control systems," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 386–400, 2022.

[48] A. Meza, F. Restuccia, and R. Kastner, "Safety verification of third-party hardware modules via information flow tracking," in *1st Real-time And intelliGent Edge computing workshop (RAGE) co-located with the 2022 59th Design Automation Conference (DAC)*.

[49] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Bounding memory access times in multi-accelerator architectures on fpga socs," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 154–167, 2022.

[50] L. Valente, F. Restuccia, D. Rossi, R. Kastner, and L. Benini, "Top: Towards open & predictable heterogeneous socs," *arXiv preprint arXiv:2401.15639*, 2024.