An FPGA-based accelerator design methodology for smart UAVs in precision agriculture: a case study

Omitted for blind review

Abstract

Smart and Precision Agriculture (SPA) methods and technologies, such as autonomous robots, AI/ML, sensors, and actuators, enhance farming productivity by automating the retrieval of environmental parameters and the decision-making process, while Fog- and Edge-based paradigms enable more informed and responsive practices. Unmanned Aerial Vehicles (UAVs) can autonomously inspect crops and promptly cooperate with terrestrial vehicles to perform treatments, as recently demonstrated by the EU-funded COMP4DRONES (C4D) research project, focused on the provisioning of innovative UAV technologies for civilian applications. Modern companion-equipped UAV leverage Heterogeneous Systems-on-Chip (HeSoCs) to execute complex on-board tasks. HeSoCs generally combine a general-purpose, multi-core processor with a domain-specific accelerator-rich subsystem, massively integrating application-specific accelerators. Field Programmable Gate Array (FPGA)-based HeSoC are ideal fabrics to attain high performance and energy efficiency because of their massively parallel, deeply pipelined, non-Von-Neumann processing logic and custom memory hierarchies. Automated hardware-software co-design methodologies, e.g., FPGA overlays and toolflows, largely simplify the design phases, including the optimization of the accelerator interfaces, such as the merging of redundant components to reduce area usage. In this context, our contribution consists of a System-Level Design (SLD) methodology for the design of overlay-based UAV companion computers, including a modular and scalable accelerator-rich RISC-V HeSoC, a heterogeneous software stack and an automation toolchain to generate and integrate application-specific accelerators into our overlay. Our results show three optimized overlay variants targeting an UAV-based system employed in a SPA context. Experimental results denote improvements in performance and area usage, up to 18.5% on a FPGA-based HeSoC with respect to traditional design flows.

Keywords: UAV, Smart and Precision Agriculture, Companion Computer, FPGA Overlay, Accelerator-Rich Architecture

1. Introduction

Smart and Precision Agriculture (SPA) is revolutionizing the agricultural industry with innovative methods to improve operational efficiency, farming yield and crop health [1, 2]. These approaches largely evolved with the advent of Internet of Things (IoT) solutions, which allowed farmers to monitor many environmental parameters, perform analytics and observe the status of farms and machinery, including soil and crop health, usage of agrochemicals, fertilizers, water and seeds, livestock conditions, temperature and energy consumption, damage assessment caused by wildlife or other natural forces, and more [3, 4, 5]. This is achieved employing cutting-edge technologies—including autonomous robots, AI/ML algorithms, advanced sensors and actuators—to automate and enhance traditional human-centered farming activities [6, 7, 8, 9].

Moreover, agriculture demands spot-on remedies at exact space and time to reduce wastage of resources and ensure land and plant productivity [10, 11]. State-of-the-Art (SoA) computing paradigms—such as *Cloud*, *Fog* and *Edge*—are thus key to enable SPA [12]. These show large advantages compared to traditional *Human-Centered* or *Offline* practices, that generally postpone the application of treatments and/or farmer intervention with a negative impact on the crops' health status, costs and productivity. Indeed, *Offline* approaches include an initial on-field data collection and recordings, such as Light Detection and Ranging (LiDAR) point-clouds and multispectral images, and then the disjoint post-processing of raw measurements, which is performed by offline servers extracting insightful parameters and analyzing and classifying activities [13, 14, 15]. However, this implies significant delays in the decision-making process and a negative impact in the plantation foreseeable future [16]. *Cloud*-based solutions process data online on dedicated cloud servers. However, these retain major challenges, including privacy, scalability, high latency, network congestion and Quality of Service (QoS) degradation when huge amounts of heterogeneous farming data are transmitted [11, 17, 18]. Moreover, certain applications can suffer from the high energy consumption due to the data transfer to (and from) the cloud [19]. *Fog*- and *Edge*-based solutions commonly find similar definitions, as both are based on the concept of bringing cloud services and

resources closer to the edge data sources [12]. This is becoming computationally practical with the advent of modern computing platforms, thus enabling approaches which expedite data processing and enable more informed and responsive practices by moving the decision-making process closer to the crops [20, 21].

Autonomous Unmanned Aerial Vehicles (UAVs) will be pivotal in SPA activities, constituting reasonable human substitutes to examine the health status and growth evolution of the plantations, thus rapidly applying proper treatments [22]. The adoption of UAVs is mainly driven by the increased intelligence aboard and the advent of *companion-equipped drones* capable of executing complex tasks, e.g., real-time, data processing and analytics, or complex actuation [23, 24, 25, 26]. Another driver is the value generated by collected data, sampled closer to the crops thus guaranteeing more informed decisions [27]. Moreover, UAVs are expected to extensively cooperate with other robots, including Unmanned Ground Vehicles (UGVs), for synergistic and collaborative aerial and ground operations, aimed at enhancing operational outcomes and providing greater flexibility in adapting to dynamic environments [28, 29]. Given the heterogeneity of robots, the cooperation may happen in different ways to exploit their complementarities [30].

Besides, the EU's Common Agricultural Policy noted that greener farming practices will be fundamental to increase crop yield and meet the expected doubling of world food production by 2050, given no additional land will be available to meet these needs [31]. It is widely acknowledged that the future of farming relies on research, innovation, and capacity building in the agri-food sector leveraging on cutting-edge technologies, including advanced embedded systems, robotics, and digital platforms, which help to build a more sustainable and efficient food system. A recent example is provided by the EU-funded COMP4DRONES (C4D) research project[1] that focused on innovating UAV technologies for civilian applications [32]. One of the use-cases was specifically devoted to the improvement of the autonomous capabilities of UAV-based systems in the SPA context[2]. Meeting this requirement complies with the ad-hoc re-design of the drone system and its components, where the *Flight Controller (FC)* represents the most critical.

The FC is the core component of any UAV and several Commercial Off-

[1] `https://cordis.europa.eu/project/id/826610`

[2] `https://www.youtube.com/watch?v=lNkCDg6HjiQ`

The-Shelf (COTS) solutions are typically based on μ-controllers [33, 34]. Modern UAVs also integrate cutting-edge, *on-board companion computers*, delivering SoA computing, memory and network characteristics, which enable advanced on-board data processing and real-time decisions. Such companion computers commonly adopt embedded Heterogeneous Systems-on-Chip (HeSoCs), of which several COTS-based solutions are available [35]. These combine a general-purpose, multi-core processor with a domain-specific subsystem for advanced processing, e.g., a custom ASIC, GP-GPU or *Field Programmable Gate Array (FPGA)*.

In particular, massively integrating application-specific accelerators in a single FPGA-based HeSoC—following a so-called *accelerator-rich paradigm*—offer unique opportunities for autonomous UAVs and SPA applications [36, 37]. Indeed, such a companion computer can flexibly map and accelerate sophisticated workloads on its massively parallel, deeply pipelined, non-Von-Neumann processing logic and custom memory hierarchies, thus attaining high performance and low energy consumption, which is ideal for SPA applications [38, 39]. However, heterogeneity hardens the hardware-software co-design process associated with bringing up the companion computer, hence requiring approaches to mitigate design costs [40, 41].

For instance, *FPGA overlays* consist of Hardware (HW) abstraction layers that facilitate the mapping and scheduling of applications tasks to the FPGA fabric [42]. Besides, *toolflows* are also crucial for the optimization of the accelerators communication and control interfaces, including I/O data ports, register files, etc. In this context, Coarse-Grain Reconfigurable (CGR) accelerators represent a valuable option because they comprise low-overhead, decoupled, and inexpensive control interfaces. These features enable various optimization strategies, including merging redundant accelerator parts. This optimization would reduce area usage without additional overheads from complete, or partial, FPGA reconfiguration.

Under this premise, our contribution consists of a System-Level Design (SLD) methodology[3] for the design of overlay-based UAV companion computers, including:

- *On-board Overlay Compute Platform (OOCP)*, consisting of a modular and scalable accelerator-rich RISC-V HeSoC;

[3]Our contributions—OOCP, OODK and MDC—are open-source, as parts of the C4D research project.

- *On-board Overlay Development Kit (OODK)*, a heterogeneous Software (SW) stack to streamline the development of the heterogeneous application;
- *Multi-Dataflow Composer (MDC)*, an automation toolchain that allows automatic generation and integration of CGR accelerators inside our OOCP.

We show the results achieved with three alternative variants of the OOCP architecture. Experimental results were obtained by running the C4D use-case application on a COTS FPGA-based HeSoC from the AMD Zynq Ultrascale+ (US+) family. We demonstrate improvements in both performance and area usage of up to 18.5% compared to traditional integration flows such as [43], thanks to the merging and optimization features of our proposed flow.

The manuscript is organized as follows. Section 2 describes background content, including the C4D's project, the SPA use-case scenario and UAV components. Section 3 discusses the related works, while Section 4 introduces the components of the SLD methodology—OOCP, OODK and MDC. Then, in Section 5, we discuss the experimental results after the design of three variants of our overlay-based UAV companion computer, as well as an assessment of the methodology scalability for larger and more complex UAV systems. To demonstrate the broad applicability of the proposed approach, not limited to the SPA context, Section 6 describes additional projects enabled with the proposed approach.

2. Background

2.1. The COMP4DRONES project

The ECSEL JU project C4D—Framework of Key Enabling Technologies for Safe and Autonomous Drones—terminated in 2023 under the coordination of Indra and involvement of 50 partners. The Italian cluster included 12 partners and was led by the University of Sassari (UNISS), also involved in providing one of the assessment test case for the C4D technologies. The main goal of the project was to offer a framework of key enabling technologies for the design and operation of drones, ranging from application to electronic components, realized as a tightly integrated multi-vendor and compositional embedded architecture solution, and a toolchain to assemble and safely operate drones [32]. The main idea of the assessment test case from the Italian

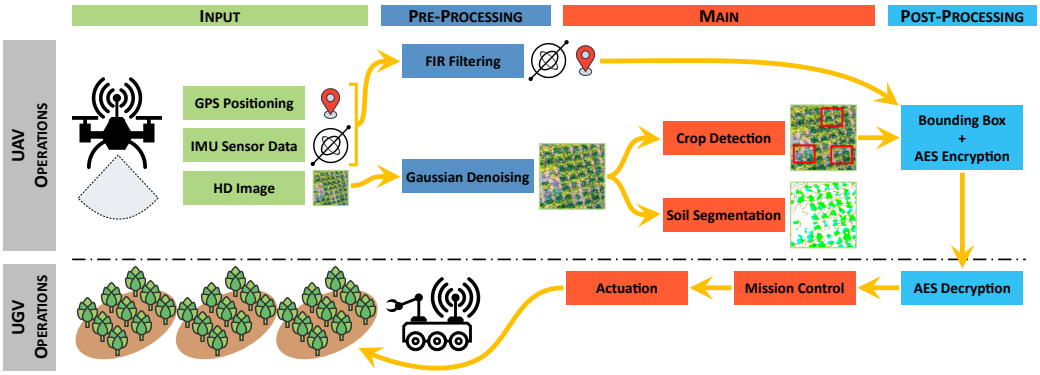


Figure 1: Representation of the C4D use-case application scenario.

cluster was to improve SPA technologies by providing more advanced observation and intervention methodologies through combined usage of a UAV and a UGV.

2.2. UAV for smart and precision agriculture

In the SPA context, UAV may handle many tasks, including the mapping and detection of crops and plants [44, 45, 46], monitoring sessions and inspection operations [47, 48, 49], disease detection [50, 51, 52] and application of treatments [53, 54]. Besides, flight-oriented data processing—such as path planning, obstacle detection and collision avoidance—are also crucial for safe environment exploration [55, 56, 57].

The C4D SPA use-case scenario embodies a subset of the tasks above and, specifically, the autonomous detection and monitoring of artichoke crops. The final goal consists of preserving a healthy yield status by monitoring the crop health status and searching for potential anomalies during the UAV's aerial traversal over a preset plantation route. During the flight, the drone examines the plants and searches for crop discontinuities. Besides, the UAV collaborates with a UGV in a synergistic effort for plant monitoring and health and growth management. Closing the loop faster brings advantages in the agriculture context, which largely benefits from fast reactions to anomalies, at exact space and time. In general, both the UAV and the UGV act as mobile sensors: the drone acts as a *sentinel* ensuring rapid reconnaissance and online decisions, while the terrestrial rover acts as a *mobile actuator*, meant for detailed recognition, spot-on spraying of nutrients, water and pesticides and support in case of anomalies or abnormal circumstances. Besides,

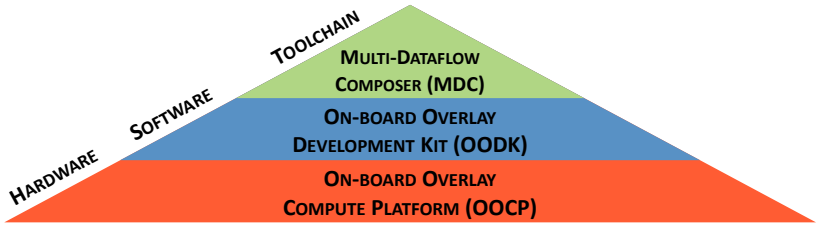


Figure 2: System-level design stack.

both extract and process complex plantation images to deduce relevant information and communicate over a trusted and secure medium, which is ensured with lightweight cryptographic modules for data-privacy, authentication and communication integrity.

Figure 1 shows the C4D use-case application scenario, where the UAV and UGV collaborate synergistically. In this manuscript, we focus our attention to the portion of the use-case executed by the UAV, given the UGV employs the same hardware-software technologies. Hence, the processing pipeline implemented by the UAV consists of: (i) INPUT DATA ACQUISITION; (ii) PRE-PROCESSING; (iii) MAIN; and (iv) POST-PROCESSING.

The UAV initially samples both *visual and text information*—including a 1440×900 HD image, IMU sensor data, and GPS position—and then forwards them to the companion computer, which initializes computing resources and starts executing the successive processing stages. As soon as the pre-processing phase starts, the field image is smoothed by applying *Gaussian Denoising* (`Conv`) to prune image noise for the subsequent processing steps, while IMU and GPS data encompass a *Finite Impulse Response* (`FIR`) filtering stage. Afterwards is the main processing phase, where *Crop Detection* and *Soil Segmentation* are executed. Crop detection is implemented by a *Convolutional Neural Network* (`CNN`) model for artichoke plant detection and classification through a customized Feature Pyramid Network CNN that works as a Single Shot Detector (SSD) [46]. Besides, image segmentation leverages at its heart a *Canny edge detector* (`Canny`) [58]. Finally, the post-processing stage packs textual data in a *Bounding BOX* (`BBOX`), which is subsequently encrypted with an *Advanced Encryption Standard* (`AES`) for trusted and secure communication toward the UGV.

2.3. UAV components

An UAV system comprises many components, including a multi-rotor or fixed-wing frame, Electronic Speed Control (ESC) modules, and a navigation system, relying on various sensors including Inertial Measurement Unit (IMU), barometers, Global Navigation Satellite System (GNSS), and radio transmitters [35]. Moreover, a crucial UAV component is the FC, of which many COTS alternatives are available [33]. The FC is generally based on μ-controllers and supports flexible I/O interfaces to facilitate the integration of additional sensors and mission-specific payloads, such as cameras and LiDARs. Besides, the FC handles various on-board functionalities, including flight control, navigation, data acquisition and processing, mission control and actuation [34].

Modern UAVs couple the FC with cutting-edge, *on-board companion computers* to empower the drone system with advanced processing capability, as well as SoA memory and network capability [16]. UAV companion computers typically adopt embedded HeSoCs, combining a general-purpose, multi-core processor with a domain-specific subsystem for advanced processing, such as a custom ASIC, GP-GPU or FPGA.

We believe that FPGA-based HeSoCs offer significant advantages for the design of complex UAV stacks, including the on-the-fly deployment of computing tasks and the availability of embedded soft-cores to simplify the HW/SW partitioning of the target application [38, 59]. However, existing tools are not mature enough to reduce the design strain to bring up a full-fledged FPGA-based companion computer and the integration of heterogeneous components and accelerators [40]. Thus, in the following sections, we propose an innovative SLD methodology, based on a FPGA overlay, to streamline the design and optimization of accelerator-rich HeSoCs.

3. Related work

3.1. About the design of UAV companion computers for SPA applications

This section compares existing literature on companion-equipped drones targeting SPA applications, including our work. In particular, our analysis focuses on works where the UAV companion computer enables autonomous and informed decisions on crops, safe navigation within plantations, and communication with other system components, including ground stations or other robots [26, 35, 28]. Table 1 summarizes the features of each approach.

Table 1: Overview of UAV companion computers for SPA applications.

	Technology			Main SPA Task	
	Platform	**Type**	**Accelerator**	**Task**	**Algorithm**
Alsalam et al. [60]	Odroid-U3+	CPU	—	Weed detection	Vision
Dai et al. [61]	Intel i5-6260	CPU	—	Pesticides spraying	Vision
Horstrand et al. [62]	NVIDIA Jetson TK1	HeSoC	GPU	Crop monitoring	VI[1]
Saddik et al. [63]	Intel Cyclone V	HeSoC	FPGA	Crop monitoring	VI[1]
This Work	**AMD Zynq US+**	**HeSoC**	FPGA	**Crop detection, Soil Segmentation**	**CNN, Edge Detection**

[1] VI: Vegetation Index

UAV companion computers can be designed using homogeneous platforms, where SPA tasks are executed entirely on general-purpose CPUs. *Alsalam et al.* use the Odroid-U3+ to detect invasive weeds and implement a decision-making control loop [60]. The companion computer runs vision algorithms and controls the UAV flight height. Other components handle path planning, environment inspection, and herbicide application. Although the Odroid-U3+ integrates an ARM Cortex-A9 CPU and a Mali-400 GPU, it is only used to a limited extent and GPU acceleration is neither utilized nor discussed. Similarly, *Dai et al.* present a vision-based UAV system for spraying pesticides on fruit trees [61]. The system employs an Intel i5-6260 CPU to process images, handle flight control, and communicate with a ground station. In contrast, our UAV system communicates with a UGV in a synergistic effort to improve response time to anomalies in the plantation. Moreover, our approach and other works advocate heterogeneous platforms to increase intelligence aboard UAVs by combining CPUs and accelerators, such as FPGAs and GPUs, for more complex SPA tasks.

Horstrand et al. developed an autonomous UAV for hyperspectral measurements that provides vegetation indices for vineyard analysis [62]. The onboard computer includes an NVIDIA Jetson TK1 for autonomous flight control, data acquisition, processing, and communication with a ground station. However, the main limitations of GPUs are the non-extendable HW components, non-deterministic execution, and high power consumption, often exceeding the availability of resource-constrained autonomous robots [38, 64].

To this end, FPGA-based HeSoCs are ideal solutions for UAV companion computers. *Saddik et al.* propose a vegetation monitoring system that accelerates GRVI and GLI indices using an Intel Cyclone V FPGA [63]. However, their Digital Signal Processor (DSP) utilization is low ($\text{DSP}_{\%,\text{ref}} = 6\%$),

Table 2: Overview of SLD methodologies for FPGA-based HeSoCs.

		Accelerator				Overlay					
		Type				Host		Proxy Core[2]		Interconnect[3]	
	TI[1]	HLS	RTL	CGR	System-Level Integration	RV64	AArch64	RV32	MicroBlaze	Cluster-level	HeSoC-level
PYNQ [67]		✓			✓		✓		✓	-	STREAM
RobotCore [68]	✓	✓			✓		✓			-	STREAM
OpenESP [69]	✓	✓	✓		✓	✓				-	NOC
TaPaSCo [70]	✓	✓	✓		✓		✓			XBAR	XBAR
This Work	✓	✓	✓	✓	✓	✓	✓	✓	✓	**XBAR**	**XBAR**

[1] TI: Technology-Independent.
[2] Note that both OpenESP and TaPaSCo support RV32, but they do not exploit it as a proxy core, i.e., for tightly-coupled runtime management and optimization of accelerator scheduling, memory transfers and other pivotal platform tasks, as discussed in Section 4.1.
[3] Interconnects are classified as: (i) XBAR: Crossbar; (ii) NOC: Network-on-Chip; and (iii) STREAM: Based on a streaming protocol.

whereas previous works show benefits of highly-pipelined DSP datapaths in FPGA-based accelerators [65, 66]. In contrast, our companion computer uses an accelerator-rich paradigm, where many application-specific accelerators are integrated to fully exploit the advantages of FPGA fabrics.

3.2. About SLD methodologies to simplify the design of FPGA-based HeSoCs

In this section, we examine various techniques for streamlining the design of FPGA-based HeSoCs. To this end, Table 2 summarizes characteristics of several industrial and academic approaches, including our own.

PYNQ is an open-source overlay design methodology that improves usability of AMD HeSoC products [67]. It is particularly suitable for non-FPGA experts, as it greatly simplifies the application development for embedded FPGA-based HeSoCs by offering support for Ubuntu Linux, Python, and Jupyter notebooks [71]. PYNQ overlays enable the deployment of accelerator-rich systems with support for dynamic partial reconfiguration [72]. However, beyond provided examples, users must still undergo traditional platform development using the AMD Vivado Suite. In contrast, our SLD methodology fully automates the customization of accelerator-rich overlays. Users need only supply accelerator sources, e.g., designed with High Level Synthesis (HLS), and Python specification files to define: (i) how accelerators are interfaced and optimized (e.g., through datapath merging), as described in Section 4.2.2; and (ii) how accelerators are aggregated at the system level

along with their respective optimizations, as detailed in Section 4.3.

The PYNQ IP portfolio includes all AMD IPs, including: (i) a 64-bit ARM host processor, supporting legacy SW stacks; (ii) a MicroBlaze soft core, a flexible and deterministic interface to a wide range of peripheral I/Os. In comparison, our OOCP features a broad portfolio of open-source IPs from the Parallel Ultra Low Power (PULP) Platform. Unlike PYNQ, our overlay is technology-independent and our design flow can be extended to support FPGAs from different vendors.

Finally, the PYNQ soft core communicates with the host processor via shared memory, but supports only basic communication libraries [73]. To enable more complex interactions, users must implement their own communication SW, requiring development for two processors. However, complex UAV applications demand robust communication mechanisms between the host and the FPGA subsystem. Moreover, accelerator-rich systems require flexible control and optimization of data transfers and accelerator management. To address these needs, our OODK supports the OpenMP programming model, which simplifies the development of complex heterogeneous software. With this model, users write a single application that covers host and proxy core. The heterogeneous application begins execution on the host, while compute-intensive tasks are offloaded to the accelerator-rich subsystem using a dedicated computation offloading command [74]. Our FPGA overlay thus provides built-in support for flexible communication between the host and the accelerator-rich subsystem. This interaction is managed by the proxy core, which acts as a man-in-the-middle to coordinate data movement and task execution across subsystems.

RobotCore targets Robot Operating System (ROS) 2 computational graphs for robotic applications [68]. The approach can accelerate compute nodes and ROS 2 communication channels to avoid expensive interactions with the host processor. However, RobotCore's FPGA subsystem is hardwired, making it less flexible than our programmable FPGA overlay.

OpenESP consists of a SLD framework for the seamless hardware-software integration of application-specific accelerators into HeSoCs [69]. Unlike our proposal, OpenESP uses the FPGA fabric as an emulation platform for ASIC prototyping and integrates coarse-grained accelerators that are loosely coupled with the processor. In contrast, our accelerator-rich overlay comprises numerous fine-grained accelerators, which are grouped into clusters and then integrated at the system level—for example, to form more sophisticated accelerator-rich pipelines. Additionally, our approach differs from OpenESP

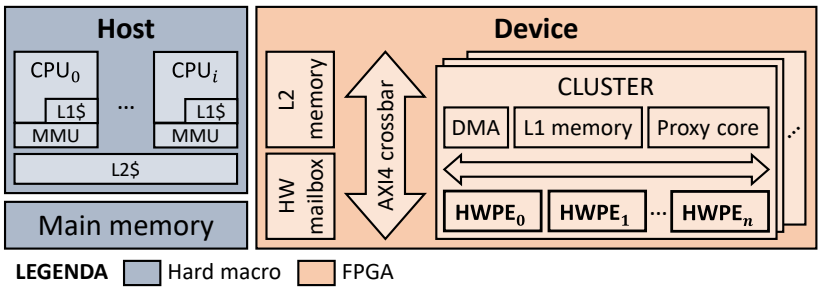
in terms of SW support. While ESP interfaces accelerators using a Linux driver, OOCP connects accelerators directly within the heterogeneous application or through user-space libraries. This is enabled by our OODK, which provides full support for OpenMP [75].

Task Parallel System Composer (TaPaSCo) is an open-source methodology designed to simplify the development of accelerator-rich platforms, supporting many FPGA fabrics—from embedded devices to High Performance Computing (HPC) systems [70]. In TaPaSCo, HLS accelerators of the same type are grouped into homogeneous clusters. Heterogeneous platforms are then built by interconnecting different clusters. In contrast, our approach introduces heterogeneity directly at the level of OOCP clusters, which integrate heterogeneous, fine-grained accelerators. This architectural choice is fundamental to efficiently supporting accelerator-rich optimizations, including: (i) area-oriented techniques, such as datapath merging enabled by MDC; and (ii) performance-oriented strategies, such as the use of a proxy core for double-buffering and memory latency hiding, as discussed in Section 6.

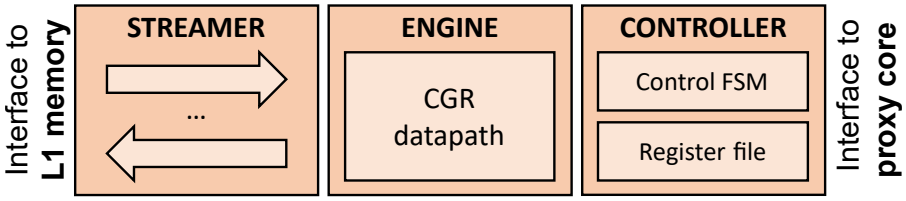
TaPaSCo also supports RISC-V soft cores, used as processing elements for many-core platforms. Our proxy core is also RISC-V compliant but differs from the TaPaSCo soft core in intent. Indeed, the proxy core is a key enabler of high-performance, accelerator-rich platforms by orchestrating their many, co-existing tasks. This approach is flexible because it is SW-defined. This also includes the task scheduling of fine-grained accelerators, with automatic generation of SW libraries and Application Programming Interfaces (APIs). Furthermore, it is low-cost since the proxy core is tightly-coupled to the components of the OOCP cluster, including accelerators. This supervisory role of the proxy core does not limitat the SW developer, who can still choose to deploy computation on the proxy core, as demonstrated in Section 6.2.

4. Methodology

This section describes our SLD methodology, shown in Figure 2 and including: OOCP(i) OOCP, our FPGA overlay consisting of a modular and scalable accelerator-rich RISC-V HeSoC; (ii) OODK, our heterogeneous SW stack to streamline the development of the heterogeneous application; and (iii) MDC, our automation toolchain that allows automatic generation and integration of CGR accelerators inside OOCP. We start by describing these components and the complete workflow that enables the design of overlay-based UAV companion computers.



(a) HeSoC architecture deployed on Zynq US+ devices.



(b) HWPE-based wrapper.

Figure 3: Overview of the OOCP components.

4.1. Overlay architecture

Our FPGA overlay consists of two components: (i) *OOCP*, a modular and scalable accelerator-rich RISC-V HeSoC [43]; and (ii) *OODK*, a heterogeneous SW stack to streamline the development of the heterogeneous application. The architectural template of the overlay builds on top of the PULP Platform, an open and scalable hardware-software research and development platform for highly parallel, ultra-low-power architectures, based on the open-source *RISC-V* Instruction Set Architecture (ISA) [76].

The OOCP architecture sticks to a standard HeSoC with two independent subsystems—*host* and *device*—as shown in Figure 3a. The *host* is a Linux-capable multi-core application processor. If the target HeSoC fabric supports a physical host processor, then it can be leveraged with its HW/SW legacy. For instance, this is the case for the Zynq US+ family of devices, where an AArch64 application processor is integrated [77, 78]. Alternatively, our architectural template also supports the OpenHW Group CVA6 based on the RV64 ISA, which can implemented in FPGA [79].

The *device* subsystem consists of an accelerator-rich subsystem, including *clusters* of processing elements sharing a multi-banked SW-programmable L2

Scratchpad Memory (SPM) via a fully-connected AXI4 CrossBAR (XBAR). The communication between the host-device subsystems happens via coarse-grained computational offloading, which leverages the off-chip main memory and a HW mailbox for synchronization.

The *cluster* integrates *accelerators* and additional support components, including a *proxy core*, L1 memory—implemented as a multi-banked Tightly-Coupled Data Memory (TCDM)—and a SW-programmable Direct Memory Access (DMA) engine for efficient bi-directional 64-bit data transfers between the L2 and L1 memory hierarchies. L1 memory accesses occur through the cluster Low-Latency Interconnect (LIC).

The *proxy core* is a pivotal component meant for tightly-coupled run-time management of various cluster tasks, such as the orchestration of the accelerator tasks, DMA bandwidth monitoring and control operations and many more [80, 81]. It consist of a low-cost soft-core, implementing the RV32 ISA and fetching instructions from the L2 memory through its instruction cache [82, 83]. This approach is more flexible compared to a dedicated Finite State Machine (FSM) and more scalable and tightly-coupled compared to a costly invocation of the host processor for the same intent. Moreover, the proxy core implements a flexible and customizable host-device interface.

Accelerators leverage Hardware Processing Engine (HWPE) interfaces to ensure trustworthy and flexible communication between their datapaths and the cluster components [84, 85]. HWPE-based accelerators are integrated inside the OOCP cluster and tightly-coupled to the L1 memory.

Figure 3b shows the key modules of a HWPE wrapper: (i) an *engine*, implementing the accelerator datapath; (ii) a *streamer*, providing an I/O data interface to L1 memory; and (iii) a *controller*, for coarse-grained accelerator control/(re)configuration. The use of streaming interfaces decouples the CGR datapath from the outer system, with benefits in terms of latency tolerance. Besides, the adoption of the HWPE wrapper increases the MDC merging flexibility. Indeed, a typical concern of merging approaches is that they cannot usually cope with the merging of kernels with different communication protocols because of incompatibilities at the I/O interface. With the adoption of HWPE, more uneven CGR datapaths can be merged in a single accelerator wrapper, thus integrated into our accelerator-rich OOCP.

The OODK consists of a heterogeneous SW stack supporting the application developer. OODK supports the OpenMP programming model to facilitate the development of complex heterogeneous SW, as well as a SoA LLVM-based heterogeneous compiler, which allows for the development of

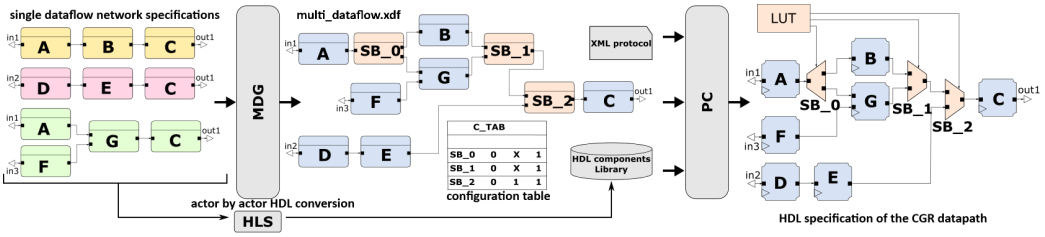


Figure 4: MDC: From dataflow specification to the generation of a CGR datapath.

single-source single-binary heterogeneous applications with OpenMP 4.5 offloading. The heterogeneous application always starts on the host processor and then critical tasks can be transparently offloaded to the accelerator-rich subsystem, simply taking advantage of OpenMP pragmas.

4.2. Automatic generation of CGR accelerators

The automated design and integration of application-specific accelerators is accomplished by *MDC*, an open-source automated tool for the generation of CGR accelerators and their integration inside OOCP [86, 87].

Generally speaking, according to the typical MDC flow, the generated HW/SW components consist of:

- *accelerator datapaths*, merging different functionalities in a single accelerator exploiting the principles of CGR reconfigurability;
- *accelerator wrappers*, implementing communication/control interfaces meant for the interaction of the CGR accelerator datapaths with other HW components that, in typical usage so far, were indended as communication/control interface toward the processing system;
- *accelerator drivers*, to simplify and abstract the configuration and management of the CGR accelerator implemented within the wrapper.

4.2.1. CGR datapath generation

The generation of CGR datapaths is in charge of the baseline MDC functionality, which is composed of a front-end and a back-end. The front-end is called Multi-Dataflow Generator (MDG) and is responsible of datapath merging; whereas the back-end is called Platform Composer (PC), and is responsible of translating a high-level merged dataflow model into an Hardware Description Language (HDL) specification.

The MDG mandates a *dataflow specification*, which comprises a set of tasks—or functions— to be accelerated defined as dataflow networks, that are directed graphs connecting actors through First-In First-Out queues (FIFOs). In the example of Figure 4, three tasks are represented in the top-left corner. The MDG block analyzes the input networks, locates the shared actors—A and C are shared, in this specific case—and generates a *multi-dataflow network* combining all the functionalities. The combination is done by generating only one instance of each shared actor and including a set of switching boxes (SBoxes). These SBoxes allow the selection of the path that the data will follow when selecting a specific functionality to be executed. To do so, a *configuration table* gathering the specific values to configure the SBoxes is also automatically generated.

As a second step, the PC derives the HDL specification of the CGR datapath from the multi-dataflow network, requiring: (i) the XML definition of the communication protocol between actors; and (ii) the HDL description of the actors, e.g., SystemVerilog, Verilog or VHDL, contained in the *HDL components library*. This latter can be manually generated or automatically sythesized leveraging HLS techniques, e.g. Vitis HLS.

4.2.2. Automatic integration inside the FPGA overlay

An automatic procedure also occurs for generating accelerator wrappers, which are necessary to integrate CGR accelerators within our OOCP.

MDC has been extended to support the HWPE interface to ensure trustworthy and flexible communication between accelerators and the OOCP components. HWPE-based accelerators are integrated inside the OOCP cluster and tightly-coupled to the L1 memory.

The Co-Processor Generator (CPG) functionality of the MDC tool, shown in Figure 5, currently supports a novel feature to automatically generate the HWPE-compliant wrapper. Through this feature, the CPG became capable of encapsulating the PC outcome, the CGR datapath of a reconfigurable accelerator, with the necessary glue logic for integrating it inside the OOCP cluster. Moreover, as part of the new feature, MDC CPG also takes care of generating the necessary accelerator drivers to invoke the accelerator from the heterogeneous application, developed via the OODK.

To this end, the tool requires a *wrapper specification* along with the *CGR datapath specification*. The former describes the characteristics of the integrated CGR datapath for customizing the HWPE streamer and controller.

Multiple CGR datapaths cannot run concurrently when merged in the

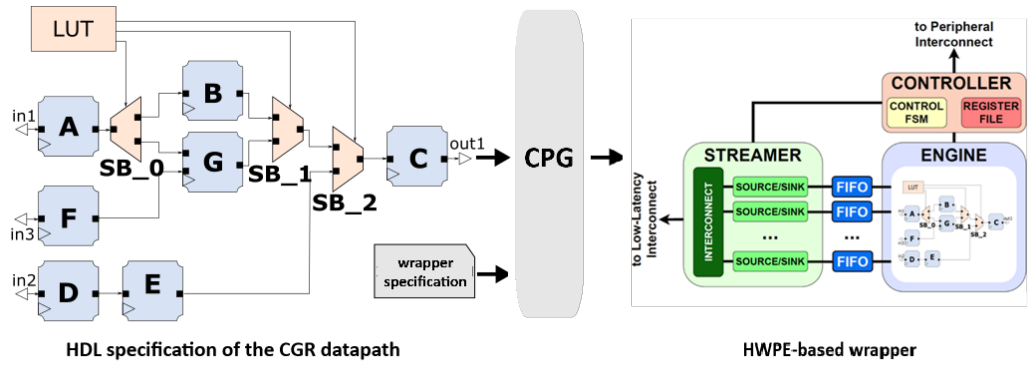


Figure 5: MDC: From multi-dataflow specification to the generation of the HWPE wrapper.

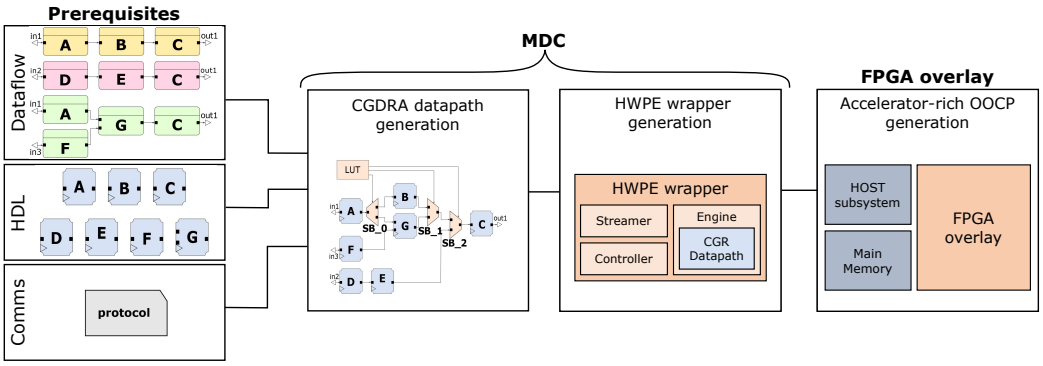


Figure 6: SLD methodology to design full-fledged UAV companion computers based on an accelerator-rich FPGA overlay.

same HWPE wrapper. Yet, to switch functionality requires a single clock cycle, given task selection occurs via the simple SW configuration of a HWPE register. Besides, the area usage of a multi-functional accelerator, where several tasks co-exist in the same wrapper, is minimized thanks to the merging process of actors, I/O resources and register files.

4.3. SLD methodology for UAV companion computers

The combination of our FPGA overlay with the MDC framework results in the SLD methodology, shown in Figure 6. This workflow is meant to streamline the design, optimization and deployment of accelerator-rich UAV companion computers. The flow offers an abstract HW/SW co-design inter-

```
class oocp_specs:

    def OOCP(self):
        self.name      = Accelerator-rich platform
        self.target    = FPGA fabric
        self.l2_mem    = [Number of ports, Size]
        return self

    def cluster_0(self):
        self.acc       = [Accelerator name, ...]
        self.proxy     = [IP, Number of cores, ...]
        self.dma       = [IP, Job queue size, ...]
        self.l1_mem    = [Number of ports, Size]
        return self

    ...

    def cluster_N(self)
        ...
```

Listing 1: Example of OOCP specification.

face to improve ease-of-use and supports datapath merging and automatic accelerator integration in an accelerator-rich context.

As prerequisites for using our SLD methodology, the user defines the application requirements, taking note of the application task(-s) to accelerate, leveraging the presented dataflow approach and providing the input specifications as *dataflow networks*. Moreover, the user must provide a library of actors, described as *HDL specifications*, and the *communication* protocol specifying their data exchange protocol.

At this point, MDC automatically generates the *CGR datapaths*, merging the input tasks' specification (as detailed in Section 4.2.1). Subsequently, MDC also generates the HWPE wrappers and related drivers (as detailed in Section 4.2.2). Please note that the merging process is completely under the control of the user. Indeed, the user should select the set of input dataflow specifications to be accelerated, and pass them in input to MDC. In this work, we opted for two different strategies to select the set of tasks to be merged within the HWPE, *application-aware*, that takes into consideration the application task scheduling, and *specification-aware*, taking into account the I/O characteristics of the input networks.

The integration of generated HWPE-based accelerators into the OOCP and the UAV companion computer is performed using a *toolchain* included in the OODK. A *OOCP specification file* defines the key attributes of the

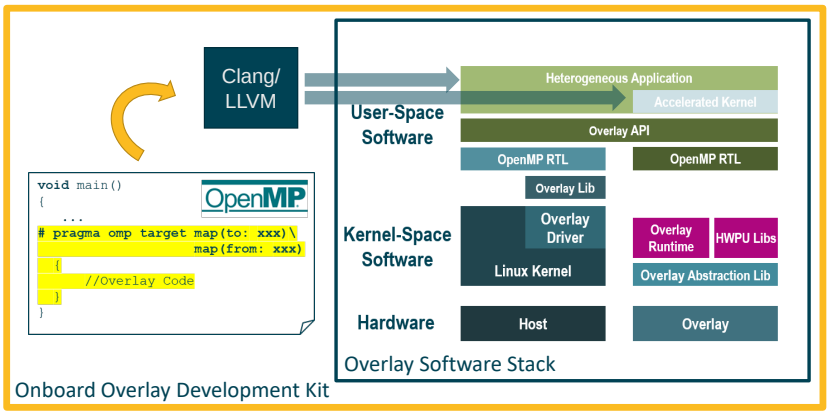


Figure 7: OODK software compilation flow using OpenMP runtime API.

accelerator-rich platform. These include the characteristics of the OOCP and *cluster* fabrics, as well as the distribution of accelerators across the platform. Listing 1 provides an example of the `oocp_specs` class. The `OOCP` (rows 3-7 of Listing 1) method specifies the `name` and `target` for the FPGA fabric. `l2_mem` defines instead size and number of ports for L2 memory. The `cluster` (rows 9-19 of Listing 1) method specifies the attributes of the *cluster fabrics*. An arbitrary number of heterogeneous clusters can be defined, from `cluster_0` to `cluster_N`. The `acc` object specifies which accelerators to integrate within an *cluster*. Additionally, the values assigned to `proxy`, `dma`, and `l1_mem` enable specialization of other *cluster* components. Additional parameters specify the number of *proxies* and allow flexible configuration of the DMA design. The L1 memory is similarly defined in terms of its size and number of ports. Based on these specifications, an accelerator-rich platform is automatically generated, with system-level integration of accelerators using the generated HWPE. Finally, an FPGA bitstream is built using vendor-dependent tools (e.g., AMD Vivado Suite) to implement a specific *overlay* instance and complete the SLD design process.

Users can execute code on the overlay through a high-level and established programming model such as OpenMP. Figure 7 shows the compilation flow included in the OODK. By using the OpenMP accelerator model and target directives, users can offload portions of execution from the host processor to the overlay. Consequently, the accelerator APIs can be used to access accelerated functionalities. This high-level programming model proves to be much more flexible and agile compared to custom, low-level, vendor-specific

Table 3: FPGA resource availability of the AMD-Xilinx ZU9EG HeSoC fabric.

Resource Type	**Availability**
Look-Up Tables (LUTs)	274080
Flip Flops (FFs)	548160
Block RAMs (BRAMs)	912
Digital Signal Processors (DSPs)	2520

models used in overlays like PYNQ and ESP.

5. Experimental Evaluation

In this section, our methodology is leveraged to design a UAV companion computer for the C4D use-case application scenario. Initially, we characterize the C4D use-case tasks and inspect those traits that are exploited at merge-time. Then, we propose three variants of our FPGA overlay, each integrating the C4D tasks; then, we profile performance and area usage to select the most suitable architecture at serving the SPA application.

5.1. Experimental setup

We implement OOCP on an AMD ZCU102 board, which integrates a Zynq US+ ZU9EG HeSoC. Table 3 reports the resource availability of the FPGA fabric, including: (i) Look-Up Tables (LUTs), i.e., reconfigurable digital circuits to implement combinational logic functions; (ii) Flip Flops (FFs), to save logical states, e.g., for processor and accelerator registers; (iii) Block RAMs (BRAMs), that can be cascaded to create large and fast memory arrays, e.g., SPMs and FIFOs; and (iv) DSPs (Digital Signal Processors), i.e., dedicated DSP slices.

Since the ZU9EG HeSoC includes a physical ARM Cortex-A running at $f_{Host} = 1.2GHz$, we leverage it as a host processor. Besides, we deploy the accelerator-rich subsystem to the FPGA fabric at a target frequency $f_{FPGA} = 100MHz$. The FPGA design is synthesized, implemented and deployed leveraging the AMD Vivado Suite, including: (i) Vivado HLS v2018.2 to design the HW-accelerated use-case kernels; (ii) Vivado v2019.2 to synthesize and implement the FPGA design; and (iii) Petalinux v2019.2 to boot the OOCP. Experimental reports discussed in the following sections include runtime performance measurements and FPGA resource utilization, retrieved after the implementation phase.

Table 4: Overview of the C4D use-case kernels.

	Task	Kernel	Description	I/O
Pre Processing	***Low-pass filter***	`FIR`	Fully-parallel systolic implementation, configurable with 64 or 128 coefficients.	1/1
	Blur	`Conv`	Dataflow implementation of a Gaussian Convolution with internal stages optimized in a pipeline manner.	1/1
Main	***Crop detection***	`CNN`	CNN Accelerator, configurable in different modes to accommodate variegated convolutional layers. The implementation is based on previous C4D partners work [46].	2/1
	Soil segmentation	`Canny`	Canny edge detector, based on a Sobel implementation [58].	2/1
Post Processing	***Data annotation***	`BBOX`	Packing of textual metadata with image and IMU data.	3/1
	Data encryption	`AES`	AES data encryption.	1/1

5.2. Overview of the use-case SPA kernels

Acquiring details about the features of the SPA kernels is essential to guide the merge process, which is then by different strategies, e.g., the *application task scheduling* and the characteristics of the *I/O accelerator interfaces*.

Table 4 resumes the tasks composing the C4D's SPA use-case scenario, along with a brief functional description. The Pre-Processing and Main phases are HW-accelerated in the accelerator-rich subsystem, while the Post-Processing is run in SW on the host processor. Notably, a companion computer targeting SPA applications is called to perform multiple accelerated functions, during the mission. Moreover, it is worth pointing out that MDC supports different design approaches, which have been employed to develop these kernels. In particular, the CAPH language is used to develop the `Canny` [88], while HLS C/C++ is leveraged to design `FIR`, `Conv` and `CNN`. Finally, the last column, on the right, shows the number of I/O interfaces for each kernel implementation. In the case of a HW accelerator, this consists of the actual I/O ports exposed at the interface of the CGR datapath, while for SW implementations, it consists of the number of I/O data arrays.

Figure 8 shows the latency of execution of the SPA kernels. Each bar

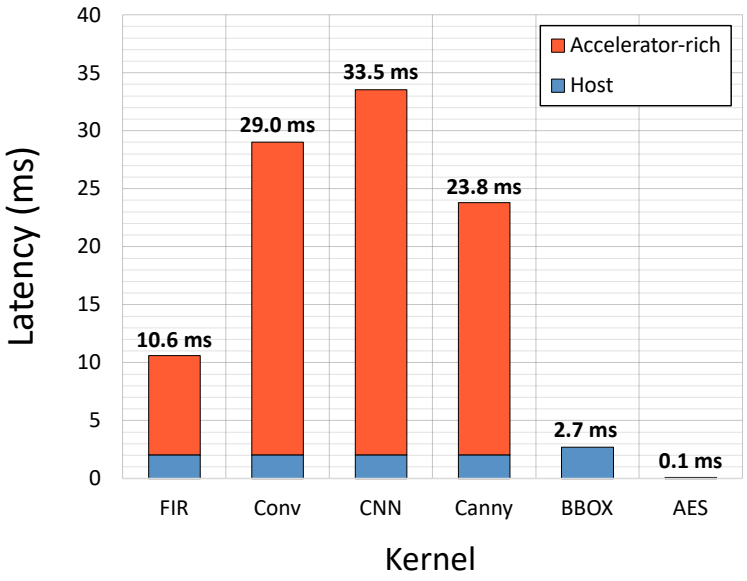


Figure 8: Latency profiling of the use-case kernels.

expresses a latency breakdown of the executions on: (i) the host, including startup, allocation and offloading; and (ii) the accelerator-rich subsystem, including the proxy core startup, DMA data transfers and accelerator execution.

5.3. Assembling an overlay-based UAV companion computer

We proceed by assembling *three variants* of our overlay-based UAV companion computer, both leveraging the proposed methodology, as well as the acquired information concerning the SPA kernels. The variants are meant to fit the requirements of the C4D's SPA use-case, described in Section 2, and *differ by the adopted merge strategies*, as shown in Table 5: (i) the BASELINE instantiates a dedicated HWPE wrapper for each CGR datapath; (ii) the SCENARIO-AGNOSTIC strategy groups CGR datapaths by the *number of input accelerator ports*, thus merging `Conv` and `FIR`, as well as `CNN` and `Canny`; and (iii) the SCENARIO-AWARE strategy is driven by the *application task scheduling*, where the goal is to maximize the number of parallel tasks. For instance, `CNN` and `Canny` are instantiated in dedicated wrappers to run in parallel, as they retain no mutual dependency and are very latency-expensive. Besides, `Conv` and `FIR` are merged, given the former has a dependency on

Table 5: Overview of the adopted merge strategies.

Strategy	**Implementation**[1]
Baseline	$[\text{FIR}]_a$, $[\text{Conv}]_b$, $[\text{CNN}]_c$, $[\text{Canny}]_d$
Scenario-Agnostic	$[\text{FIR} + \text{Conv}]_a$, $[\text{CNN} + \text{Canny}]_b$
Scenario-Aware	$[\text{FIR} + \text{Conv}]_a$, $[\text{CNN}]_b$, $[\text{Canny}]_c$

[1] SPA kernels are either instantiated in dedicated wrappers $[\text{kernel}]_i$ or merged in a single wrapper $[\sum \text{kernels}]_i$.

both `CNN` and `Canny`; indeed, the execution of `Canny` can be hidden by running it in parallel to the other kernels.

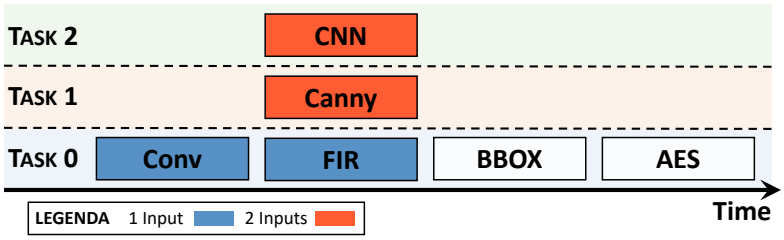
The SW application starts on the host processor, which acquires data from sensors and then offloads the Pre-Processing and Main stages to the accelerator-rich subsystem. Once offloading completes, the RISC-V proxy core commands the DMA to move input data to local accelerator buffers. Data are divided into tiles due to the limited dimension of the L1 memory. As soon as the transfer completes, accelerators are launched, according to the execution profile of the C4D application. Figure 9a consists of a time diagram corresponding to the Scenario-Aware strategy, which guides the accelerator optimization given the information about the application task scheduling, as well as the Baseline, where no merging constraints are applied to accelerators. Differently, the Scenario-Agnostic strategy forces `CNN` and `Canny` to run sequentially, being merged in the same wrapper. The diagram also includes colour-coded features that describe the number of I/O interfaces of the accelerators. `BBOX` and `AES` are not colour-coded, as they are executed in SW on the host processor.

5.3.1. Latency

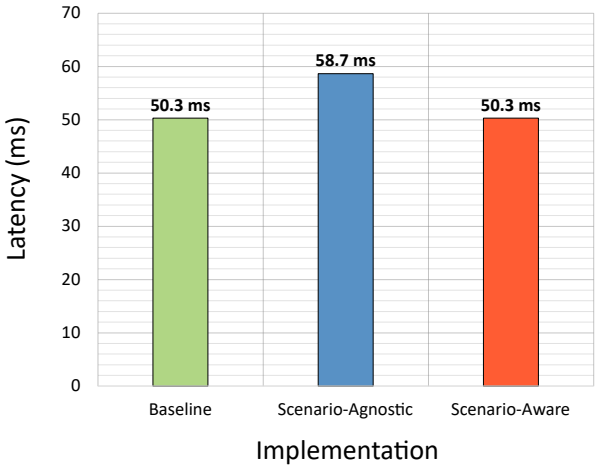
Figure 9b shows the resulting execution latency for each variant. The Scenario-Aware approach terminates in 50.3ms, while the Scenario-Agnostic strategy degrades of a factor of 13.7%, as of the serialization of the `CNN` and `Canny` tasks. Additionally, the Baseline does not constrain kernels by merging their interfaces; thus, each can operate freely in parallel, following the application's profile. As a consequence, the former performs as the Scenario-Aware approach.

5.3.2. Area usage

Table 6 shows the FPGA resource utilization of the Baseline variant, normalized on the resource availability of the ZU9EG HeSoC fabric. Values



(a) Application task scheduling



(b) Execution latency

Figure 9: Application task scheduling and execution latency concerning the three architectural variants of the overlay-based UAV companion computer.

are organized following three hierarchical levels: (i)*overlay-level*, referring to an OOCP instance with a single cluster, the host communication interface, i.e., the HW mailbox, and the L2 memory subsystem, accounting for resource usage of the AXI4 interconnect and the L2 SPM; (ii) *cluster-level*, with a RISC-V proxy core, DMA, L1 memory and the cumulative occupation of accelerator wrappers and datapaths; and (iii) *accelerator-level*, including specific information concerning the HWPE wrappers and CGR datapaths.

6

The main benefit of the merge process consists of the area savings originating from the sharing of the HWPE wrapper, thus we expect a reduction of their area usage with a higher employment of the optimization. With a reduction of the wrappers, fewer L1 memory ports are required, thus positively affecting the occupation of the cluster LIC (cumulated with the L1 memory).

Table 6: FPGA utilization of the baseline C4D OOCP variant.

Overlay-level		**LUT%**	**FF%**	**BRAM%**	**DSP%**
	Host communication	2.0	0.7	0	0
	L2 memory	5.0	1.4	7.0	0
	Cluster	59.7	23.8	17.4	36.9
Cluster-level		**LUT%**	**FF%**	**BRAM%**	**DSP%**
	RISC-V soft-core	8.6	1.5	9.6	0.4
	DMA	2.3	1.1	0	0
	L1 memory	11.1	3.7	2.9	0
	Accelerator wrappers	7.6	2.9	0	0
	Accelerator datapaths	30.3	14.7	4.9	36.6
Accelerator-level		**LUT%**	**FF%**	**BRAM%**	**DSP%**
FIR	Streamer	0.8	0.2	0	0
	Controller	0.4	0.4	0	0
	Datapath	8.6	8.5	0	5.1
Conv	Streamer	1.7	0.4	0	0
	Controller	0.9	0.4	0	0
	Datapath	1.6	0.8	2.4	0.5
CNN	Streamer	1.2	0.4	0	0
	Controller	0.6	0.4	0	0
	Datapath	10.5	1.6	0.1	30.5
Canny	Streamer	1.4	0.3	0	0
	Controller	0.6	0.4	0	0
	Datapath	9.6	3.8	2.4	0.5

NOTE: Normalized FPGA utilization (%) refers to the resource availability of the AMD-Xilinx ZU9EG HeSoC fabric used in our experiments.

Thus, LUT would decrease of up to $\Delta_{\text{LUT,max}} = -18.7\%$, consisting of about 28% of the BASELINE overlay. Besides, FF decrease is expected for up to $\Delta_{\text{FF,max}} = -6.6\%$, hence 26% of the BASELINE overlay.

Table 7 shows the occupation results of each merging strategy, to confirm these conjectures. Indeed, at the overlay-level, we denote a maximum decrement in resource occupation of $\Delta_{\text{LUT}} = -12.7\%$ and $\Delta_{\text{FF}} = -4.4\%$, in correspondence with the SCENARIO-AGNOSTIC approach. No variation applies to BRAMs and DSPs because the merge does not apply to the CGR datapaths or other overlay components making use of those macros. Zooming in at the cluster-level, the same decrement is found, as the merge only influences the cluster components: the accelerator wrappers and the cluster LIC.

Table 7: FPGA utilization of the C4D OOCP variants.

Overlay-level	**LUT%**	**FF%**	**BRAM%**	**DSP%**
Baseline	66.7	25.9	24.4	36.9
Scenario-Agnostic	54.0	21.5	24.4	36.9
Scenario-Aware	55.7	22.3	24.4	36.9
Cluster-level	**LUT%**	**FF%**	**BRAM%**	**DSP%**
Baseline	59.7	23.8	17.4	36.9
Scenario-Agnostic	46.9	19.4	17.4	36.9
Scenario-Aware	48.7	20.2	17.4	36.9
Accelerator-level	**LUT%**	**FF%**	**BRAM%**	**DSP%**
Baseline	37.8	17.5	4.9	36.5
Scenario-Agnostic	34.9	16.1	4.9	36.6
Scenario-Aware	35.2	16.7	4.9	36.6

Note: Normalized FPGA utilization (%) refers to the resource availability of the AMD-Xilinx ZU9EG HeSoC fabric used in our experiments.

5.3.3. Performance and area trade-off

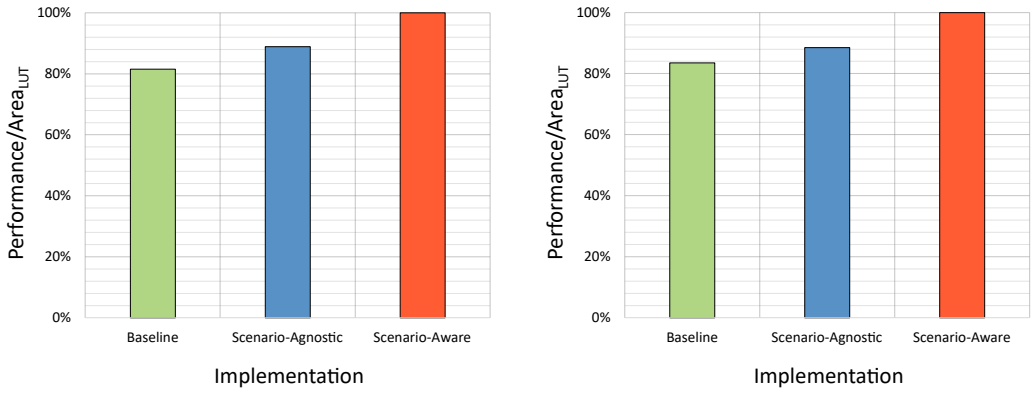
We dig deeper into the exploration of the three variants and investigate aggregated trade-offs, derived from performance and FPGA resource utilization—LUT and FF usage. The implementation that better responds to trade-offs is determined by the larger value of the aggregated metric, which is normalized on the results of the Scenario-Aware strategy

Figures 10a and 10b shows the trade-off between performance and LUT occupation. The Scenario-Aware strategy better performs than the Baseline OOCP by a factor of 16.5% and 18.5%, respectively at overlay- and cluster-level. Even though the Scenario-Agnostic worsen performance compared to the Baseline—as described in Section 5.3.1—then it still prevails on the Baseline by factors of 5.0% and 7.4%. Besides, it is also worth noting the magnification of the metric when zooming inside the cluster, again explainable with the merge only applying to the cluster components.

Figures 10c and 10d shows instead the performance and FF occupation trade-off. In this case, the Scenario-Aware approach improves the Baseline OOCP by a factor of 14.0% and 15.2%—at overlay- and cluster-level—while the Scenario-Agnostic by factors of 2.7% and 7.3%.

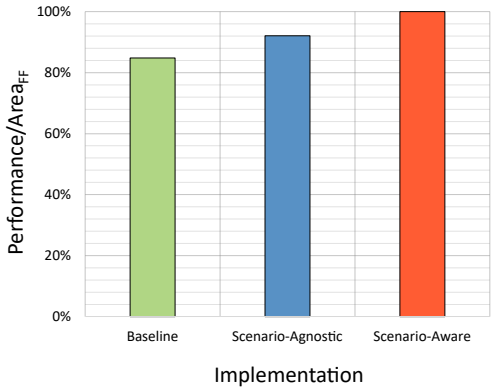
5.3.4. Scalability analysis

This experiment assesses the scalability of the proposed methodology for larger UAV companion computers. We assume that a total of $N_{acc} = 16$ accel-

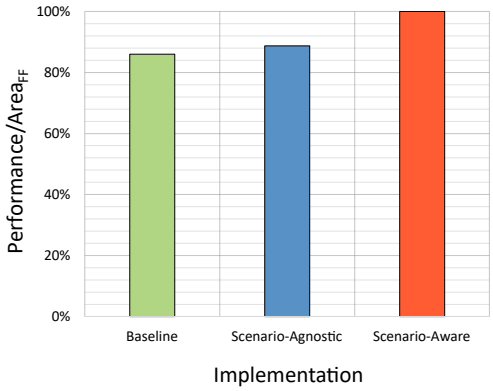


(a) Overlay-level

(b) Cluster-level



(c) Overlay-level



(d) Cluster-level

Figure 10: Trade-off between performance and FPGA resource utilization (LUT, FF).

erators is sufficient to gain meaningful insights, as commercial HeSoCs tend to integrate low-tens of accelerators into shared-memory clusters [89, 90, 91]. Further scaling introduces challenges due to single-cluster limitations, such as contention and interference among shared components. A widely adopted solution is to aggregate accelerators into multi-cluster, accelerator-rich architectures. This approach reduces pressure on shared cluster components but incurs additional FPGA resource overhead, thereby necessitating an investigation into the scalability of our overlay.

To this end, we assemble five multi-cluster OOCP variants using our SLD methodology. Each variant includes N_{acc} accelerators that are evenly distributed across up to 16 clusters. Additionally, we scale the L2 memory

Table 8: Scalability analysis of the proposed FPGA overlay.

			LUT		FF		BRAM		DSP	
Overlay[1]	$\mathbf{N_{C_L}}$[2]	$\mathbf{N_{L2P}}$[3]	**Abs**[4]	**Rel**[5]	**Abs**	**Rel**	**Abs**	**Rel**	**Abs**	**Rel**
1CL	1	1	89495	1.0	56395	1.0	184	1.0	9	1.0
2CL	2	2	129342	1.45	98755	1.75	368	2.0	18	2.0
4CL	4	4	222986	2.49	173135	3.07	736	4.0	36	4.0
8CL	8	8	407514	4.55	314644	5.58	1472	8.0	72	8.0
16CL	16	16	899865	10.05	590244	10.47	2944	16.0	144	16.0

[1] **Overlay**: Multi-cluster variants of the FPGA overlay.
[2] $\mathbf{N_{C_L}}$: Number of clusters.
[3] $\mathbf{N_{L2P}}$: Number of L2 memory ports.
[4] **Abs**: Absolute FPGA utilization.
[5] **Rel**: Relative FPGA utilization normalized to the absolute value of the 1CL variant.

subsystem according to the number of clusters (N_{C_L}), i.e., we expose memory ports and banks (N_{L2P}) enabling each cluster to access data and instructions independently. We scale N_{C_L} and N_{L2P} simultaneously. The largest variant is 16CL, with each accelerator integrated into a dedicated cluster.

Table 8 reports absolute and relative values from the experiment. Relative values are normalized to the value of the 1CL variant to gain insights into multi-cluster scalability relative to a baseline, i.e., the single-cluster instance of our FPGA overlay. These values represent the overlay implementation costs, excluding any application-specific accelerator datapath or wrapper. In principle, BRAMs and DSPs scale poorly, exposing a linear trend correlated with N_{C_L} because of the additional L2 memory ports and proxy cores, i.e., 1 per cluster. In contrast, LUTs and FFs scale a bit better — around 10× when 16CL are instantiated. Re-aggregating accelerators into multiple clusters reduces the cost per cluster, as fewer LIC accelerator ports are exposed, and some components, such as those dedicated to host communication, are not replicated when scaling, as previously shown in Table 6. However, a critical contributor to LUTs and FFs is the fully-connected AXI4 XBAR at HeSoC-level. To this end, it is worth noting that we are working on extending the IP portfolio of our FPGA overlay with a Network-on-Chip (NoC) architecture to enhance the scaling capability of our methodology [92].

The former discussion provides insights into the implementation cost of an empty overlay, i.e., the "skeleton" of a multi-cluster, accelerator-rich HeSoC. We conclude by presenting an additional analysis that integrates the previous findings. Figure 11 shows five example accelerators from the literature, including: (i) `DPU-B512`, AMD Deep Learning Processor Unit (DPU) for Convolutional Neural Network (CNN) acceleration (B512 configuration) [93];

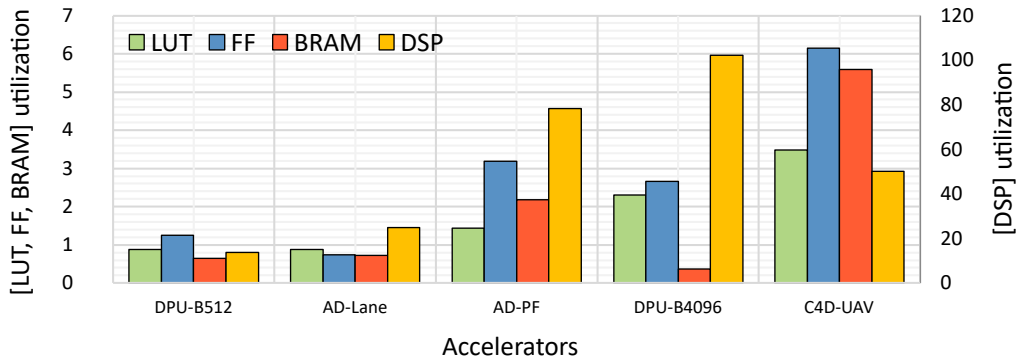


Figure 11: Examples of five accelerators from the literature, including: (i) `DPU-B512`, AMD DPU for CNN acceleration (B512 configuration) [93]; (ii) `AD-Lane`, CV accelerator-rich pipeline for lane detection [94]; (iii) `AD-PF`, PF to accelerate the localization component in a perception-plan-act AD stack [80]; (iv) `DPU-B4096`, AMD DPU for CNN acceleration (B4096 configuration) [93]; and (v) **C4D-UAV**, our accelerator-rich use-case. We analyze the FPGA resource usage of each accelerator, normalized over a single cluster overlay resource consumption. Results are presented using two vertical axes: the *left axis* for LUTs, FFs, BRAMs, and the *right axis* for DSPs.

(ii) `AD-Lane`, Computer Vision (CV) accelerator-rich pipeline for lane detection [94]; (iii) `AD-PF`, Particle Filter (PF) to accelerate the localization component in a perception-plan-act Autonomous Driving (AD) stack [80]; (iv) `DPU-B4096`, AMD DPU for CNN acceleration (B4096 configuration) [93]; and (v) **C4D-UAV**, our UAV companion computer for SPA applications implementing the SCENARIO-AWARE strategy, described in Section 5.3. As a side note, more details concerning `AD-Lane` and `AD-PF` are also found in Section 6.

We evaluate the FPGA resource usage of each accelerator, normalized to the resource consumption of a single cluster overlay.[4] This normalization enables a meaningful comparison of the FPGA resources required to implement real-world accelerators, including those from our accelerator-rich use-case scenario. The results intuitively reveal how efficiently resources are utilized: higher normalized values suggest more effective use of FPGA resources by the accelerator itself, while lower values imply greater overhead introduced by the overlay. However, it is important to emphasize that the

[4]A normalized value of 1.0 indicates that the accelerator uses the same amount of resources as the overlay; a value of 2.0 indicates twice the resource usage.

overlay cluster should not be regarded as overhead per se. As discussed in Section 4.1, the cluster and its components are fundamental to building high-performance, accelerator-rich systems [95, 36, 37].

The results are presented using two vertical axes: the left axis shows LUT, FF, and BRAM utilization, while the right axis displays DSP usage. This dual-axis format improves readability, as the normalized values for these two groups of FPGA resources differ significantly in scale.

It can be observed that, for simple accelerators and pipelines such as AccDpuSmall and `AD-Lane`, the resource overhead introduced by the overlay is comparable, and in some cases equivalent ($\approx 1\times$), to that of individual application-specific datapaths. This indicates that, even in fine-grained scenarios, the cost associated with the overlay is not prohibitive and does not hinder its practical adoption.

More notably, in more complex configurations—such as `DPU-B4096`, `AD-PF`, and our **C4D-UAV**—the resource impact of the overlay becomes virtually negligible, with overheads ranging from slightly above $2\times$ to up to $6\times$ across all resource classes.

6. Discussion

In this section, we discuss further application scenarios enabled with our SLD methodology. These demonstrate its broader applicability, not limited to the SPA context. In particular, we focus on two AD use-cases concerning F1TENTH autonomous racing competitions [96].

F1TENTH is an open-source platform for autonomous racing with 1:10 scaled cars for safe and rapid experimentation of AD systems and algorithms. Given their small size, these cars mount very constrained computing platforms, thus mandating deep AD stack optimization to attain target Key Performance Indicators (KPIs), e.g., performance, area, and energy consumption.

These examples demonstrate that our FPGA overlay, with its automation flow and proxy core availability, streamlines the integration of application-specific accelerators, as well as the HW/SW partitioning and optimization of complex applications deployed to FPGA-based HeSoCs.

6.1. AD-Lane: A computer vision pipeline for lane detection

The first use-case concerns `AD-Lane`, a CV pipeline for *lane detection* that autonomously identifies road lane markings and guarantees that the

AD system follows the correct path [94].

Using the same methodology applied to the UAV companion computer, we examine the pipeline stages and accelerate the most critical stages with the AMD Vitis Vision HLS Library [97]. Non-accelerated tasks are executed in SW, given the limited capability of the AMD Kria KV260 board. We integrate `AD-Lane` using our FPGA overlay and explore the benefits of the following HW/SW optimizations: (i) the *proxy core* improves the performance of accelerators with many input parameters with respect to looser and costly interactions with the host processor; (ii) the *L1 memory* benefits memory-bound stages, given that a tightly-coupled accelerator SPM provides lower access latency and better predictability compared to the off-chip main memory; and (iii) *pipelining* the memory and compute phases according to a double buffering memory scheme, orchestrated by the proxy core, improves overall performance compared to sequential execution. Ultimately, we achieve a 22× speedup, compared to a SW-only implementation.

6.2. AD-PF: Real-time LiDAR-based autonomous localization

The second use-case concerns `AD-PF`, a typical perception-plan-act AD stack, where we accelerate the localization component, a Monte-Carlo method named *PF* [80]. We leverage the PF to localize the car in a pre-built map using perception data from a LiDAR sensor, while the optimal car trajectory is computed offline and the control loop is closed with a pure pursuit algorithm. As the application concerns racing vehicles, HW/SW optimizations aim to maximize the average speed on single laps and peak speed in head-to-head situations. Experiments are conducted on real-life race tracks with two boards (AMD ZCU102, Avnet Ultra96). We examine the impact of the single pipeline stages and accelerate the most critical one, i.e., the *Ray Marching (RM)* stage that takes up to 90% of the overall PF execution. Hence, we design a *Ray Marching Engine* (`RME`) with HLS, that achieves a 2.5× speedup compared to a pure SW implementation. Performance is further improved by a factor of $2-3\times$ integrating the `RME` within our FPGA overlay and exploiting the *proxy core* to optimize the memory transfers between the HeSoC subsystems—host and device.

7. Conclusions

In this work, we have contributed an innovative SLD methodology for the design of overlay-based UAV companion computers. In particular, the

methodology tackles the challenge of facilitating the assembling of full-fledged UAV companion computers, targeting *SPA* applications, as is one of the use-case scenarios of the EU-funded C4D research project.

The workflow consists of a modular and scalable accelerator-rich RISC-V HeSoC, a heterogeneous SW stack to streamline the development of the heterogeneous application and a toolflow to automatically generate and integrate application-specific CGR accelerators inside the FPGA overlay.

We show the results of extensive benchmarking, based on three alternative variants of the OOCP architecture. Experimental results are retrieved by executing the C4D use-case application on a COTS FPGA-based HeSoC, belonging to the AMD Zynq US+ family. These demonstrate improvements in performance and area usage, up to 18.5%.

The proposed methodology not only demonstrates the integration capabilities of accelerator-rich systems, but also paves the way for new research directions. First, the architectural template can be further extended with additional IPs to achieve better scalability in increasingly accelerator-dense environments. In this regard, we are already working on expanding the IP portfolio of our overlay with FlooNoC[5] [92]. A second promising direction is to investigate and integrate automated design space exploration methods for optimal selection of integration and merging strategies for accelerators.

References

[1] R. Gebbers, V. I. Adamchuk, Precision agriculture and food security, Science 327 (5967) (2010) 828–831.

[2] R. Bongiovanni, J. Lowenberg-DeBoer, Precision agriculture and sustainability, Precision agriculture 5 (2004) 359–387.

[3] A. Monteiro, S. Santos, P. Gonçalves, Precision agriculture for crop and livestock farming—brief review, Animals 11 (8) (2021) 2345.

[4] H. Azadi, S. M. Moghaddam, S. Burkart, H. Mahmoudi, S. Van Passel, A. Kurban, D. Lopez-Carr, Rethinking resilient agriculture: From climate-smart agriculture to vulnerable-smart agriculture, Journal of Cleaner Production 319 (2021) 128602.

[5] `https://github.com/pulp-platform/FlooNoC`

[5] N. Ahmed, D. De, I. Hussain, Internet of things (iot) for smart precision agriculture and farming in rural areas, IEEE internet of things journal 5 (6) (2018) 4890–4899.

[6] T. A. Shaikh, T. Rasool, F. R. Lone, Towards leveraging the role of machine learning and artificial intelligence in precision agriculture and smart farming, Computers and Electronics in Agriculture 198 (2022) 107119.

[7] H. Yin, Y. Cao, B. Marelli, X. Zeng, A. J. Mason, C. Cao, Soil sensors and plant wearables for smart and precision agriculture, Advanced Materials 33 (20) (2021) 2007764.

[8] D. Xie, L. Chen, L. Liu, L. Chen, H. Wang, Actuators and sensors for application in agricultural robots: A review, Machines 10 (10) (2022) 913.

[9] P. K. R. Maddikunta, S. Hakak, M. Alazab, S. Bhattacharya, T. R. Gadekallu, W. Z. Khan, Q.-V. Pham, Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges, IEEE Sensors Journal 21 (16) (2021) 17608–17619.

[10] E. Karunathilake, A. T. Le, S. Heo, Y. S. Chung, S. Mansoor, The path to smart farming: Innovations and opportunities in precision agriculture, Agriculture 13 (8) (2023) 1593.

[11] J. Liu, J. Xiang, Y. Jin, R. Liu, J. Yan, L. Wang, Boost precision agriculture with unmanned aerial vehicle remote sensing and edge intelligence: A survey, Remote Sensing 13 (21) (2021) 4387.

[12] Y. Kalyani, R. Collier, A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture, Sensors 21 (17) (2021) 5922.

[13] M. P. Christiansen, M. S. Laursen, R. N. Jørgensen, S. Skovsen, R. Gislum, Designing and testing a uav mapping system for agricultural field surveying, Sensors 17 (12) (2017) 2703.

[14] P. Katsigiannis, L. Misopolinos, V. Liakopoulos, T. K. Alexandridis, G. Zalidis, An autonomous multi-sensor uav system for reduced-input

precision agriculture applications, in: 2016 24th Mediterranean Conference on Control and Automation (MED), IEEE, 2016, pp. 60–64.

[15] J. Primicerio, S. F. Di Gennaro, E. Fiorillo, L. Genesio, E. Lugato, A. Matese, F. P. Vaccari, A flexible unmanned aerial vehicle for precision agriculture, Precision Agriculture 13 (4) (2012) 517–523.

[16] D. Madroñal, F. Palumbo, A. Capotondi, A. Marongiu, Unmanned vehicles in smart farming: A survey and a glance at future horizons, in: Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings, 2021, pp. 1–8.

[17] Y. Liu, X. Ma, L. Shu, G. P. Hancke, A. M. Abu-Mahfouz, From industry 4.0 to agriculture 4.0: Current status, enabling technologies, and research challenges, IEEE transactions on industrial informatics 17 (6) (2020) 4322–4334.

[18] J. Chen, X. Ran, Deep learning with edge computing: A review, Proceedings of the IEEE 107 (8) (2019) 1655–1674.

[19] H. A. Alharbi, M. Aldossary, Energy-efficient edge-fog-cloud architecture for iot-based smart agriculture environment, Ieee Access 9 (2021) 110480–110492.

[20] J. Su, X. Zhu, S. Li, W.-H. Chen, Ai meets uavs: A survey on ai empowered uav perception systems for precision agriculture, Neurocomputing 518 (2023) 242–270.

[21] P. P. Ray, A review on tinyml: State-of-the-art and prospects, Journal of King Saud University-Computer and Information Sciences 34 (4) (2022) 1595–1623.

[22] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, I. Moscholios, A compilation of uav applications for precision agriculture, Computer Networks 172 (2020) 107148.

[23] L. Valente, A. Nadalini, A. Veeran, M. Sinigaglia, B. Sa, N. Wistoff, Y. Tortorella, S. Benatti, R. Psiakis, A. Kulmala, et al., A heterogeneous risc-v based soc for secure nano-uav navigation, arXiv preprint arXiv:2401.03531.

[24] A. Di Mauro, M. Scherer, D. Rossi, L. Benini, Kraken: A direct event/frame-based multi-sensor fusion soc for ultra-efficient visual processing in nano-uavs, arXiv preprint arXiv:2209.01065.

[25] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, et al., Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight, Science robotics 7 (67) (2022) eabl6259.

[26] A. Douklias, L. Karagiannidis, F. Misichroni, A. Amditis, Design and implementation of a uav-based airborne computing platform for computer vision and machine learning applications, Sensors 22 (5) (2022) 2049.

[27] D. C. Tsouros, S. Bibi, P. G. Sarigiannidis, A review on uav-based applications for precision agriculture, Information 10 (11) (2019) 349.

[28] I. Munasinghe, A. Perera, R. C. Deo, A comprehensive review of uav-ugv collaboration: Advancements and challenges, Journal of Sensor and Actuator Networks 13 (6) (2024) 81.

[29] M. Mammarella, L. Comba, A. Biglia, F. Dabbene, P. Gay, Cooperation of unmanned systems for agricultural applications: A case study in a vineyard, biosystems engineering 223 (2022) 81–102.

[30] J. Chen, X. Zhang, B. Xin, H. Fang, Coordination between unmanned aerial and ground vehicles: A taxonomy and optimization perspective, IEEE transactions on cybernetics 46 (4) (2015) 959–972.

[31] S. J. Undertaking, et al., European drones outlook study: unlocking the value for europe.

[32] R. Nouacer, M. Hussein, H. Espinoza, Y. Ouhammou, M. Ladeira, R. Castiñeira, Towards a framework of key technologies for drones, Microprocessors and Microsystems 77 (2020) 103142.

[33] N. Aliane, A survey of open-source uav autopilots, Electronics 13 (23) (2024) 4785.

[34] E. Ebeid, M. Skriver, J. Jin, A survey on open-source flight control platforms of unmanned aerial vehicle, in: 2017 euromicro conference on digital system design (dsd), IEEE, 2017, pp. 396–402.

[35] A. Wilson, A. Kumar, A. Jha, L. R. Cenkeramaddi, Embedded sensors, communication technologies, computing platforms and machine learning for uavs: A review, IEEE Sensors Journal 22 (3) (2021) 1807–1826.

[36] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, G. Reinman, Accelerator-rich architectures: Opportunities and progresses, in: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, 2014, pp. 1–6.

[37] M. J. Lyons, M. Hempstead, G.-Y. Wei, D. Brooks, The accelerator store: A shared memory framework for accelerator-based systems, ACM Transactions on Architecture and Code Optimization (TACO) 8 (4) (2012) 1–22.

[38] Z. Wan, A. Lele, B. Yu, S. Liu, Y. Wang, V. J. Reddi, C. Hao, A. Raychowdhury, Robotic computing on fpgas: Current progress, research challenges, and opportunities, arXiv preprint arXiv:2205.07149.

[39] Y. Chi, W. Qiao, A. Sohrabizadeh, J. Wang, J. Cong, Democratizing domain-specific computing, Communications of the ACM 66 (1) (2022) 74–85.

[40] L. P. Carloni, The case for embedded scalable platforms, in: Proceedings of the 53rd Annual Design Automation Conference, 2016, pp. 1–6.

[41] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, et al., Chipyard: Integrated design, simulation, and implementation framework for custom socs, IEEE Micro 40 (4) (2020) 10–21.

[42] M. H. Quraishi, E. B. Tavakoli, F. Ren, A survey of system architectures and techniques for fpga virtualization, IEEE Transactions on Parallel and Distributed Systems 32 (9) (2021) 2216–2230.

[43] G. Bellocchi, A. Capotondi, F. Conti, A. Marongiu, A risc-v-based fpga overlay to simplify embedded accelerator deployment, in: 2021 24th Euromicro Conference on Digital System Design (DSD), IEEE, 2021, pp. 9–17.

[44] A. Lambertini, E. Mandanici, M. A. Tini, L. Vittuari, Technical challenges for multi-temporal and multi-sensor image processing surveyed

by uav for mapping and monitoring in precision agriculture, Remote Sensing 14 (19) (2022) 4954.

[45] C. Chen, Z. Zheng, T. Xu, S. Guo, S. Feng, W. Yao, Y. Lan, Yolo-based uav technology: A review of the research and its applications, Drones 7 (3) (2023) 190.

[46] A. Sassu, J. Motta, A. Deidda, L. Ghiani, A. Carlevaro, G. Garibotto, F. Gambella, Artichoke deep learning detection network for site-specific agrochemicals uas spraying, Computers and Electronics in Agriculture 213 (2023) 108185.

[47] F. A. Almalki, B. O. Soufiene, S. H. Alsamhi, H. Sakli, A low-cost platform for environmental smart farming monitoring system based on iot and uavs, Sustainability 13 (11) (2021) 5908.

[48] C. Qu, J. Boubin, D. Gafurov, J. Zhou, N. Aloysius, H. Nguyen, P. Calyam, Uav swarms in smart agriculture: Experiences and opportunities, in: 2022 IEEE 18th International Conference on e-Science (e-Science), IEEE, 2022, pp. 148–158.

[49] A. I. de Castro, Y. Shi, J. M. Maja, J. M. Peña, Uavs for vegetation monitoring: Overview and recent scientific contributions, Remote Sensing 13 (11) (2021) 2139.

[50] A. Abbas, Z. Zhang, H. Zheng, M. M. Alami, A. F. Alrefaei, Q. Abbas, S. A. H. Naqvi, M. J. Rao, W. F. Mosa, Q. Abbas, et al., Drones in plant disease assessment, efficient monitoring, and detection: a way forward to smart agriculture, Agronomy 13 (6) (2023) 1524.

[51] N. Kitpo, M. Inoue, Early rice disease detection and position mapping system using drone and iot architecture, in: 2018 12th South East Asian Technical University Consortium (SEATUC), Vol. 1, IEEE, 2018, pp. 1–5.

[52] J. Su, C. Liu, M. Coombes, X. Hu, C. Wang, X. Xu, Q. Li, L. Guo, W.-H. Chen, Wheat yellow rust monitoring by learning from multispectral uav aerial imagery, Computers and electronics in agriculture 155 (2018) 157–166.

[53] T. Talaviya, D. Shah, N. Patel, H. Yagnik, M. Shah, Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides, Artificial Intelligence in Agriculture 4 (2020) 58–73.

[54] B. S. Faiçal, H. Freitas, P. H. Gomes, L. Y. Mano, G. Pessin, A. C. de Carvalho, B. Krishnamachari, J. Ueyama, An adaptive approach for uav-based pesticide spraying in dynamic environments, Computers and Electronics in Agriculture 138 (2017) 210–223.

[55] D. Debnath, F. Vanegas, J. Sandino, A. F. Hawary, F. Gonzalez, A review of uav path-planning algorithms and obstacle avoidance methods for remote sensing applications, Remote Sensing 16 (21) (2024) 4019.

[56] S. Ahmed, B. Qiu, F. Ahmad, C.-W. Kong, H. Xin, A state-of-the-art analysis of obstacle avoidance methods from the perspective of an agricultural sprayer uav's operation scenario, Agronomy 11 (6) (2021) 1069.

[57] L. Wang, Y. Lan, Y. Zhang, H. Zhang, M. N. Tahir, S. Ou, X. Liu, P. Chen, Applications and prospects of agricultural unmanned aerial vehicle obstacle avoidance technology in china, Sensors 19 (3) (2019) 642.

[58] C.-L. Sotiropoulou, C. Gentsos, S. Nikolaidis, A. Rjoub, Fpga-based canny edge detection for real-time applications, in: published at the 26th Conference on Design of Circuits and Integrated Systems (DCIS), Albufeira, Portugal, 2011.

[59] M. Bouhali, F. Shamani, Z. E. Dahmane, A. Belaidi, J. Nurmi, Fpga applications in unmanned aerial vehicles-a review, in: Applied Reconfigurable Computing: 13th International Symposium, ARC 2017, Delft, The Netherlands, April 3-7, 2017, Proceedings 13, Springer, 2017, pp. 217–228.

[60] B. H. Y. Alsalam, K. Morton, D. Campbell, F. Gonzalez, Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture, in: 2017 IEEE Aerospace Conference, IEEE, 2017, pp. 1–12.

[61] B. Dai, Y. He, F. Gu, L. Yang, J. Han, W. Xu, A vision-based autonomous aerial spray system for precision agriculture, in: 2017 ieee international conference on robotics and biomimetics (ROBIO), IEEE, 2017, pp. 507–513.

[62] P. Horstrand, R. Guerra, A. Rodríguez, M. Díaz, S. López, J. F. López, A uav platform based on a hyperspectral sensor for image capturing and on-board processing, IEEE Access 7 (2019) 66919–66938.

[63] A. Saddik, R. Latif, A. El Ouardi, Low-power fpga architecture based monitoring applications in precision agriculture, Journal of Low Power Electronics and Applications 11 (4) (2021) 39.

[64] N. H. Malle, F. F. Nyboe, E. S. M. Ebeid, Onboard powerline perception system for uavs using mmwave radar and fpga-accelerated vision, IEEE Access 10 (2022) 113543–113559.

[65] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, A survey of fpga-based neural network accelerator, arXiv preprint arXiv:1712.08934.

[66] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, S. A. Fahmy, Deco: A dsp block based fpga accelerator overlay with low overhead interconnect, in: 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2016, pp. 1–8.

[67] PYNQ - Python productivity for Zynq.
URL `http://www.pynq.io/`

[68] V. Mayoral-Vilches, S. M. Neuman, B. Plancher, V. J. Reddi, Robotcore: An open architecture for hardware acceleration in ros 2, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2022, pp. 9692–9699.

[69] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, L. P. Carloni, Accelerator integration for open-source soc design, IEEE Micro 41 (4) (2021) 8–14.

[70] C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, A. Koch, The tapasco open-source toolflow: For the automated composition of task-based parallel reconfigurable computing systems, Journal of Signal Processing Systems 93 (2021) 545–563.

[71] B. Hutchings, M. Wirthlin, Rapid implementation of a partially reconfigurable video system with pynq, in: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2017, pp. 1–8.

[72] The Composable Video Pipeline — PYNQ Composable Overlays 1.0.2 documentation.
URL https://pynq-composable.readthedocs.io/en/latest/video_pipeline.html

[73] J. Goeders, T. Gaskin, B. Hutchings, Demand driven assembly of fpga configurations using partial reconfiguration, ubuntu linux, and pynq, in: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2018, pp. 149–156.

[74] G. Tagliavini, D. Cesarini, A. Marongiu, Unleashing fine-grained parallelism on embedded many-core accelerators with lightweight openmp tasking, IEEE Transactions on Parallel and Distributed Systems 29 (9) (2018) 2150–2163.

[75] A. Kurth, A. Capotondi, P. Vogel, L. Benini, A. Marongiu, Hero: An open-source research platform for hw/sw exploration of heterogeneous manycore systems, in: Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems, 2018, pp. 1–6.

[76] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, L. Benini, Pulp: A parallel ultra low power platform for next generation iot applications, in: 2015 IEEE Hot Chips 27 Symposium (HCS), IEEE, 2015, pp. 1–39.

[77] Zynq UltraScale+ MPSoC.
URL https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[78] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, R. Wittig, Ultrascale+ mpsoc and fpga families, in: 2015 IEEE Hot Chips 27 Symposium (HCS), IEEE, 2015, pp. 1–37.

[79] F. Zaruba, L. Benini, The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-v core in 22-nm FDSOI technology 27 (11) 2629–2640. doi:10.1109/TVLSI.2019.2926114.

[80] A. Bernardi, G. Brilli, A. Capotondi, A. Marongiu, P. Burgio, An fpga overlay for efficient real-time localization in 1/10th scale autonomous vehicles, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2022, pp. 915–920.

[81] G. Valente, G. Brilli, T. Di Mascio, A. Capotondi, P. Burgio, P. Valente, A. Marongiu, Fine-grained qos control via tightly-coupled bandwidth monitoring and regulation for fpga-based heterogeneous socs, IEEE Transactions on Parallel and Distributed Systems.

[82] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, L. Benini, Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications, in: 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), IEEE, 2017, pp. 1–8.

[83] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, L. Benini, Near-threshold RISC-v core with DSP extensions for scalable IoT endpoint devices 25 (10) 2700–2713. doi:10.1109/TVLSI.2017.2654506.

[84] F. Conti, L. Benini, A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters, in: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 683–688, ISSN: 1558-1101. doi:10.7873/DATE.2015.0404.

[85] F. Conti, P. D. Schiavone, L. Benini, XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference 37 (11) 2940–2951. doi:10.1109/TCAD.2018.2857019.

[86] C. Sau, T. Fanni, C. Rubattu, L. Raffo, F. Palumbo, The multi-dataflow composer tool: An open-source tool suite for optimized coarse-grain reconfigurable hardware accelerators and platform design, Microprocessors and Microsystems 80 (2021) 103326.

[87] T. Fanni, A. Rodríguez, C. Sau, L. Suriano, F. Palumbo, L. Raffo, E. de la Torre, Multi-grain reconfiguration for advanced adaptivity in cyber-physical systems, in: 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), IEEE, 2018, pp. 1–8.

[88] J. Sérot, F. Berry, S. Ahmed, Caph: a language for implementing stream-processing applications on fpgas, in: Embedded Systems Design with FPGAs, Springer, 2013, pp. 201–224.

[89] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, L. Benini, Gap-8: A risc-v soc for ai at the edge of the iot, in: 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), IEEE, 2018, pp. 1–4.

[90] B. D. De Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss, et al., A clustered manycore processor architecture for embedded and accelerated applications, in: 2013 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2013, pp. 1–6.

[91] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, Nvidia tesla: A unified graphics and computing architecture, IEEE micro 28 (2) (2008) 39–55.

[92] T. Fischer, M. Rogenmoser, T. Benz, F. K. Gürkaynak, L. Benini, Floonoc: A 645-gb/s/link 0.15-pj/b/hop open-source noc with wide physical links and end-to-end axi4 parallel multistream support, IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

[93] Y. Lei, Q. Deng, S. Long, S. Liu, S. Oh, An effective design to improve the efficiency of dpus on fpga, in: 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2020, pp. 206–213.

[94] A. Magnani, G. Brilli, A. Marongiu, Architectural design exploration of a lane detection vision pipeline for fpga-based f1tenth autonomous vehicles, in: *To appear* CF '25 Companion: Proceedings of the 22st ACM International Conference on Computing Frontiers: Workshops and Special Sessions, ACM, 2025.

[95] G. Bellocchi, A. Capotondi, L. Benini, A. Marongiu, Enabling fast system-level integration and prototyping of accelerator-rich platforms, in: Proceedings of the 22nd ACM International Conference on Computing Frontiers: Workshops and Special Sessions, 2025, pp. 54–57.

[96] M. O'Kelly, H. Zheng, D. Karthik, R. Mangharam, F1tenth: An open-source evaluation environment for continuous control and reinforcement learning, Proceedings of Machine Learning Research 123.

[97] Vitis Libraries.
URL `https://www.xilinx.com/products/design-tools/vitis/vitis-libraries.html`