MPSoC4Drones: An Open Framework for ROS2, PX4, and FPGA Integration

Frederik Falk Nyboe, *IEEE Member*, Nicolaj Haarhøj Malle, *IEEE Member*, Emad Ebeid, *IEEE Senior Member*
Drone Infrastructure Inspection and Interaction (DIII) Group,
SDU UAS Center, University of Southern Denmark, Odense, Denmark
ffn|nhma|esme@mmmi.sdu.dk

***Abstract*— Autonomous drones are facing a tough efficiency challenge due to limitations on the utilized processing hardware units. Among these limitations is the tradeoff between fast computing and low power consumption; between functional complexity and flight time. Recent progressions point to FPGAs for accelerating heavy processing. In this work, we present the MPSoC4Drones Framework; a novel framework for organizing FPGA-design and OS build projects. The framework combines tools for creating bootable images for the Ultra96-V2 board.**

We show how MPSoC4Drones organizes the build, combining the latest well-known tools for research and industrial drone development, Ubuntu 20.04, PX4 autopilot, and ROS2 middleware. We validate the framework through a computationally intensive deep learning use case implemented in the MPSoC4Drones framework. We show the superior throughput and low power consumption of FPGA processing compared to classical CPU and GPU approaches. Finally, we offer the full framework as open-source.

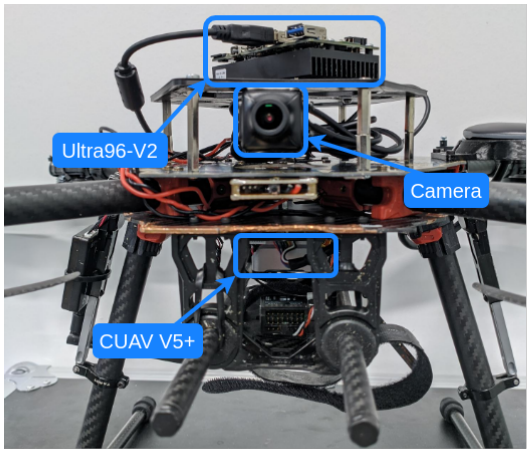

Fig. 1. Drone platform for hardware acceleration example

I. Introduction

The market of small aerial robots, so called drones, is growing rapidly and is expected to reach 1.59 billion units by 2024 according to FAA aerospace forecast report [1]. Drone technologies are also growing steadily, utilizing advanced algorithms to become more intelligent and capable of accomplishing sophisticated tasks [2]. Running these algorithms effectively onboard the drone while meeting their real-time requirements and energy efficiency requires significant hardware resources [3]. Currently, Central Processing Unit (CPU) or Graphics Processing Unit (GPU) card-sized embedded computers are widely adopted by the drone communities due to their flexibility and ease-of-use. However, when it comes to processing speed and energy consumption, conventional CPUs and GPUs are not suited for latency-critical applications and power-limited devices, such as small autonomous drones, due to their generic hardware architecture [2].

On the other hand, FPGA reconfigurable hardware is designed to handle latency-critical tasks while being energy efficient due to the ability to parallelize computations massively [4]. That brings FPGAs to be the next candidate for autonomous drone design. However, working with FPGAs requires a deep understanding of hardware logic and that creates a barrier in adopting them in the robotic communities.

In this paper, we propose an open-source framework that offers seamless integration of the FPGA into the robotic and drone systems. The framework follows the bottom-up design process approach to enable the robotics and drone developers to build their FPGA hardware architecture to accelerate heavy tasks while being able to exchange data seamlessly with high-level software tasks running as ROS2 nodes. The framework utilizes the cutting edge Multiprocessing System-on-Chip (MPSoC) chips to run the ROS2 software stack with PX4 communication on the top of the FPGA on a single chip.

The contributions of this work are

- Full integration of Ubuntu 20.04, ROS2, and PX4 communication nodes with an FPGA-aware kernel;
- Easy organization of development projects for the Ultra96-V2 board, supporting an iterative workflow;
- Automatic build from design sources to bootable artifacts supporting FPGA design at a low abstraction level and with High-level Synthesis (HLS);
- Open-source project template, scripts, and example.

The framework is available at our GitHub [5].

The rest of the paper is structured as follows: Sec. II outlines related work; sec. III gives an overview of the background characteristics of Field-Programmable Gate Arrays (FPGAs), MPSoCs, and the Ultra96-V2 single-board computer; sec. IV presents the methodology used in developing the MPSoC4Drones framework; sec. V outlines in greater detail the functionality of the tools used by the MPSoC4Drones framework; sec. VI gives a full description of the workflow proposed with the MPSoC4Drones framework;

sec. VII outlines the validation of the framework, showcasing a project implemented using MPSoC4Drones; finally, sec. VIII concludes the work.

II. Related Work

Notable attempts have been made to leverage the power of FPGAs in Linux based systems, starting with the well-established approach of building a Yocto based distribution [6] with tools provided by chip manufacturer Xilinx, acquired by AMD in 2022. While this option comes with the advantage of full support from Xilinx and is favored by some industrial players, it requires extensive knowledge in hardware and embedded systems. Furthermore, a Yocto based Linux distribution is infeasible for rapid development in robotics and Unmanned Aerial Vehicle (UAV) research, where package reuse is of the essence. With the motivation to introduce the versatility of an Ubuntu based Linux distribution as well as the ease of the Python programming language, the open-source PYNQ project [7] offers a fully software-centric approach. Here, the Python developer has at her disposal a set of FPGA modules or *overlays* with various functionalities which she can load onto the FPGA and call with live data using the PYNQ Python API. The PYNQ image is provided for a selected set of boards either as a pre-built image or as a set of scripts to re-create the released image. While the software-centric approach to Linux-FPGA design, where an FPGA module is called from the CPU to offload a processing task, is extremely powerful, it makes up only half of the story; the designer might want the flexibility of fine-grained control of the FPGA implementation, as with the hardware-centric approach. Additionally, as for any system with performance requirements, a roboticist might want to use C or C++.

Recent years have seen a spike in efforts to combine the Robot Operating System (ROS) with FPGAs in various ways. In 2015, Yamashina et al. [8] proposed a methodology for designing ROS compliant FPGA components. In 2016, the group [9] continued to build a tool for automatic generation of these ROS compliant components in order to increase productivity. The same year, Ohkawa et al. [10] presented a case study utilizing the same tool, evaluating different compute architectures for the implementation of a simultaneous localization and mapping (SLAM) algorithm, exploring parallelization by FPGA and remote server processing.

In 2017, Sugata et al. [11] proceeded to introduce FPGA based acceleration at the transport layer of ROS, compatible with the previous work in ROS compliant FPGA components. In 2019, further FPGA implementation at the ROS communication layer was achieved [12]. With ROS2, specific applications have been demonstrated such as hardware accelerated reinforcement learning [13] and object detection [14], with clear performance gains compared to CPU implementations. Additionally, a tool has been presented in [15] for automatic generation of ROS2 nodes for integration with FPGA modules built with the Vivado HLS tool, a predecessor to Vitis HLS. Additionally, methodologies have been proposed for general integration of FPGAs with ROS [16] [17], and for full or partial implementation of ROS2 nodes on FPGAs [18].

The most notable contribution to the ecosystem of ROS2 based FPGA acceleration tools is the Kria Robotics Stack (KRS) [19] by Xilinx. The KRS provides extensions for ROS2 which enable the developer to accelerate ROS2 nodes through the normal ROS2 development interface. Thus, the hardware considerations of usual FPGA development is abstracted away. KRS is a very promising candidate for a ROS2-FPGA development framework, however currently like the PYNQ project suffers from the inflexibility only being software-centric. We acknowledge, however, that the KRS will likely overcome this limitation as the tool matures; the interested reader can see [20] and [21].

Also in drone applications, FPGAs have been applied, adding to the proof that hardware acceleration by FPGA is and will be an important technology for drones. In 2019, Cain et al. [22] presented an FPGA based drone flight controller. The flight controller was implemented as an overlay based on the PYNQ project. In 2020, Moreac et al. [23] proposed FPGA based computer vision for drones by hardware-in-the-loop simulation. Finally in 2021 at our research group, Kövari [24] demonstrated the use of FPGA accelerated deep learning inference for autonomous drone navigation, also based on PYNQ.

III. Background

FPGAs have been increasingly used in various applications since the '90s. A series of technological breakthroughs have established FPGAs as a well reputable tool for accelerating processes which have very high requirements for real-timeliness and data throughout. Formerly, FPGAs were mostly used within telecommunication, whereas the last 20 years have seen an expansion of FPGA technology into various other areas of computation, most notably in data centers where high throughput is of the essence. In later years, there has been a push for the use of FPGA related technologies in robotics and drones as well, mostly due to 1) the ongoing increase in computational complexity within robotics applications, and 2) the increasing maturity of the available tools for FPGA development [25] [26].

A. The Field-Programmable Gate Array (FPGA)

The FPGA is an integrated circuit which in a grid structure features a significant number of various logic gates, lookup tables, memory elements, digital signal processing blocks, and other logic building blocks linked by a configurable array of electrical connections. This enables the designer to carefully configure and reconfigure any specialized logic circuit for any application. FPGAs are traditionally programmed in a hardware descriptive language such as VHDL, however recent years have seen a great maturation of tools which allow FPGA design at higher levels of abstraction [26].

B. The Multiprocessing System-on-Chip (MPSoC)

The Zynq UltraScale+ MPSoC series from Xilinx is an evolution of the 2011 Zynq-7000 series of chips which combined FPGA fabric with a CPU. The Zynq UltraScale+ MPSoCs features a quad-core CPU denoted the Application Processing Unit (APU), a dual-core CPU called the Real-time Processing Unit (RPU), peripherals, memory, data infrastructure, and FPGA fabric. This enables the designer to run non real-time critical software with a high level of abstraction on the APU using an operating system (OS) such as Linux; to design real-time critical software for the RPU to run bare-metal or on a real-time OS; and finally to design any application specific hardware acceleration and custom logic circuit for the chip's FPGA [27] [28].

The MPSoC uses the AMBA Advanced eXtensible Interface (AXI) open data communication standard for intra-chip communication. This enables the designer to build FPGA circuit which maps into the memory of the CPU for easy integration of implemented hardware acceleration cores in the application [29].

C. The Ultra96-V2

The Ultra96-V2 is a development board from Avnet which features a Zynq UltraScale+ MPSoC. A block diagram of the Ultra96-V2 is seen in Fig. 2. In comparison to other off-the-shelf MPSoC development boards, the board has a small form factor and is light weight. Additionally, the board features plenty of I/O including USB. Finally, the board features a WiFi module which is essential for development. The board comes at a price of 299 USD, which is considerably lower than some other boards in the same category. For these reasons, the board lends itself well for drone onboard computing applications [31].

IV. Methodology

The need for this framework stems in general from the wish to combine FPGA functionality with a regular CPU setup with Ubuntu and ROS on a drone. The natural choice for this purpose is to leverage a chip from the state-of-the-art Zynq UltraScale+ MPSoC family from Xilinx, as mentioned previously. Running Ubuntu on the APU would then provide companion computer capabilities with the addition of the Programmable Logic (PL) (synonym for the FPGA fabric in the MPSoCs). High-level mission logic and general functionality runs on the CPU in Ubuntu using ROS; complex, high-throughput processing is accelerated in the PL. The Ultra96-V2 development board by Avnet is deemed fit for this purpose of advanced companion computing [28].

Developing designs for the PL of the Ultra96-V2 is deeply ingrained in the tools and workflows of Xilinx-specific embedded systems development. Incorporating a custom PL design into a Linux OS becomes complicated as changes to the PL essentially mean changes to the hardware from the point of view of the OS, i.e. the physical addresses and hardware devices. Therefore, the kernel needs to be customized and rebuilt for changes to the PL. Accordingly, Xilinx provides the tool Vivado for PL design, and the tool PetaLinux for customizing and building the OS [32] [6].

Additionally, Avnet provides scripts for building an image for the Ultra96-V2 using these tools. The scripts automatically configures parts of the PetaLinux build which relies on the static hardware components on the Ultra96-V2, the so called Board Support Package (BSP) contained in a set of meta-layers for the underlying Yocto build process [33] [6].

The MPSoC4Drones framework presented in this article relies on these tools. The framework is structured as a set of directories and scripts within a parent directory. In here, all tools are contained for setting up the project, building the components of the project, packaging the build products, and committing the changes back to the setup scripts. Using the original MPSoC4Drones repository as a template, the tools then allow versioning of changes made to the specific project.

The framework was designed to do the following:

- Ease the integration of Ultra96-V2 BSP details provided by Avnet in custom PetaLinux projects;
- Allow for seamless integration of custom Vivado FPGA designs into the PetaLinux project;
- Utilize the power of PetaLinux for customization of the Linux kernel, modules, and drivers, but build a Ubuntu 20.04 OS on top as favored among the drone/robotics research community;
- Incorporate automatic installation of ROS2 and PX4 dependencies in order to enable Ultra96-V2-PX4 communication;
- Supply all of the tools necessary to accomplish this in a project oriented framework fit for iterations on all parts of the design flow;
- Facilitate that the changes to the project can be committed, such that the project scripts automatically generates the customized project instead of the default project so the specific project can be redistributed by using the original GitHub repository as template.

V. Toolchain and Standards for Embedded System Design on the Ultra96-V2

Xilinx provides a set of tools for developing embedded systems targeting their architecture. Following a layered approach to embedded design, the integration of the FPGA with the CPU enables the designer to not only customize the firmware but additionally to customize the hardware on top of which the firmware is running. Having built a custom hardware layout, other tools are used for building the boot files and the Linux kernel on top of this hardware. Finally, custom scripts assemble a fully functional Ubuntu 20.04 root file system on top of the built artifacts. The chain of tools used in the MPSoC4Drones workflow are outlined in this section. The toolchain is visualized in figure 3.

A. Xilinx Vivado

The first fundamental tool in the chain is Xilinx Vivado [32] in which the custom hardware implementations are integrated. The Vivado Design Suite is a fully equipped IDE for FPGA design for Xilinx' devices. The tool employs a block oriented design workflow where the developer is able to graphically integrate blocks with various functionality into a single logic circuit for deployment on an FPGA. Additionally, the designer is able to manage non-fixed I/O in

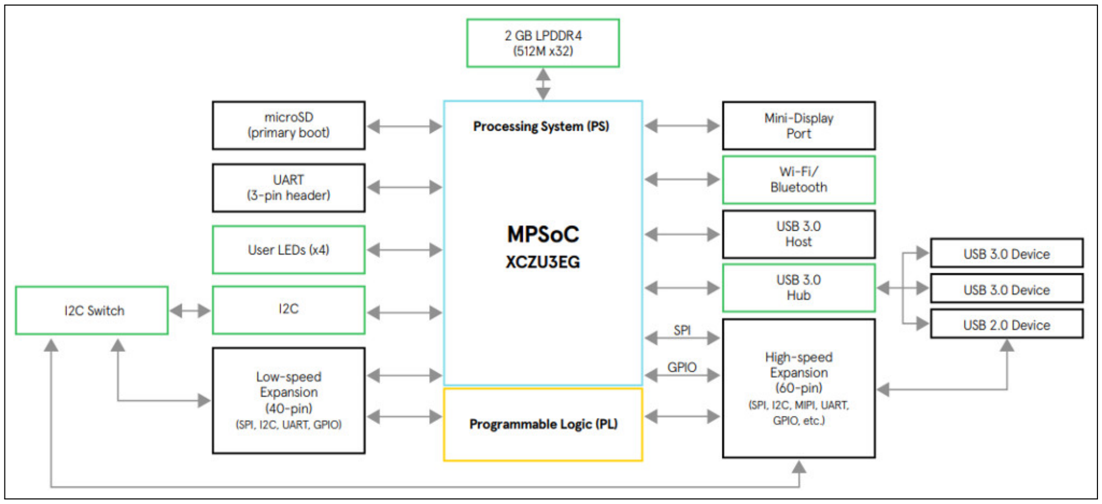

Fig. 2. The Ultra96-V2 block diagram [30]

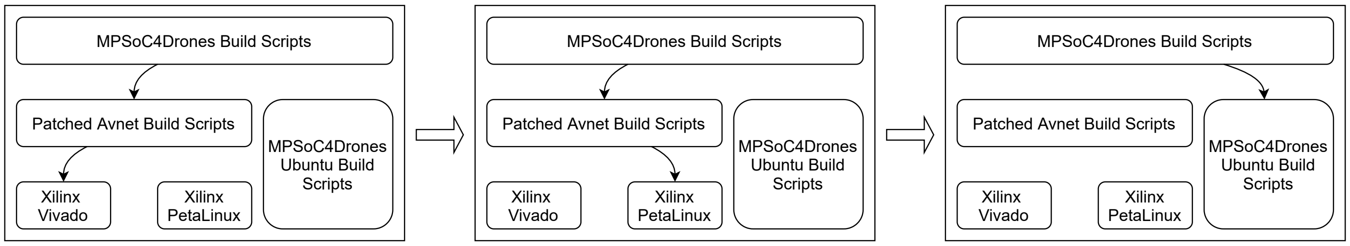
MPSoC4Drones Build Scripts
Patched Avnet Build Scripts
MPSoC4Drones Ubuntu Build Scripts
Xilinx Vivado
Xilinx PetaLinux

Fig. 3. Visualization of the toolchain used in MPSoC4Drones for building the hardware, the kernel, and the Ubuntu root filesystem, from left to right respectively

Vivado. Where classical approaches for programmable logic design with Register Transfer Level (RTL) coding in Verilog and VHDL is supported for fine grained control of the functionality, the tool enables the designer to integrate blocks developed with other tools as well. An important auxiliary tool for implementation of more advanced processing in hardware is Xilinx Vitis HLS [34]. An example of the use of Vitis HLS with Finn [35] for the deployment of hardware accelerated neural network inference under the MPSoC4Drones framework is given in section VII.

Secondly, Vivado builds the actual bitstream which is the file that describes the hardware layout for the FPGA from the block design. Finally, Vivado provides detailed insights into the implementation.

With MPSoC4Drones, a default hardware layout is automatically supplied as a project in Vivado upon project setup. Custom FPGA implementations can then be appended to the default project natively in Vivado, or the default project can be used as is. MPSoC4Drones automatically handles the design build all the way from RTL synthesis to export of an `.xsa` hardware specification file to be used in PetaLinux.

B. Xilinx PetaLinux

For building Linux distributions targeting boards featuring their chips, Xilinx provides the PetaLinux tool [6]. PetaLinux combines an underlying Yocto Project [36] embedded Linux build with layers and functionality provided by Xilinx. Hence, the flexibility of Linux development with Yocto is paired with automatic integration of FPGA implementations as well as Xilinx kernel modules and libraries for control of the hardware features of the chip. Though PetaLinux can build a full Linux system for deployment, in MPSoC4Drones only the boot files and kernel with modules are built with PetaLinux, as the root filesystem should be Ubuntu.

In a PetaLinux project, the developer can add additional kernel drivers and modules. Adding custom modules can be done as in any Yocto Project, but is out of the scope of this article. Arbitrary stock kernel modules can be configured in a simple fashion inside the PetaLinux project. Additionally, the kernel can be made aware of custom implemented FPGA modules by simple additions to the device tree file. Once again, this is demonstrated in section VII.

In MPSoC4Drones, the integration of the hardware specification obtained from Vivado with the PetaLinux project is automatically handled. Additionally, the full PetaLinux build is also covered by the MPSoC4Drones commands. When changes are made to the PetaLinux project, be it modified FPGA design, changes to the kernel configuration, or additions to the device tree, previous builds are cached and subsequent builds will be faster than the initial build.

C. Avnet Build Scripts and Meta Layers

From Avnet, a set of GitHub repositories are provided publicly which hosts scripts and files related to OS builds for their boards. Their build architecture dictates the build to be run with a single entry point. Using their scripts for building a Linux image for the Ultra96-V2, a default FPGA design is generated, a default kernel configuration is used, and a pre-determined set of packages is installed in the root filesystem, which is delivered from the PetaLinux build and is thus not Ubuntu, but Yocto based. The build works in a monolithic manner and does not facilitate modifications to the hardware design, the kernel configuration, or the root filesystem without deep modifications to the scripts.

There are some clear advantages, though, to what Avnet provides, which is 1) a template for a build which is known to work, and 2) the BSP for the Ultra96-V2 contained in a set of Yocto meta-layers. The most significant role of the provided meta-layers are to specify how the external hardware features on the board is connected to the chip which is essential information in order for the OS to correctly interface with the WiFi module, the USB ports, the serial I/O, etc. Additionally, the meta-layers provide a set of board-specific patches to the Linux kernel, and a board-custom kernel module driver for the WiFi module.

In MPSoC4Drones, the Avnet build scripts are used as a starting point for providing a reliable build. However, the scripts are patched in order to break down their otherwise monolithic structure. This enables a fully partitioned build process which in much greater sense accommodates an incremental and iterative design process. The Avnet meta-layers are also patched for the MPSoC4Drones build in order to omit all data regarding the root filesystem, as this is not built with PetaLinux in MPSoC4Drones.

D. MPSoC4Drones Build Scripts

Supervising the full build, the MPSoC4Drones scripts organize the build files and call the build tools. Additionally, MPSoC4Drones features custom scripts for building the Ubuntu root filesystem on top of the Vivado and PetaLinux build products. With multiple entry points, the MPSoC4Drones scripts can carry out full or partial builds. MPSoC4Drones utilizes the PetaLinux caching mechanism for faster subsequent builds. Additionally, the Ubuntu root filesystem build also leverages previous builds.

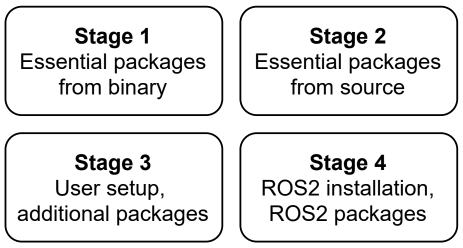
Stage 1
Essential packages from binary
Stage 2
Essential packages from source
Stage 3
User setup, additional packages
Stage 4
ROS2 installation, ROS2 packages

Fig. 4. Stages of the Ubuntu root filesystem build

The MPSoC4Drones Ubuntu root filesystem build scripts builds Ubuntu in four stages. These stages are seen in figure 4. From an officially released base Ubuntu 20.04 root filesystem for ARM64 architectures, the first stage of the build primarily populates the filesystem with essential packages which can be installed from binaries. In the second stage, further essential packages are installed from source, the most important of which are userspace packages provided by Xilinx for control of the hardware features of the chip. The installation of these packages into the root filesystem are handled by PetaLinux in builds where the PetaLinux supplied root filesystem is used in the end application. However, as a custom Ubuntu root filesystem is build on top of the PetaLinux built kernel in MPSoC4Drones, these packages are built from source. In order to achieve a working installation, patched installation scripts from the PYNQ project are utilized in this stage [37]. In stage 3, the primary preparations of the root filesystem happen. Here, the user is added, permissions are set up, and additional packages are installed. A developer using MPSoC4Drones would in most cases add any custom packages to be installed in this stage. Finally, stage 4 covers installation of ROS2 Foxy, PX4 dependencies, PX4 ROS2 packages, and any additional ROS2 packages specified by the developer.

E. Memory Mapped Hardware Device Integration

To illustrate how the tools are used in succession, an example of integration of a memory mapped hardware device into the built system is given here.

Assuming a hardware component has been developed, e.g. using Vitis HLS, which maps into the memory of the CPU by connection to the AXI bus, the process of enabling the utilization of this hardware component with ROS2 involves the usage of the above tools. First, the hardware block is integrated into the block design in Vivado. The AXI connections are made and the addresses are configured, using Vivado's address editor for the AXI bus. The hardware is then built to obtain the hardware specification file. The hardware specification file is included into the PetaLinux project. A node is appended to the device tree specification file to associate the custom hardware device with the generic userspace I/O driver in Linux. The PetaLinux project and the Ubuntu root filesystem is built. If the hardware device was originally developed with Vitis HLS, the automatically generated userspace drivers can then be included in a ROS2 package to facilitate control of the hardware device from ROS2. Existing ROS2 nodes can be appended to the ROS2 package installation in the Ubuntu root filesystem setup stage 4.

For further details and examples on the described methodology, the reader is referred to the DIII GitHub page [38]. Additionally, the workflow of the MPSoC4Drones framework is described in greater detail in the following section.

VI. The MPSoC4Drones Project Framework

The MPSoC4Drones framework is an MPSoC project framework for drone research applications targeting the Ultra96-V2 as companion computer with external PX4 powered flight controller. The framework applies the toolchain described previously for building images with custom programmable logic design, Ubuntu 20.04, and ROS2. Thus, the framework wraps a set of existing tools for embedded development provided by Xilinx as well as scripts provided by Avnet and custom scripts in order to execute the builds.

The work aims to provide an easy-to-use framework for integration projects targeting the Ultra96-V2 board. The framework facilitates integration of FPGA PL design with well-established tools in the drone research community. The framework lets the developer

- Handle the full process of boot image creation for custom drone applications targeting the Ultra96-V2 board;
- Design full project specific FPGA layouts from a bare-minimum default block design in Xilinx Vivado 2020.2;
- Handle a PetaLinux project using either a custom or the default FPGA design;
- Automatically incorporate the Ultra96-V2 specific meta-layers provided from Avnet into the PetaLinux project;
- Modify the meta-layers of the PetaLinux project in compliance with custom FPGA designs;
- Build the Vivado project and PetaLinux project;
- Build an Ubuntu 20.04 root filesystem ready with ROS2 Foxy and Fast-RTPS for PX4 Flight Controller communication; and
- Package the integrated FPGA layout and Ubuntu OS onto a bootable SD card.

A system diagram visualizing the outcome of an MPSoC4Drones built image is given in Fig. 5.

The framework consists of a directory structure and a set of command line tools, and everything is supplied within the repository. One instance of the repository is copied for one MPSoC4Drones project. The interested reader is referred to the MPSoC4Drones GitHub repository for more detail on the available commands [39].

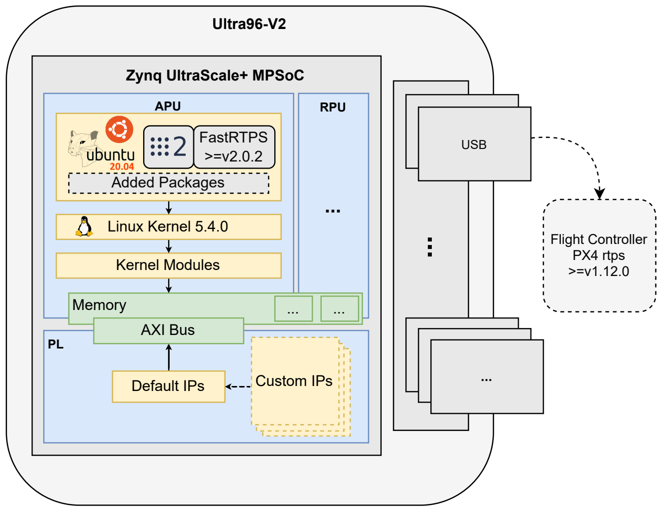

Fig. 5. System diagram of MPSoC4Drones build results

A. Framework Workflow

The framework supports an iterative workflow where changes to the PL design, PetaLinux meta-layers, and Ubuntu root filesystem setup is made and remade throughout the development process. A project is instantiated by using the MPSoC4Drones repository as a template for a new repository on GitHub.

The workflow is grouped in the following four steps:

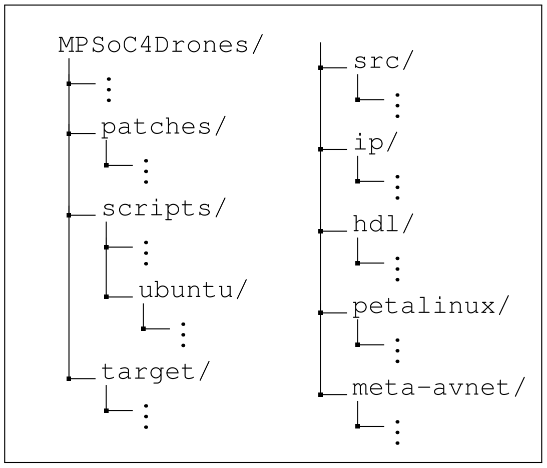

Fig. 6. The essential parts of the MPSoC4Drones framework directory structure

1) **Setup**: Setting up the project structure. In this step, the relevant Avnet repositories [33] containing the Avnet meta-layers and the Avnet build-script, as well as the PYNQ repository [37], are cloned and patched, the Vivado and PetaLinux projects are generated.
2) **Build**: Building the individual project components. This step covers the process of building the hardware design in the Vivado project, the kernel in the PetaLinux project, and the Ubuntu root filesystem.
3) **Packaging**: Packaging the build products. This step covers packaging the build products onto an already formatted SD card as well as creating a boot image for later flashing onto an SD card.
4) **Commitment**: Committing the changes made to the project. This step covers committing changes made to different parts of the project back to the scripts that generate the project in the *setup* step, such that the project can be pushed to a new git repository.

The workflow is visualized in Fig. 7. In the figure, the small boxes represent sub-steps within each of the four steps explained above. The normal arrows indicate step precedence in the toolchain, meaning that traversing the arrows backwards indicate dependency of execution between the steps. The grey ellipses represent specific development scenarios, and the dotted arrows from the development scenarios to the workflow sub-steps indicate the toolchain entry point given the specific scenario.

B. Project Directory Structure

Fig. 6 shows the relevant parts of the MPSoC4Drones directory structure.

In a fresh copy of the MPSoC4Drones repository, the naked default project is contained. This consists simply of the `scripts/` folder containing all of the scripts comprising the set of MPSoC4Drones tools including the `settings.sh` script, which sources the command line tools, the `patches/` folder containing the patches to apply to the Avnet and PYNQ repositories, the `src/` folder containing the design sources, and the `ip/` folder for user added hardware IPs.

In the *setup* step of the workflow, the Avnet repositories are cloned into the `bdf/`, `hdl/`, `petalinux/`, and `meta-avnet/` folders and patched. Due to the nature of the Avnet build scripts and how they interact, the directory structure proposed by Avnet is kept within MPSoC4Drones.

The cloned directories comprises the following: The `bdf/` folder contains the Avnet bdf repository, which provides board definition files for the Ultra96-V2, used in Vivado to specify the characteristics of the board;

The `hdl/` folder contains the Avnet hdl repository which provides the Vivado specific scripts for project creation and hardware building. The repository is patched from the original provided by Avnet in order to more flexibly facilitate hardware development in Vivado, and also to define a bare minimum Vivado project necessary for the built OS to be able to run. The `hdl/` folder contains a subfolder `hdl/projects/u96v2_sbc_mp4d_2020_2/` in which the Vivado project lives. The entry point for making modifications to the default Vivado project and to integrate with custom PL designs is the Vivado project file `u96v2_sbc_mp4d.xpr`;

The `petalinux/` folder contains the Avnet petalinux repository which provides the PetaLinux specific scripts for project creation and PetaLinux building. The repository is patched from the original provided by Avnet in order to more flexibly facilitate development with/without updating the hardware specification, without rebuilding the project in its entirety, etc. The `petalinux/` folder contains a subfolder `petalinux/projects/u96v2_sbc_mp4d_2020_2/` in which the PetaLinux project is contained. Should the user wish to apply advanced configurations to the PetaLinux project or otherwise directly utilize the PetaLinux API, it would be done in this location;

Finally, the `meta-avnet/` folder contains the Avnet meta-avnet repository. This folder contains the meta-layers provided by Avnet for the PetaLinux build. These layers define, among other things, patches to the kernel, hardware drivers, etc., specific to the Ultra96-V2, as well as the kernel driver configuration for the PetaLinux project. The repository is patched from the original provided by Avnet in order to remove specification of excessive packages to be installed in the user layer of the root filesystem built by PetaLinux, as this filesystem is replaced in MPSoC4Drones by a Ubuntu 20.04 filesystem. This modification significantly decreases build time. Additionally, some changes are made to the kernel driver configuration necessary for adding PX4 communication to the built OS. When developing with MPSoC4Drones, if any changes to the kernel configuration is required, or if additional custom kernel modules are to be added, it must be applied to the appropriate parts of the `meta-avnet/` folder in a Yocto Linux fashion.

After having invoked the *build* step or *package* step of the framework, a final folder will appear, the `target/` folder. In this folder, the `BOOT/` and `rootfs/` folders represent the BOOT and rootfs partitions on the SD card to be flashed to, containing the built boot loader files and the built root filesystem, respectively. Additionally, the `target/` folder contains some build products from the PetaLinux build to be used during the Ubuntu build.

C. Project Setup

This step of the MPSoC4Drones workflow generates the project structure from the naked repository. The *setup* is done in the following three steps, as is also visualized in Fig. 7:

Setup step 1: **Cloning and patching the Avnet repositories**. This step clones all of the necessary Avnet repositories into the respective folders. For each repository that needs to be patched, the patch is located in the `patches` folder and applied on the repository;

Setup step 2: **Setting up the Vivado project**. Using the scripts in the patched version of the Avnet hdl repository now contained in the `hdl/` folder, the Vivado project is created in the directory `hdl/projects/u96v2_sbc_mp4d_2020_2/`. The default bare-minimum block design is generated using the modified Avnet `TCL` scripts and an exported `TCL` script for recreation of the block design, the board definition files are added, and a constraints file is added to the project for reference and later pinout. This step requires that *setup step 1* has been executed;

And *Setup step 3*: **Setting up the PetaLinux project**. Using the scripts in the patched version of the Avnet petalinux repository now contained the `petalinux/` folder, the PetaLinux project is created in the directory `petalinux/projects/u96v2_sbc_mp4d_2020_2/`. This step requires that *setup step 1* has been executed.

Having sourced the `settings.sh` script, the *setup* step is invoked with the command `mp4d-setup`. Options can be specified to call the sub steps individually.

D. Project Build

The *build* step builds the individual components of the MPSoC4Drones project: The hardware, the boot loader, the kernel, and the Ubuntu root filesystem. The build has the following five steps, as is also visualized in Fig. 7:

Build step 1: **Building the Vivado project hardware specification**. This step runs synthesis and implementation on the design contained in the `hdl/projects/u96v2_sbc_mp4d_2020_2/` Vivado project directory. The build utilizes the patched versions of the Avnet hdl build scripts. The design bitstream is generated and the hardware specification `.xsa` file is exported. This step requires that *setup step 2* has been executed;

Build step 2: **Configuring the PetaLinux project**. This step configures the PetaLinux project for build. This is done by importing the built hardware specification and importing the Avnet meta-layers. The process is handled by the modified Avnet PetaLinux build scripts. This step requires that *setup step 3* and *build step 1* have been executed;

Build step 3: **Building the PetaLinux project**. This step builds the PetaLinux project contained in the `petalinux/projects/u96v2_sbc_mp4d_2020_2/` directory, including the boot loader and the kernel. The build is once again facilitated through the use of the patched versions of the Avnet PetaLinux build scripts. The boot files output products

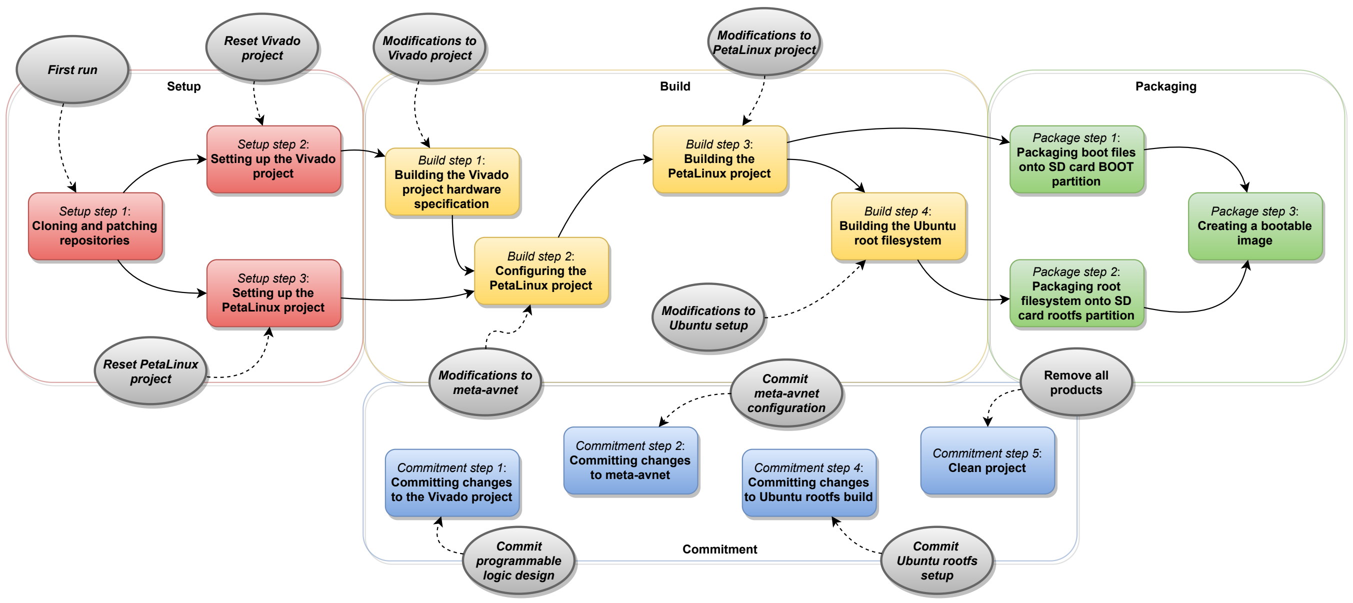
First run
Reset Vivado project
Modifications to Vivado project
Modifications to PetaLinux project
Setup
Build
Packaging
Setup step 2: Setting up the Vivado project
Build step 1: Building the Vivado project hardware specification
Build step 3: Building the PetaLinux project
Package step 1: Packaging boot files onto SD card BOOT partition
Setup step 1: Cloning and patching repositories
Build step 4: Building the Ubuntu root filesystem
Package step 3: Creating a bootable image
Build step 2: Configuring the PetaLinux project
Setup step 3: Setting up the PetaLinux project
Package step 2: Packaging root filesystem onto SD card rootfs partition
Modifications to Ubuntu setup
Reset PetaLinux project
Modifications to meta-avnet
Commit meta-avnet configuration
Remove all products
Commitment step 2: Committing changes to meta-avnet
Commitment step 1: Committing changes to the Vivado project
Commitment step 4: Committing changes to Ubuntu rootfs build
Commitment step 5: Clean project
Commit programmable logic design
Commitment
Commit Ubuntu rootfs setup

Fig. 7. Visualization of the MPSoC4Drones framework workflow

are stored in the target directory `target/BOOT/`. This step requires that *build step 2* has been executed;

And *Build step 4*: **Building the Ubuntu root filesystem**. This step builds the Ubuntu 20.04 root filesystem. First, the raw Ubuntu 20.04 rootfs for ARM architecture is fetched. Upon this, certain packages are added for later cross compilation of PX4 dependencies.

Then, the build proceeds in the four stages mentioned previously using Qemu sessions to virtually install the required packages and to set up the OS. The majority of the installed packages are listed in the `scripts/stage_*_packages.txt` files, and additional packages can be added there for installation in the respective stages. The kernel modules and drivers from the PetaLinux build is imported into the root filesytem. The root user and the system user is configured, ROS2 is installed, and Fast-RTPS is cross compiled in order to facilitate PX4-ROS2 communication. The PX4 ROS2 packages are fetched into the created ROS2 workspace under the primary user. Also here can the developer add ROS2 packages to be pulled for deployment. Finally, automatic loading of the WiFi kernel module driver is added. The built root filesystem is contained in the target directory `target/rootfs/`. This step requires that *build step 3* has been executed.

Having sourced the `settings.sh` script, the *build* step is invoked with the command `mp4d-build`. Options can be specified to call the sub steps individually.

E. Project Packaging

The *packaging* step of the MPSoC4Drones framework is simply used for packaging the build products onto the SD card. This is essentially done with the following steps, as visualized in Fig. 7:

Packaging step 1: **Packaging boot files onto the SD card BOOT partition**. This step packages the boot files contained in the target directory `target/BOOT/` onto the BOOT partition of the SD card. This step requires that *build step 3* has been executed;

Packaging step 2: **Packaging the root filesystem onto the SD card rootfs partition**. This step packages the full root filesystem contained in the target directory `target/rootfs/` onto the rootfs partition of the SD card. This step requires that *build step 4* has been executed;

Having sourced the `settings.sh` script, the *packaging* step is invoked with the command `mp4d-package`. Options can be specified to call the sub steps individually.

F. Project Commitment

The *commitment* step of the MPSoC4Drones framework finalizes the design process by writing back the changes - the developed FPGA design, changes to the device tree, changes to the kernel module configuration, and additions to the Ubuntu build scripts - to the scripts that generate the project during the *setup* step. The *commitment* step consists of the following sub-steps:

Commitment step 1: **Committing changes made to the Vivado project**. This step modifies the setup scripts such that the design changes applied to the initial Vivado project is preserved through running the setup scripts again. This is achieved by storing any added source design files and any added IPs to the patch that is applied to the Avnet hdl repository in *setup step 1*. Additionally, the `TCL` script which generates the block design is updated to reflect the resulting design. Finally, the constraints file is updated to preserve changes. The updated patch file `hdl_repo.patch` is committed to git.

Commitment step 2: **Committing changes made to the meta-avnet layers**. In this step, changes applied to the meta-avnet layers in the `meta-avnet/` folder for the PetaLinux build is similarly pushed back to the patch which is applied to the Avnet meta-avnet repository in *setup step 1*. Any modification made in the `meta-avnet/` folder is preserved. The updated patch file `patches/meta-avnet_repo.patch` is committed to git.

Commitment step 3: **Committing changes made to the Ubuntu rootfs setup-scripts**. This step simply commits changes made to the Ubuntu build scripts, `scripts/qemu_ubuntu_setup.sh` and `scripts/ubuntu_packages.sh`, to git. Though the functionality is trivial, the step is included to keep consistency in the framework.

Commitment step 4: **Cleaning the project**. This steps removes all of the generated files and folders in the project. Specifically, the step will remove all files listed on the `.gitignore`. Only to be executed after having committed all changes to be kept through the above *commitment* steps, the functionality in this final step of the framework workflow prepares the project for distribution and versioning on git.

Having sourced the `settings.sh` script, the *commitment* step is invoked with the command `mp4d-commit`. Options can be specified to call the sub steps individually.

VII. Framework Validation

The framework is validated with an example project implementation. The goal of the example is to show the integration of high-level APU software with ROS2 and low-level FPGA accelerated modules within an MPSoC4Drones project. For the application, a convolutional neural network accelerator is running in the FPGA. The accelerator appears as a memory mapped hardware device to the OS. A ROS2 system continuously calls the accelerator on an image stream from a camera to obtain bounding boxes of people in the image. The position of the bounding box in the image is then used to arm/disarm the drone through PX4. The example project is available at GitHub [40].

A. Example Setup

A custom quantized neural network for person detection is defined and trained in Pytorch with Brevitas [41]. Using the Finn [42] compiler, a hardware implementation is then built for the neural network. The neural network accelerator is imported into the Vivado block design. Two FPGA cores are made to handle the IO communication with the neural network block. The IO cores propagate images written to the respective address from the CPU to the neural network and read the output from the neural network back to the AXI registers accessible from the CPU. The blocks are implemented with Vitis HLS which supplies Linux UIO drivers. Before running the MPSoC4Drones build, the entries for the IO FPGA cores are added to the `src/system-user.dtsi` device tree file to instruct the OS to regard the hardware devices as generic UIO. This enables userspace applications to utilize the generated drivers. Based on the generated drivers, a ROS2 node is written which simply subscribes to an image topic, calls the hardware accelerator, and publishes the bounding box.

B. Results

Running the application, the throughput of the full pipeline from receiving a frame from the camera to having published the bounding box is measured at 25 FPS. The total latency of writing the image to the hardware accelerator, running the accelerator, and reading the result is measured at ≈ 86 ms.

The power consumption is < 6.4 W while running the application and < 5.8 W when idle. The measurements of the power consumption are shown in Fig. 8.

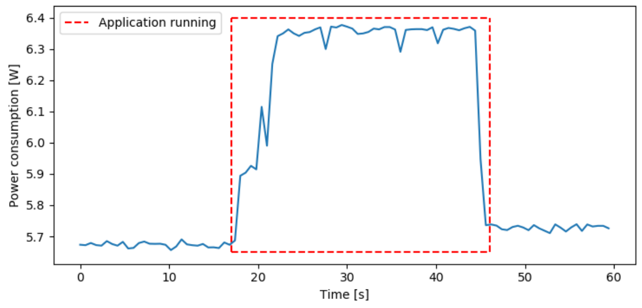

Fig. 8. Measured power consumption while running the application

The same bounding box neural network is tested running on the CPU of the Ultra96-V2, on a Raspberry Pi 3B+, and on an Nvidia Jetson Nano, all using Pytorch on Ubuntu 20.04. The performance metrics are compared in table I. The FPGA implementation's numbers include any performance impact ROS2 might have, whereas the other comparisons represent a best-case scenario with only inference running. The Finn compiler estimates a maximum throughput of 42 FPS for the accelerator itself, but the data loader core is not optimized and is likely the bottleneck.

Hardware	**Throughput**	**Peak power consumption**
Ultra96-V2 FPGA	25.0 FPS	6.4 W
Ultra96-V2 CPU	2.2 FPS	7.2 W
Nvidia Jetson Nano GPU	21.3 FPS	12 W
Nvidia Jetson Nano CPU	3.4 FPS	7 W
Raspberry Pi 3B+	1.6 FPS	6 W

TABLE I
Throughput and power consumption comparison

VIII. Conclusion

This work presented the MPSoC4Drones framework for orchestrating heterogeneous system development for the Ultra96-V2. The framework fully enables robotics and drone researchers to utilize the versatility and acceleration capabilities of FPGAs with standard tools of research, Ubuntu 20.04 and ROS2 as well as PX4 communication. A demonstration case was presented for validation of the framework. The example showcased hardware acceleration of an application in which the drone is commanded to arm its motors based on visual feedback supplied from an FPGA implemented object detection convolutional neural network inference core. The reported throughput and power consumption is shown to be competitive with conventional methods, thus making the case for MPSoCs for drone onboard computing.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111, Drones4Safety.

References

[1] FAA, "Fact sheet – the federal aviation administration (faa) aerospace forecast fiscal years," 2020, [Online; posted 26/03/2020]. [Online]. Available: https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=24756

[2] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury, and S. Liu, "A survey of fpga-based robotic computing," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 48–74, 2021.

[3] A. Bachrach, "Skydio Autonomy Engine: Enabling The Next Generation Of Autonomous Flight," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1–43.

[4] Xilinx, "Real-life benchmarks vs cpus and gpus," 2020. [Online]. Available: https://china.xilinx.com/content/dam/xilinx/publications/presentations/D1-02.pdf

[5] Drone Infrastructure Inspection and Interaction (DIII), "MPSoC4Drones," https://github.com/DIII-SDU-Group/MPSoC4Drones, 2022.

[6] Xilinx. Visited on 05/01/2022. [Online]. Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

[7] ——. Visited on 27/01/2022. [Online]. Available: http://www.pynq.io/

[8] O. T. O. K. Yamashina, K. and T. Yokota, "Proposal of ROS-compliant FPGA component for low-power robotic systems," *International Conference on Intelligent Earth Observing and Applications 2015*, 2015.

[9] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, and T. Yokota, "CReComp: Automated Design Tool for ROS-Compliant FPGA Component," *Proceedings - IEEE 10th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2016*, 2016.

[10] T. Ohkawa, K. Yamashina, T. Matsumoto, K. Ootsu, and T. Yokota, "Architecture exploration of intelligent robot system using ROS-compliant FPGA component," *Proceedings of the 2016 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, RSP 2016*.

[11] Y. Sugata, K. Ootsu, T. Ohkawa, and T. Yokota, "Acceleration of Publish/Subscribe Messaging in ROS-compliant FPGA Component," *ACM International Conference Proceeding Series*, 2017.

[12] T. Ohkawa, Y. Sugata, H. Watanabe, N. Ogura, K. Ootsu, and T. Yokota, "High level synthesis of ros protocol interpretation and communication circuit for FPGA," *Proceedings - 2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering, RoSE 2019*, 2019.

[13] D. P. Leal, M. Sugaya, H. Amano, and T. Ohkawa, "FPGA Acceleration of ROS2-Based Reinforcement Learning Agents," *Proceedings - 2020 8th International Symposium on Computing and Networking Workshops, CANDARW 2020*, 2020.

[14] H. Amano, H. Mori, A. Mizutani, T. Ono, Y. Yoshimoto, T. Ohkawa, and H. Tamukoh, "A dataset generation for object recognition and a tool for generating ROS2 FPGA node," *2021 International Conference on Field-Programmable Technology (ICFPT)*, 2021.

[15] D. P. Leal, M. Sugaya, H. Amano, and T. Ohkawa, "Automated Integration of High-Level Synthesis FPGA Modules with ROS2 Systems," *Proceedings - 2020 International Conference on Field-Programmable Technology, ICFPT 2020*, 2020.

[16] A. Podlubne and D. Gohringer, "FPGA-ROS: Methodology to Augment the Robot Operating System with FPGA Designs," *2019 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2019*, 2019.

[17] M. Eisoldt, S. Hinderink, M. Tassemeier, M. Flottmann, J. Vana, T. Wiemann, J. Gaal, M. Rothmann, and M. Porrmann, "ReconfROS: Running ROS on reconfigurable SoCs," *ACM International Conference Proceeding Series*, 2021.

[18] C. Lienen, M. Platzner, and B. Rinner, "ReconROS: Flexible Hardware Acceleration for ROS2 Applications," *Proceedings - 2020 International Conference on Field-Programmable Technology, ICFPT 2020*, 2020.

[19] Xilinx. Visited on 26/01/2022. [Online]. Available: https://www.xilinx.com/an/kria-robotics-stack.html

[20] V. Mayoral-Vilches, "Kria Robotics Stack A ROS 2-centric Approach for Hardware Acceleration in Robotics," 2021.

[21] V. Mayoral-Vilches and G. Corradi, "Adaptive Computing in Robotics Leveraging ROS 2 to Enable Software-Defined Hardware for FPGAs."

[22] B. Cain, Z. Merchant, I. Avendano, D. Richmond, and R. Kastner, "PynqCopter - An Open-source FPGA Overlay for UAVs," *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, 2019.

[23] E. Moreac, E. M. Abdali, F. Berry, D. Heller, and J. P. Diguet, "Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV," *Proceedings of the International Workshop on Rapid System Prototyping*, 2020.

[24] B. B. Kovari and E. Ebeid, "MPDrone: FPGA-based Platform for Intelligent Real-time Autonomous Drone Operations," *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2021.

[25] C. Maxfield, "Chapter 3 - the origin of fpgas," in *The Design Warrior's Guide to FPGAs*. Newnes, 2004, pp. 25–56. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780750676045500040

[26] S. Liu, Z. Wan, and B. Yu, *Robotic Computing on FPGAs*, 2021.

[27] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and B. Stewart, *Exploring Zynq MPSoC With PYNQ and Machine Learning Applications*, 2019. [Online]. Available: https://www.zynq-mpsoc-book.com/

[28] Xilinx. Visited on 02/09/2021. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[29] A. Developer. Visited on 16/02/2022. [Online]. Available: https://developer.arm.com/documentation/102202/latest/

[30] Farnell. Visited on 17/02/2022. [Online]. Available: https://www.electronics-lab.com/wp-content/uploads/2019/03/avnet_ultra96v2_block.jpg

[31] Avnet. Visited on 17/02/2022. [Online]. Available: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/ultra96-v2/

[32] Xilinx. Visited on 05/01/2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[33] Avnet. Visited on 05/01/2022. [Online]. Available: https://github.com/Avnet

[34] Xilinx. Visited on 26/01/2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[35] ——. Visited on 26/01/2022. [Online]. Available: https://github.com/Xilinx/finn

[36] Y. Project. Visited on 26/01/2022. [Online]. Available: https://www.yoctoproject.org/

[37] Xilinx. Visited on 27/01/2022. [Online]. Available: https://github.com/Xilinx/PYNQ

[38] D. Group. Visited on 08/09/2021. [Online]. Available: https://github.com/DIII-SDU-Group

[39] Drone Infrastructure Inspection and Interaction (DIII), "MPSoC4Drones," https://github.com/DIII-SDU-Group/MPSoC4Drones, 2022.

[40] ——, "Mp4d-ai-acceleration," https://github.com/DIII-SDU-Group/MP4D-AI-Acceleration, 2022.

[41] A. Pappalardo, "Xilinx/brevitas," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[42] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.