# HW-SW Management Using a Lightweight Yocto-based OS Running on a ZCU102

Raffaele Meloni[1], Giuseppe Meloni[2] and Daniel Madroñal[1]

[1]Università degli Studi di Sassari, Sassari 07100, Italy
[2]Abinsula Srl, Sassari 07100, Italy

Abstract

This work presents the development of a lightweight OS based on Yocto that has been extended to support the transparent management of HW accelerators. The accelerator is defined using the tool called MDC and, then, the co-processor (connecting the processor and the HW accelerator) is automatically generated together with a set of dedicated APIs to manage the accelerator.

This procedure is validated in the context of Comp4Drones ECSEL JU project, implementing and accelerating a soil segmentation algorithm on the ZCU102 FPGA architecture, reaching speedups up to 1.55× with real images from the artichoke field.

**Keywords**

Yocto, FPGA-based SoC, Embedded Systems, Precision Agriculture

## 1. Introduction

The increase of on-chip integration capabilities has allowed decades of improvements and performance boosts with respect to possible target technologies for digital designs. This trend has enabled the switching from standard computing to the embedded one, porting real-time solutions closer to the user [1] in almost every field, especially those in which unmanned vehicles are involved [2].

One of these fields is the precision agriculture, in which the use of drones and rovers has become a must, as stated by the European Drones Outlook Study distributed by SESAR JU [3]. This report presented the following problematic: by 2050 the food production will need to be doubled, whilst the available land will remain the same. To meet this requirement, smart farming and precision agriculture is considered an essential driver. The main limitation when implementing efficient solutions in this field, is the fact that the farmer themselves is still strongly involved in the processing flow. That is, normally, the drone/rover is just used to collect all the required information. Afterwards, this information is analyzed offline, where the farmer has to understand the data and make decisions accordingly [4]. Consequently, the use of embedded computing has been extended to not only controlling the auto-pilot, but also performing on-board complex operations to save time and increase drone/rover autonomy.

*CPS Workshop 2022, September 19, 2022, Pula, Italy*
r.meloni10@studenti.uniss.it (R. Meloni); giuseppe.meloni@abinsula.com (G. Meloni); dmadronalquin@uniss.it (D. Madroñal)
0000-0002-8173-6612 (R. Meloni); 0000-0001-5994-7440 (D. Madroñal)

This autonomy is thus considered one of the key enablers to boost the food production. In this regard, the project in which this work is contextualized (Comp4Drones ECSEL JU [5]) provides enabling technologies for real-time monitoring and accurate crop analysis. These capabilities are based on an architecture supporting self-adaptability and secure communications.

Another factor to be considered in this context is the energy budget limitation, therefore energy efficiency is sought also when selecting the target architecture. Among the embedded architectures, FPGAs [1] are normally considered a suitable option due to its great performance-energy trade-off [6, 7, 8]. Nevertheless, it is widely known that the management of such devices is not straightforward for the SW developers. On top of that, the FPGA needs to perform: 1) on-board processing using the programmable logic; 2) system control; and 3) synchronization with the autopilot module. In this regard, when used on top of a drone/rover, the FPGA requires an Operating System (OS) that links both the accelerated tasks and the autopilot subsystem.

These FPGA-based heterogeneous platforms are becoming widely adopted in the Cyber Physical Systems (CPS) context due to their ability to support dynamic and reactive behaviours while guaranteeing computing efficiency. An example in the context of precision agriculture is synthesized in Figure 1. The figure represent a CPS composed of two units, a drone and a rover that collaborate to detect, e.g., artichokes, requiring a specific irrigation action (carried out by means of a dedicated spraying arm, for example). In this context, the drone could be equipped with a camera, providing a video stream that is analyzed on the on-board FPGA, running a soil segmentation to detect the regions of the field occupied with artichokes. Later, this information is encrypted and sent to the rover through a dedicated edge gateway. Once this information arrives to the rover, the information is decrypted and merged with the output of a plant recognition algorithm so as to plan the path that the rover itself will follow. Finally, if an action is required for one or more artichokes, the rover will autonomously navigate the field to apply specific treatments.

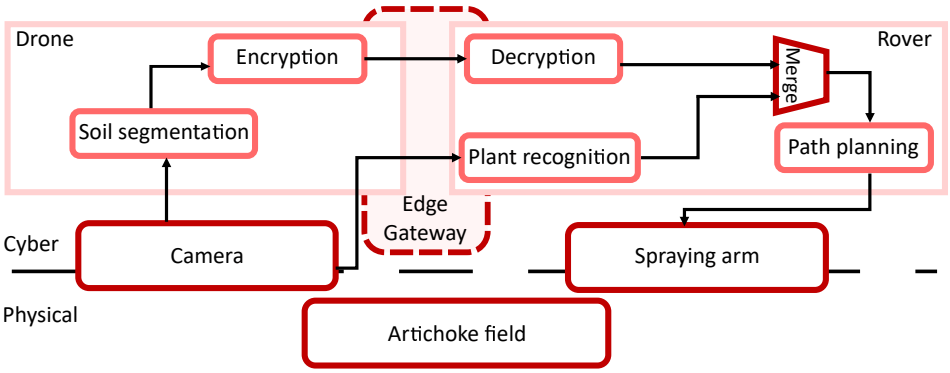


**Figure 1:** Precision agriculture scenario explained as a CPS

Starting from a scenario as the one explained before, this work aims at offering the support for the design and the management of a companion computer for online onboard processing. This paper presents the following novelties:

- Support of a lightweight embedded OS on the ZCU102 FPGA platform.

- Transparent management of HW accelerators from the SW, using the aforementioned OS.
- Analysis and acceleration of the soil segmentation algorithm that runs on the drone, which is used to validate the OS support on the target architecture.

The rest of the paper is organized as follows: first, section 2 presents an outline of the OS and of the adopted FPGA. Then, the details on how the OS has been adapted to run on the ZCU102 are described in section 3. After this, section 4 details the algorithm that has been used to validate the system, together with the validation itself. Finally, the future research ideas are briefly discussed in section 5.

## 2. Background

### 2.1. Yocto

Yocto is an Open Source project[1], part of the Linux Foundation, that helps developers create custom Linux-based systems, regardless of the hardware architecture. It provides a flexible set of tools and a space to share technologies, SW stacks, configurations and best practices that can be used to create tailored Linux images for embedded and IoT devices or anywhere a customized Linux OS is needed. The tools allow users to build and support customization for multiple hardware platforms and software stacks in a maintainable and scalable way.

The Yocto Project is based on a Layer model, to support collaboration, customization and reuse. Layers are repositories containing related sets of recipes (instructions) which tell BitBake (the system builder) what to do. Users can collaborate, share, and reuse layers, which can contain changes to previous instructions or settings. In fact, it is possible to create custom layers, including new or edited recipes of existing layers with new functionalities and suiting own product requirements.

In addition to customization, the Layer model allows to logically separate information using different layers. For example, a BSP layer, a GUI layer, a distribution configuration, middleware, or an application. Moreover isolating information into layers simplifies future customization and reuse: to add or remove functionalities the user just needs to add or remove layers.

In the following, the main steps to build a custom OS in Yocto[2] are presented, starting from developers specifications to OS image.

- Developers specify architecture, policies, patches and configuration details editing configuration files
- BitBake parses all recipes
- BitBake fetches and downloads the source code (with standard methods such as tarballs and git)
- Once downloaded, the sources are extracted and patches are applied, and steps for compiling software will be run
- The SW is installed into a temporary staging area and binaries are created
- A binary package feed is created and used to create the final root file image
- File system image is generated

[1] https://www.yoctoproject.org/
[2] https://docs.yoctoproject.org/3.4/ref-manual/

### 2.2. Target FPGA

The Zynq UltraScale+ MPSoC (ZCU102) by Xilinx[3] is an evaluation board based on a multiprocessor system-on-chip family consisting of a Processing System (PS) and a Xilinx Programmable Logic (PL) in a single device. In particular, the ZCU102 features an ARM cortex A-53 quad-core 64-bit application processing unit (APU) running at 1.5GHz and, an ARM CortexR5F real-time processing unit (RPU) and a Mali-400 MP2 (GPU).

Zynq US+ devices provide 64-bit processor scalability while combining real-time control hard engines for graphics, video, waveform, and packet processing capabilities in the PL. The PS acts as an standalone SoC able to boot and support all the features of the PS without necessarily powering on the PL. The chip allows the deployment of an embedded Linux-based OS (detailed in section 3).

It is worth highlighting that both PS and PL can be coupled with multiple interfaces to effectively integrate user-created hardware accelerators and other functions in the PL logic that can be used by the CPU, delegating part of the execution to it. Not only the PS can access its memory resources, but also the PL can do that. There are several units in the PL that have special wiring connections to the PS and the PL I/O pins to allow communication. The PL provides block RAMs, gates, clock structures, standard and high-range I/O, DSPs, and LUTs. The device includes several peripherals controllers and functional units like PCIe, Ethernet, DisplayPort and audio interfaces, among others.

## 3. Efficient HW-SW collaboration through Yocto-based lightweight OS on the ZCU102

### 3.1. Lightweight OS image creation

This section presents how a Yocto-based OS has been adapted to run on the ZCU102 platform. Specifically, the goal has been to create an embedded version of the OS that gathers the minimal required functionalities to allow the system to control the FPGA properly and communicate with the external world.

To do so, as detailed in [9] and graphically represented in Figure 2 (Yocto flow), the Yocto environment needs first to be configured to target the ZCU102 platform, the desired features are included through a set of dedicated recipes, and, to be able to load accelerators on the FPGA, the so-called `fpga-manager` needs to be added among the features to be considered in the final image. The HW accelerator is implemented using the Vivado standard workflow (named FPGA flow in Figure 2), having as an output the accelerator in a binary format.

Once the Yocto environment is properly configured, an image is generated using a `bitbake` command. In this case an image of around 29MB was obtained. The final step integrates the `fpgautil` tool that allows, among other control options, easily loading bitstreams.

[3] https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

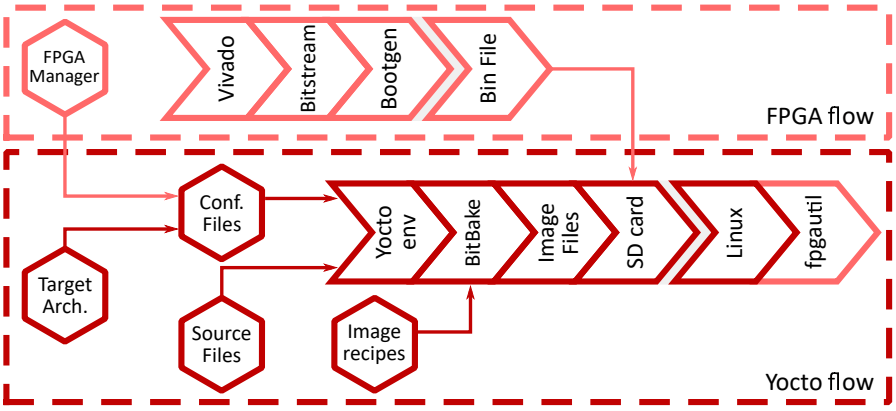


**Figure 2:** Steps to generate a Yocto-based OS with FPGA manager support

## 3.2. HW-SW communication management

Once the OS is ready to run on the architecture, the communication between PS and PL needs to be defined. To ease this step, the Multi-Dataflow Composer (MDC)[4] tool [10, 11] has been exploited. MDC automatically generates a co-processor system compatible with Vivado block design tool, in which inputs and outputs of the accelerator are connected to the PS through dedicated DMAs. After extending MDC, as detailed in [12], the tool is able to generate 1) the accelerator; 2) the co-processor infrastructure connecting PS and PL; 3) the scripts to generate the binary file of the system; and 4) the APIs to manage the accelerator from the SW application.

To exemplify the co-processor infrastructure, Figure 3 shows the block diagram that is generated when importing it into Vivado block design. As can be seen, each input/output of the accelerator (highlighted with a full-line square) is connected through a FIFO-DMA tandem (highlighted with dashed rectangles) managing the communication using a streaming strategy.

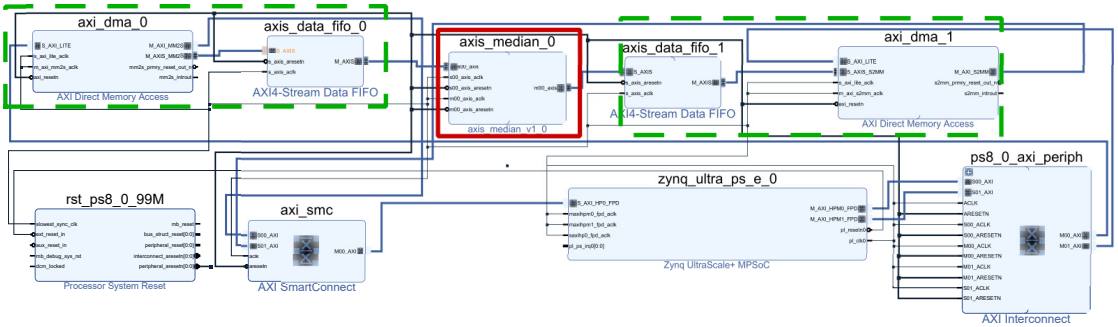


**Figure 3:** Co-processor generated with MDC to connect PS and PL and manage it from the Yocto-based OS. Accelerator with a full line and FIFO-DMA tandem in with a dashed line

[4] https://github.com/mdc-suite/mdc

# 4. Assessment

In this section, the system generation and HW accelerator management is evaluated in the context of Comp4Drones. Specifically, with respect to the scenario presented in Figure 1, the application that has been selected to be accelerated is the soil segmentation one.

## 4.1. Experiments

This application, originally developed in Python using `skimage` library, differentiates between soil with and without plants. First, it applies a block-based median filter, removing the noise that could interfere with the analysis; then, an Otsu threshold is applied to the median image, which output is used as a mask for the original image to separate pixels with predominant green contribution. Considering that the block-based median filter can be configured with different block sizes, to perform an in-depth analysis of the algorithm behavior, blocks of 8x8, 16x16 and 32x32 pixels have been tested.

To test the algorithm in a real scenario, images from the artichoke field (5472x3648 pixels) are employed, evaluating the applicability of the system within the Comp4Drones use-case.

## 4.2. Python application

This first experiment has been carried out on an AMD Ryzen 7 5800HS, running at 2.8GHz with 16GB of RAM memory. This test would be equivalent to perform the analysis of the images in an offline manner, where the images would need to be stored in the drone, downloaded on the ground station once the mission concludes, and analyzed using the ground station laptop. As can be seen in Table 1, the algorithm is divided into 5 steps, one per row in the table.

**Table 1**
Execution time (ms) of the soil segmentation application - Python version.

| Step | 8x8 | 16x16 | 32x32 |
|---|---|---|---|
| Median | 377.59 | 281.36 | 241.12 |
| RGB2YCBCR | 3.99 | 0.96 | 0.40 |
| Otsu | 3.13 | 0.81 | 0.46 |
| Resize | 338.08 | 343.26 | 351.23 |
| Masking | 232.25 | 271.89 | 193.60 |
| Total | 1244.08 | 1182.13 | 1068.80 |

As can be noticed, independently of the size of the block, the parts of the application with the largest execution times are: median filtering, resize and masking. Since the objective of this paper is to validate the methodology to generate and manage HW accelerators with the lightweight Yocto-based OS, the median filter has been selected as an example. This decision has been taken considering that applying a median filter is not limited to the selected soil segmentation algorithm, while the implementation of the other two is more application-dependent.

The in-depth analysis of the algorithm behavior demonstrates that, as can be observed in Figure 4, changing the size of the median block has no relevant impact in the soil segmentation

algorithm. As shown in the figure, using a block size of 32x32, the algorithm success in highlighting (in white) the parts of the image that can be associated with plants.

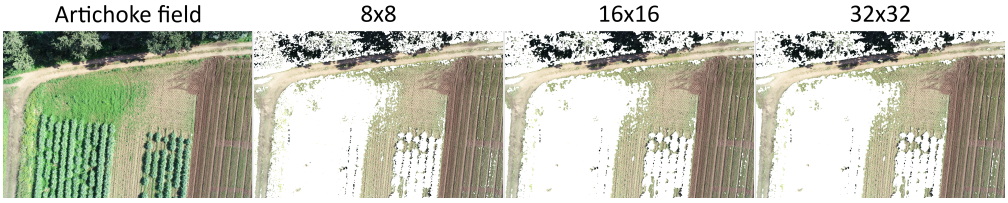


**Figure 4:** Artichoke field images segmented using the Python application.

### 4.3. HW porting and acceleration

Considering the results obtained in the previous section, it has been decided to accelerate the median filtering. In the Python implementation, the median filtering has been highly optimized, avoiding using any sorting. To be precise, the algorithm follows the recursive function explained in Listing 1.

This algorithm takes as input the pixels of the image (`imageArray`) and a candidate of the median (`pivot`). The algorithm divides the values of `imageArray` in those larger, equal and lower than the pivot. Then, it repeats the process on the larger/lower resulting array until the median is found, updating the pivot according to the maximum and minimum value of the selected array. Additionally, since the `position` of the median is already known: it is the value in the middle when the array is sorted (`imageArray.size/2`), its relative `position` in the subsequent arrays is updated with every iteration of the algorithm.

Finally, it is worth pointing out that, if the number of pixels included in the array is even, the median value will be the average of the two central pixels.

Listing 1: Median pseudo-code

```
Input: imageArray, pivot
Output: median
median(array, pivot, size, position){
    Variables: larger, lower, equal;
    for all values in array{
        if(value < pivot) include value in lower;
        else if (value > pivot) include value in larger;
        else include value in equal;
    }
    # The median is in the larger array
    if(larger.size > position){
        pivot: (max(larger) + min(larger))/2;
        position: size(larger) - position;
        median(larger,pivot,larger.size,position);
    }
```

```
    # The median is in the lower array
    else if (lower.size > position){
        pivot: (max(lower) + min(lower))/2;
        size: size(lower);
        median(lower,pivot,size,position);
    }
    # The median is in the equal array
    else
        return pivot;
}
main(){
    size: imageArray.size;
    position: imageArray.size/2;
    return (median(imageArray,pivot,size,position));
}
```

To analyze the soil segmentation behavior on the ZCU102, first, a C version has been implemented. Then, a HW accelerator featuring the median filter has been developed. This accelerator is divided in a set of sequential stages (that can be pipelined), where each stage correspond to a different recursive call in the Listing 1. This is possible considering that, since the beginning, the range of values of the different pixels composing the image is known. In this paper, `short` data type is used for the pixels composing the image, hence values from 0 to 255 are expected. As a result, 8 steps ($2^8$ is equal to 256) are implemented on HW, since, in the worst case scenario, 8 calls to the median function would be needed to discover the value of the median.

Table 2 gathers the results for the execution on the ZCU102. In this case, since only the median filter is executed on the PL, only values associated to median and total are provided for both SW and HW executions.

**Table 2**
Execution time (ms) and speedup HW vs SW (SW/HW times - ×) of the soil segmentation application - ZCU102 C in SW and C in SW+HW versions.

| Step | 8x8 | 16x16 | 32x32 |
|---|---|---|---|
| Median - SW | 14503.67 | 12285.92 | 11640.80 |
| Median - HW | 3792.46 | 3731.48 | 3676.55 |
| RGB2YCBCR | 83.05 | 20.43 | 4.90 |
| Otsu | 25.34 | 16.69 | 14.54 |
| Resize | 715.58 | 717.35 | 686.44 |
| Masking | 3180.43 | 3191.32 | 1803.17 |
| Total - SW | 30015.88 | 27662.55 | 25523.53 |
| Total - HW | 19304.67 | 18639.73 | 17571.62 |
| Speedup | 1.55× | 1.48× | 1.45× |

From this table, the values of the SW execution of the median algorithm need to be analyzed. When comparing, for example, the execution of the 32x32 kernel, it can be seen that, in C, the

value has risen to 11.64 seconds, while in Python it was 241ms. This is due to the reduced computational power of the ARM processor: 1.5GHz vs 3.4GHz of the previous one. Additionally, the available RAM memory and cache sizes are largely reduced in the FPGA.

Then, comparing SW with the HW acceleration, it is demonstrated that accelerating the algorithm using the methodology proposed in this paper (lightweight OS and PS-PL management) results in speedups up to 1.55×, even compensating the PS-PL data communication overhead. On top of this, it is worth pointing out that the system is automatically generated, including a set of dedicated APIs, easing the process of integrating HW accelerators within SW applications.

## 5. Future work

This work has presented the basis to create a lightweight OS based on Yocto, able to manage HW accelerators connected through an automatically generated co-processor infrastructure. This procedure simplifies the usability of these HW accelerators that, as a drawback, still need to be implemented by hand.

In this line, the main future works that are foreseen, starting from the one presented in this paper can be divided in: project-oriented and technology oriented. The former are:

- Implementation of the rest of the soil segmentation algorithm and, then, the rest of the UC on the ZCU102.
- OS connection with the autopilot system to work as a companion computer on-board of the drone/rover.

The latter, since the use of heterogeneous platforms in the CPS is in the hype, other future potential activities that are foreseen are:

- Extension and specialization of the proposed methodology to other fields like, for example, automotive.
- Extension of the OS to support SW and HW multithreading and multitasking, which is already already supported by the MDC tool.

## Acknowledgments

Authors would like to thank the EU commission for funding the ECSEL-JU Comp4Drones project under grant agreement no 826610.

## References

[1] S. Biookaghazadeh, M. Zhao, F. Ren, Are {FPGAs} suitable for edge computing?, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.

[2] 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.

[3] S. J. Undertaking, et al., European drones outlook study: unlocking the value for europe. (2017).
[4] L. F. Tiotsop, A. Servetti, E. Masala, An integer linear programming model for efficient scheduling of ugv tasks in precision agriculture under human supervision, Computers & Operations Research 114 (2020) 104826.
[5] 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.
[6] P. Gohl, D. Honegger, S. Omari, M. Achtelik, M. Pollefeys, R. Siegwart, Omnidirectional visual obstacle detection using embedded fpga, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015, pp. 3938–3943.
[7] M. Arakawa, Y. Okuyama, S. Mie, B. A. Abderazek, Horizontal-based Attitude Estimation for Real-time UAV control, Ph.D. thesis, MERAL Portal, 2019.
[8] M. Kowalczyk, D. Przewlocka, T. Kryjak, Real-time implementation of adaptive correlation filter tracking for 4k video stream in zynq ultrascale+ mpsoc, in: 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP), IEEE, 2019, pp. 53–58.
[9] R. Meloni, Yocto os on ultrascale+ zcu102, 2022. URL: https://mdc-suite.github.io/miscellaneous/yoctofpga.
[10] 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.
[11] N. Carta, C. Sau, F. Palumbo, D. Pani, L. Raffo, A coarse-grained reconfigurable wavelet denoiser exploiting the multi-dataflow composer tool, in: 2013 Conference on Design and Architectures for Signal and Image Processing, Cagliari, Italy, October 8-10, 2013, IEEE, 2013, pp. 141–148. URL: https://ieeexplore.ieee.org/document/6661532/.
[12] R. Meloni, Ps-pl communication management, 2022. URL: https://mdc-suite.github.io/miscellaneous/ps-pl-communication.