ePProcessor

D3.3

Initial porting of the use cases
Version 1.0

Document information

Work package	WP3: Software Applications Use Cases, Specifications and Evaluation
Contract number	956702
Project website	www.eprocessor.eu
Author(s)	Vasilis Flouris (FORTH)
Contributors	Lluc Alvarez (BSC), Arnau Bigas (BSC), Asaf Badouh (BSC), Jean-Marc Philippe (THALES), Christian Stollenwerk (UNIBI), Jens Hagemeyer (UNIBI), Nils Kucza (UNIBI)
Reviewer(s)	Stefan Krupop (CHR)
Dissemination Level	PU
Nature	O
Contractual deadline	31/07/2022

EuroHPC Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956702. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Sweden, Greece, Italy, France, and Germany.

Change Log

Version	Author(s)	Comments and Description of change
0.1	FORTH, BSC, UNIBI, THALES	Initial version with most of the technical content
0.2	FORTH, BSC, UNIBI, THALES	Reviewed version
1.0	FORTH, BSC, UNIBI, THALES	Final changes after internal review. Document reviewed. Final version

Executive Summary

This deliverable releases the initial porting of the application use cases and the microbenchmarks on the eProcessor architecture. This report starts by describing the main Software Development Vehicles (SDVs) that have been used to start porting the software, and then it describes the advances in the porting of the microbenchmarks and the application use cases. The structure of this document is as follows:

Chapter 2 presents the QEMU SDV, which is an emulator-based system that is able to run a complete Linux operating system, with libraries and applications on top of it.

Chapter 3 describes the FPGA SDV, which is an FPGA-based platform that runs the actual eProcessor core. Currently, this SDV can run simple bare-metal applications.

Chapter 4 details the initial porting of the eProcessor microbenchmarks on the FPGA SDV, including the required steps to adapt, compile and execute the microbenchmarks in this bare-metal environment.

Chapter 5, 6, 7, 8 and 9 discuss, respectively, the initial porting of the HPC, the Bioinformatics, the DeepHealth Toolkit, the Smart Mirror, and the Surveillance Border Control application use cases to the QEMU SDV. For each use case, its corresponding chapter describes the adaptations done in the application code and in the compilation scripts, the libraries that have been installed in the system to satisfy the dependencies of the applications, and the tests done to verify the correctness of the porting process.

The work reported in this deliverable has been performed by the WP3 “Software Applications Use Cases, Specifications and Evaluation” under Task 3.4 “Profiling, benchmarking, analysis, and tuning of applications use cases on the emulated RISC-V system”. This deliverable contributes to the project milestone MS3 “Single-core Tapeout: Dissemination and exploitation reports, Use-case applications, Single-core hardware and software (OS and tools for FPGA emulation, chip RTL freeze, and verification, FPGA emulation, Tapeout, PCB, first firmware and BSP release).”. The work performed in this deliverable has been carried out by the four partners that participate in Task 3.4. In particular, FORTH has contributed the FPGA and the QEMU SDVs; BSC has contributed the initial porting of the microbenchmarks, of the HPC use cases, of the Bioinformatics use cases, and of the DeepHealth Toolkit use case; UNIBI has contributed the initial porting of the Smart Mirror use case; and THALES has contributed the initial porting of the Surveillance Border Control use case.

Alongside this document, in this deliverable we do a software release of four files:

- A set of scripts (YAAFRV) that automate the process of setting up and invoking the QEMU SDV.
- A system image for the QEMU SDV that contains the operating system, the libraries required to run the application use cases, and the binaries and inputs of the application use cases.
- A snapshot of the repository of the application use cases that contains the source code of the application use cases.
- A microbenchmark that can be executed on the FPGA SDV.

The four aforementioned files that accompany this document are uploaded to the B2DROP repository of BSC, and can be accessed on the following link and password:

- Link: https://b2drop.bsc.es/index.php/s/KtkNDRjdneGsLbQ
- Password: 2PCz3fX9yN

Table of Contents

1. Introduction

The effort required to port an application into a different architecture can vary from trivial to very challenging. Some of the factors that contribute to the difficulty of the porting are:

- The application's dependence on specific hardware features (accelerators, sensors etc).
- The availability of hardware and emulators on which the software can be tested.
- The maturity of the software ecosystem of the target architecture.

One of the goals of the eProcessor project is porting a set of selected applications to utilize the eProcessor hardware resources as efficiently as possible. That means that the applications should both be tuned for the specific microarchitecture and make use of the accelerators of the platform. In this regard, the eProcessor application porting work is quite challenging, since it requires understanding and utilizing novel hardware and compiler features specifically designed for the eProcessor architecture.

Although the eProcessor ASIC is not yet available, some Software Development Vehicles (SDVs) are being developed in the project with the aim of starting the porting of the software as soon as possible. In particular, two SDVs are currently available: (i) the QEMU SDV, which can emulate a complete system including Operating System (OS), libraries and applications, and (ii) the FPGA SDV, which runs the real eProcessor core design, although currently it only supports bare-metal applications. These platforms are described in detail in later chapters. One of the key choices made for this porting effort is to have some provisioning for the transition between different SDVs, such that as little work as possible will be replicated while going from simpler to more complete platforms.

Regarding the maturity of the software environment of the RISC-V ecosystem, despite the significant progress in the past years, there are a lot of software components, libraries and tools that are still not available. For some context, the first functional version of the *Firefox* browser was announced less than a year ago, while *LibreOffice*, the first office productivity software available on RISC-V, was released a few days ago.

Given the above described challenges, limitations and choices, all eProcessor partners have progressed towards porting their applications and microbenchmarks on at least one of the SDVs. A summary of this progress is shown in Table 1 and is described in detail in the following chapters of the document.

	HiFive Unleashed & Unmatched boards	eProcessor QEMU SDV	eProcessor FPGA SDV	eProcessor ASIC
Microbenchmarks	ported	ported	partially ported	N/A
HPC use cases	ported	ported	N/A	N/A
Bioinformatics use cases	ported	ported	N/A	N/A
DeepHealth Toolkit	ported	ported	N/A	N/A
Smart Mirror	Not targeted	partially ported	N/A	N/A
Border surveillance	Not targeted	partially ported	N/A	N/A

Table 1: Progress of porting the application use cases to the different SDVs.

2. QEMU SDV

QEMU is a free and open source machine emulator and virtualizer that supports a very wide range of emulated target architectures, including RISC-V. In the eProcessor project we refer to the QEMU SDV as QEMU running a complete machine in emulation mode, that is, running a full Linux OS and applications on top of it. In Virtual Machine (VM) terminology this OS is called the guest OS. In this mode of operation, both the guest OS and the applications running require no knowledge of what is outside the VM.

This execution environment provides all ISA-specific interfaces for the functional porting of applications that do not require specific hardware components. In the case of RISC-V there are some caveats to providing such an environment that matches the eProcessor hardware design. One such detail is that, at the beginning of the project, there was no ratified version of the vector extension for RISC-V, and the version of the extension that was used for the vector processor has now been superseded by later ones. This requires a specific version of QEMU, other than the mainline, which implements the same version of the ISA vector extension as the one that is being implemented in the project.

For those reasons, and for simplifying and accelerating the work of the software engineers tasked with porting, we have developed a tool called *YAAFRV* (Yet Another Automation For Riscv-V), which automates the process of fetching and building all the required components of the QEMU SDV, and which automatically applies as many eProcessor specific changes as can be captured in this manner. The operation of YAAFRV is sketched in Figure 1.

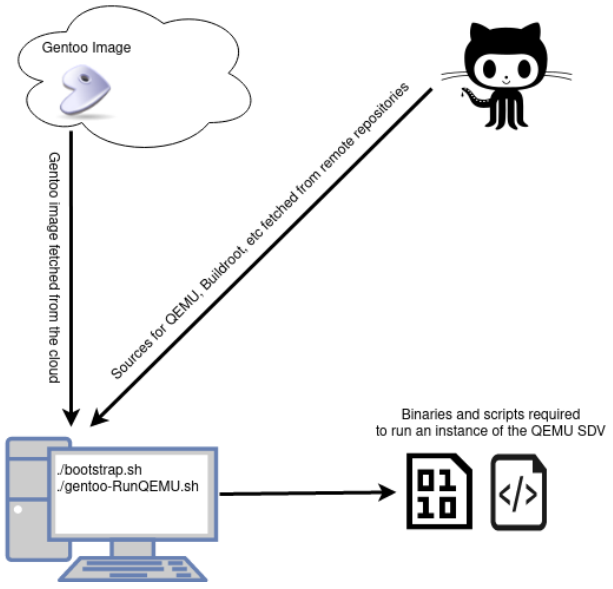

Figure 1: YAAFRV downloads, configures and builds all required components to instantiate the QEMU SDV.

Once YAAFRV is cloned, the user needs to perform the following steps:

[1] https://gitlab.bsc.es/eprocessor/ep-sw-reference/yaafrv

```
>./bootstrap.sh —clean-ubuntu # assuming a debian/ubuntu distribution
> # replace the smaller default gentoo qcow2 image with the one from b2drop
>./gentoo-RunQEMU.sh
```

The above steps should get the QEMU SDV running with the latest Gentoo system image used by the partners.

The Gentoo linux image, referenced above, has been built for the purpose of the initial porting work of eProcessor applications. One of the most important characteristics of this Gentoo image is that it is meant to be transferable between SDVs (e.g, from the QEMU SDV to the FPGA SDV). The fetching of this image is also handled by YAAFRV. Having and maintaining such an image means that, when the hardware design of eProcessor has matured enough so that the FPGA SDV is able to run a full Linux OS, the transition between the QEMU and the FPGA SDVs will be seamless, keeping the same Linux kernel, filesystem, system libraries, etc. This capability of moving the particular QEMU image into different RISC-V platforms has already been tested successfully.

After the installation of the emulation environment, some customization steps have been done to use the Gentoo linux image for software deployment:

- Resizing of the QEMU disk image
- Customization of the run script
  - 8 CPU cores utilization (maximum amount of 8 is hardcoded in QEMU)
  - 24 GB RAM utilization
  - Dynamic Memory ballooning to free unused memory on host demand
  - Forwarding of an SSH connection to enable connection for multiple users
- Configuration of Gentoo users and sudo permissions
- Configuration of the SSH service, including X tunneling, for remote connections
- Update of the Gentoo distribution
- Installation of software tools and dependencies via package manager

3. FPGA SDV

Although the QEMU SDV provides many of the architecture-specific interfaces required to port the eProcessor applications (i.e. user-space code), it does not expose to the software developer any of the eProcessor-specific system platform features. In particular, QEMU does not model microarchitecture, peripherals, system level organization, etc. Therefore, an SDV more closely approximating the eProcessor system platform is required, both for utilizing project-specific features and for understanding and tuning the performance of the applications.

In this deliverable we use the FPGA SDV developed in WP6 under the task T6.3 “FPGA emulation of the single core system”. The FPGA SDV has been recently released as part of the deliverable D6.3 “FPGA emulation of single-core system”. This FPGA SDV provides early access to the features of the eProcessor architecture, overcoming the aforementioned

limitations of the QEMU SDV. The use of an FPGA allows the rapid deployment of software to the latest hardware designs and enables the software engineers to access features as soon as they are ready (rather than having to wait for the ASIC implementation). This makes the FPGA SDV an ideal platform for porting software for such targets.

Although much progress has been made in the hardware components of eProcessor (specifically in WP5), at the time of preparing this deliverable the FPGA SDV does not yet have some of the critical features required to run a full Linux OS. Although the FPGA S DV is capable of running a variety of bare -metal applications, at the moment it is not yet a viable platform for porting and evaluating the full application use cases, as they rely on OS-managed functionalities.

Still, the currently available platform on the FPGA SDV is usable for porting the simpler microbenchmarks, as we demonstrate next. Through the process of porting, feedback on feature prioritization is given to the hardware designers, hence aiding in the progress and maturation of the overall system platform.

4. Porting of the Microbenchmarks to the FPGA SDV

A crucial aspect of the design of a processor is adjusting its microarchitectural parameters according to the requirements of the applications that the processor targets. However, in the early stages of the development of a processor, it is not feasible to use real applications or large benchmarks, because pre-silicon RTL simulations are extremely time consuming and different parts of the processor are designed and tested in isolation. As a result, the microbenchmarks used during this process have to stress only specific parts of the design while being representative of the applications of interest.

In the previous deliverable D3.2 "Microbenchmark suite to drive design decisions" we released the suite of microbenchmarks for the eProcessor project. These microbenchmarks are representative of the HPC, Bioinformatics and AI application use cases of interest for the project. The microbenchmarks target specific elements of the eProcessor architecture such as the CPU, the vector accelerator, and the AI accelerator, and they were initially tested on different SDVs such as the HiFIVE Unleashed and Unmatched boards, the Gem5 simulator, and the QEMU SDV.

For this deliverable we have focused on doing an initial porting of the microbenchmarks to the FPGA SDV. In particular, the microbenchmarks have been adapted to fulfill the specific requirements of the FPGA SDV, especially regarding the memory layout that is defined by the FPGA infrastructure. To this end, the following steps have been performed:

- Compiling the original benchmarks with a specific memory layout that is defined by the FPGA infrastructure.
- Using the ChopStiX[2] methodology to extract microbenchmarks from the new binaries. This generates microbenchmarks with the same memory layout as the original benchmark. Thus, the code and data sections that belong to the benchmark fulfill the requirements of the memory layout defined by the FPGA SDV. However, some code

[2] https://github.com/IBM/chopstix

and data sections belonging to the stack and to shared libraries cannot be mapped to specific memory addresses in the previous compilation step, so they do not fulfill the requirements of the memory layout defined by the FPGA SDV.

- Re-mapping the code and data sections belonging to the stack and to shared libraries to memory addresses that fulfill the requirements of the memory layout defined by the FPGA SDV.

After performing these steps, the resulting binaries can be successfully executed on the FPGA SDV and on the Spike RISC-V ISA simulator. Figure 2 shows the output of a microbenchmark extracted from the *Rank* function of the IS benchmark from the NAS benchmark suite running on the FPGA SDV. The microbenchmark shows a 'Hello World from eProcessor' message and then it executes the *Rank* function in a loop with a configurable number of iterations (2 in this example). For each iteration of the loop, the microbenchmark shows a 'Start' message, the contents of the *mcause* and the *mepc* registers (values 0 in the figure, which means the execution has finished without errors), an 'End' message, and the number of cycles spent in the execution of the function (94764086 and 58566086 cycles spent in the first and in the second iteration, respectively).

Figure 2: Output of a microbenchmark running on the FPGA SDV.

At the moment of writing this deliverable, we have successfully ported one microbenchmark extracted from the *Rank* function of the IS benchmark from the NAS benchmark suite. We are currently working on upgrading the ChopStiX infrastructure to automate the aforementioned steps that need to be performed to port the HPC and Bioinformatics microbenchmarks that were delivered in deliverable D3.2 "Microbenchmark suite to drive design decisions" to the FPGA SDV. Once we introduce the necessary changes in ChopStiX, we will re-generate the microbenchmarks and update the microbenchmark repository. This

will result in a set of 200 microbenchmarks that stress the CPU and the eAccelerator on the FPGA SDV and help drive microarchitectural design decisions.

Alongside this document we release a compressed file that contains the binary of the microbenchmark extracted from the Rank function of the IS benchmark from the NAS benchmark suite. The compressed file is available in the B2DROP repository.

5. Porting of the HPC Use Cases to the QEMU SDV

The NAS Parallel Benchmarks (NPB)[3][4] are used in the eProcessor project as HPC use cases. The NPB are widely used to evaluate the performance of high-end parallel systems. This benchmark suite consists of eight individual benchmark problems with special focus on computational aerophysics, although most of the benchmarks have much broader relevance, since they are typical of many real-world scientific computing applications. The eight problems consist of five kernels (CG, EP, FT, IS and MG) and three simulated Computational Fluid Dynamics (CFD) applications (BT, LU and SP). On one hand, the five kernels are relatively compact problems that emphasize a particular type of numerical computation. Compared to the simulated CFD applications, they can be ported fairly readily and provide insight as to the general levels of performance that can be expected on these specific types of numerical computations. On the other hand, the three simulated CFD applications usually require more effort to implement, but they are more indicative of the types of actual data movement and computation required in state-of-the-art CFD application codes, and in many other three-dimensional physical simulations, as well. By comparison, the simulated CFD applications require data structures and implementation techniques in three physical dimensions, and thus are more typical of real scientific applications.

For this deliverable we have ported the NPB to the QEMU SDV. To do that, we have downloaded the source code of the benchmarks, we have copied them in the system image, and we have compiled them normally. The compilation has not required any modifications to the source code of the benchmarks, nor any modification to the compilation scripts, nor the installation of any external library. To test the port is successfully working, we have executed all the benchmarks of the NPB with input class W and we have observed that they all finish with a successful result, as reported by the benchmarks.

5.1 Contents of the Use Case Repository and the System Image

Alongside this document we release the source code and the binaries of the whole NPB port. The source code can be found in the repository of the application use cases, in the `HPC` folder, which also contains instructions on how to compile and execute the benchmarks. The binaries of the benchmarks can be found in the system image, in the `/opt/npb` folder, which contains the binaries of all the benchmarks with input set classes S, W and A, as well as

[3] https://www.nas.nasa.gov/software/npb.html

[4] Bailey, David H., Eric Barszcz, John T. Barton, David S. Browning, Robert L. Carter, Leonardo Dagum, Rod A. Fatoohi et al. "The NAS parallel benchmarks summary and preliminary results." In Supercomputing'91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing, pp. 158-165. IEEE, 1991.

instructions on how to execute th e binaries. Both the repository of the application use cases and the system image are available in the B2DROP repository.

6. Porting of the Bioinformatics Use Cases to the QEMU SDV

In recent years, advances in next-generation sequencing technologies have enabled the proliferation of Bioinformatics applications that guide personalized medicine. These applications have an enormous computational cost due to the large amount of genomic data they process, so they have become a common workload in HPC systems and, at the same time, an important target for accelerators. In the eProcessor project, we use 4 Bioinformatics applications as use cases representative of different stages of typical genomic pipelines:

- The Smith-Waterman-Gotoh (SWG) is a dynamic programming algorithm that computes the pairwise alignment of two DNA sequences using affine gap penalties.
- The Banded Smith-Waterman-Gotoh (BSWG) is the banded version of the SWG algorithm, which implements a heuristic approach towards reducing the computational complexity of the original algorithm.
- The Wavefront Alignment (WFA) pairwise alignment proposes an alternative encoding of the dynamic programming matrix and an efficient algorithm to compute partial alignments with an increasing score. As a result, it computes the cells of the dynamic programming matrix by increasing score and only needs to compute a minimal number of cells to find the optimal alignment.
- The FM-index (Full-Text Index in Minute Space) search is one of the most common data structures used within aligners and metagenomics classification tools. This data structure is used in search algorithms to identify the exact-matching locations of short sequence substrings (called seeds) within a reference genome.

For this deliverable we have ported the Bioinformatics benchmarks to the QEMU SDV. To do that, we have downloaded the source code of the benchmarks and we have copied them in the system image. Then we have adapted the code to work on RISC-V, as the original source code of the benchmarks includes intrinsics to use x86 SIMD instructions. After removing the intrinsics we have been able to compile the benchmarks, without any modification to the compilation scripts nor installing any external library. To test the port is successfully working, we have executed all the benchmarks with the example inputs that the benchmarks incorporate and we have observed that they all finish with a successful result, as reported by the benchmarks.

6.1 Contents of the Use Case Reposito ry and the System Image

Alongside this document we release the source code and the binaries of the four Bioinformatics benchmarks. The source code can be found in the repository of the application use cases, in the `Bioinformatics` folder, which also contains instructions on how to compile and execute the benchmarks. The binaries of the benchmarks can be found in the system image in different folders, one per benchmark: `/opt/sw`, `/opt/bsw`, `/opt/wfa`, and `/opt/fmindex`. These four folders contain the binaries of the benchmarks,

input datasets, and instructions on how to execute the binaries. Both the repository of the application use cases and the system image are available in the B2DROP repository.

7. Porting of the Deep Health Toolkit Use Case to the QEMU SDV

The DeepHealth Toolkit[5] is a software ecosystem developed in the DeepHealth European project. Its main goal is to provide open-source libraries for AI and computer vision that can be leveraged by medical applications.

The European Distributed Deep Learning (EDDL) library is an optimized tensor library for distributed deep learning. The library is built around the concept of tensor and offers many functionalities with a device-independent interface, enabling a strong decoupling between the network training logic and the hardware implementation. This way, the EDDL library offers tailored implementations that can be efficiently executed on general CPUs, Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), or distributed systems.

The European Computer Vision Library (ECVL) facilitates the integration and exchange of data between existing state-of-the-art Computer Vision (CV) and image processing libraries. Moreover, it provides new high-level CV functionalities thanks to accelerated versions of some CV algorithms commonly employed in conjunction with deep learning algorithms. The algorithms of ECVL are adapted to hardware accelerators (i.e., GPUs and FPGAs) in a user-transparent way. To this end, the Hardware Abstraction Layer (HAL) hides hardware specific implementations of image manipulation functions, and the user only decides which device should be used for the computation and then calls a common Application Programming Interface (API) that offloads the work to the specified hardware devices.

The DeepHealth toolkit provides a set of use cases that apply deep learning techniques for automatic medical diagnosis. In the eProcessor project we use the Skin Lesion Classification use case, which targets improving melanoma diagnoses and reducing melanoma mortality by facilitating the application of digital skin imaging technologies. To do so, this use case performs a classification of sample images using a VGG16 deep learning model. The classification task has two parts, training and inference, and uses the International Skin Imaging Collaboration (ISIC) dataset. The ISIC 2019 dataset contains 25,311 images that add up to 9.8 Gigabyte of storage for training across eight different categories (i.e., classes). The dataset is divided into three subsets of images: 19,330 for training, 1,000 for validation, and 5,001 for testing. The dimensions of the images vary, but all the images are larger than 224×224 with three colors (i.e., channels).

[5] Cancilla, Michele, Laura Canalini, Federico Bolelli, Stefano Allegretti, Salvador Carrión, Roberto Paredes, Jon A. Gómez et al. "The DeepHealth Toolkit: a unified framework to boost biomedical applications." In 2020 25th International Conference on Pattern Recognition (ICPR), pp. 9881-9888.
IEEE, 2021.

7.1 Installed Dependencies and Tests

For this deliverable we have ported the DeepHealth toolkit to the QEMU SDV. To do that, we have downloaded the source code of the libraries and the use cases and we have copied them in the system image. Then we have adapted the compilation scripts to work on the QEMU SDV, as CMake was not detecting some C++17 libraries, although the backend compiler was detecting them. After fixing this issue we have installed the external libraries that are required by the DeepHealth Toolkit, which are specified in Table 2.

Name	Version	Status	Note
OpenCV	4.6.0	Installed	Installed via package manager
Eigen	3.4.0-r1	Installed	Installed via package manager
libpng	1.6.37-r2	Installed	Installed via package manager
openjpeg	2.5.0-r2	Installed	Installed via package manager

Table 2: Libraries needed by the DeepHealth Toolkit use case.

To test the port is successfully working, we have executed the skin lesion use case. To do so, we have copied the ISIC dataset in the system image and we have run the use cases with a subset of the whole dataset, including 200 images for training, 200 images for validation, and 200 images for testing.

7.2 Contents of the Use Case Repository and the System Image

Alongside this document we release the source code and the binaries of the DeepHealth Toolkit application use case. The source code can be found in the repository of the application use cases, in the `DeepHealth` folder, which also contains instructions on how to compile and execute the application. The binaries of the benchmarks can be found in the system image, in the `/opt/deephealth` folder, which contains the binaries of all the libraries, the binary of the application use case, the ISIC dataset, and instructions on how to execute the application. Both the repository of the application use cases and the system image are available in the B2DROP repository.

8. Porting of the Smart Mirror Use Case to the QEMU SDV

The smart home use case is centered around a smart mirror, which is based on the open source MagicMirror project, as introduced in deliverable D3.1 “Use cases definition, requirements and specifications reports”. The smart mirror supplements the user interface (UI) of an intelligent mirror with detection and recognition modules in order to show personalized information. Those detections and recognitions consist of object and gesture detection and face recognition, which are key in most smart home applications where resident interactions are required. The outline of the structure is shown in Figure 3.

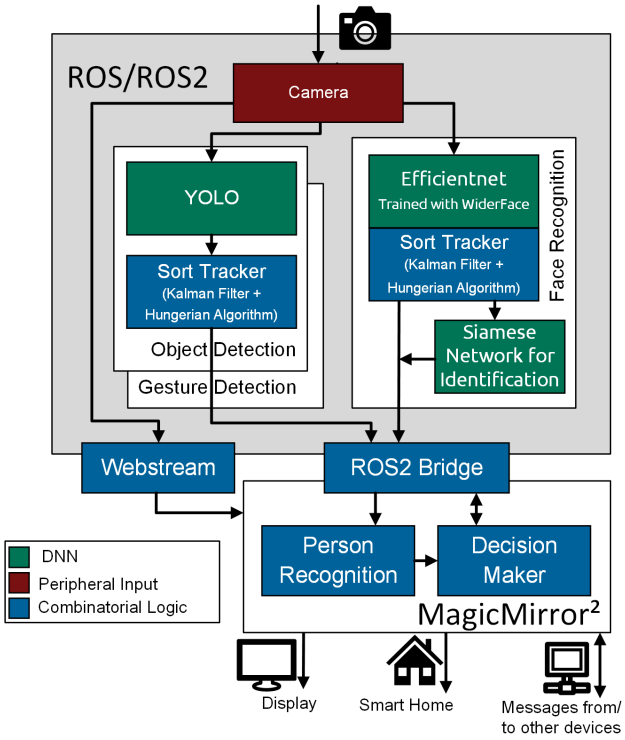

Figure 3: Overview of the Smart Mirror architecture.

The computation of the neuronal networks is curre ntly wrapped in ROS2 nodes, which are listening to an image topic, calculating the detections or recognitions, and publishing the results in JSON format to the output topic. This enables easier interchangeability and rapid prototyping. The MagicMirror appl ication is also equipped with a ROS2 interface and uses the published information. The gesture and object detections are based on YOLOv4 and implement tracking with the help of a Kalman Filter and a Hungarian algorithm (called sort algorithm). The face rec ognition is subdivided into two steps, the first step finds faces in images and also tracks them using a sort algorithm. The tracked faces are periodically published (every 500 ms) and recognized in the second step using a Siamese network and stored face embeddings of known users.

In order to port the Smart Mirror application to eProcessor, the approach is to first run all detections and recognitions on the developed accelerator, using it as a co -processor in the first step and subsequently work on all dep endencies to run the full smart mirror on the eProcessor. As a first step to this approach, the following subsection investigates which of the needed libraries are already available in the QEMU SDV and describes the basic tests for the different functional ities.

8.1 Installed Dependencies and Tests

In order to execute the smart mirror application, many software dependencies need to be installed. Table 3 shows the installation status of the required libraries. If a backend for neural networks and a middleware (in our case ROS2) is available, independent nodes for the required detections can be ported. The MagicMirror project is also in need of an interface for that middleware and some basic Node.js packages. Some of the required packages have been installed automatically and have not been affected by errors or issues, and therefore

are not listed. With the installation of TensorFlow and TensorFlow Lite, the first critical step of building the backend for the detection/recognition nodes is present. The next critica l step which has already started is the installation of a common middleware, which ROS2 is targeted for and will be evaluated next.

Name	Version	Status	Note
OpenCV	4.6.0	Installed	Installed via package manager
Rosdep	0.21.0	Installed	Installed via package manager and pip
ROS2	Humble	In progress	Built and installed without Qt GUI and unit tests
TensorFlow lite	2.9.1/master	Built and tested	Built via CMake/make. Built-in benchmark executable
OpenJDK	11.0.15	Installed	Installed via package manager
Bazel	5.0.0	Installed	Installed via package manager
TensorFlow	2.9.0	Built and tested	Built with Bazel 5.0.0. Some patches needed. First inference test done in python3 and C++
Node.js	18.4.0->16.15.1	Installed	Installed via package manager. Downgraded from 18.4.0 to 16.15.1 to evaluate some errors. Both versions are running but are having problems with the electron package.
Electron (Node.js)		Pending	Installable via Jode.js package manager, but build currently fails
MagicMirror		Pending	Depends on Node.js packages

Table 3: Libraries needed by the Smart Mirror use case.

8.1.1 TensorFlow as the Backend

The desired backend for the eProcessor accelerator is TensorFlow. Starting with the CPU version for testing purposes, the TensorFlow framework should be exchanged with a customized version, using the RISC-V architecture specific CPU features, as soon as it is available. TensorFlow version 2.9.0 has been built in the QEMU SDV by patching external dependencies where needed. Due to the emulation and usage of only 8 cores of the host system, which is the hardcoded maximum for QEMU, the building process took up to 15 hours in total. A python3 package has been installed and tested by running an inference benchmark. C++ libraries of Tensorflow have also been compiled and tested by running an object recognition inference. The current CPU only version shows poor performance, which is to be expected as it is running on an emulated environment.

8.1.2 ROS2 as the Middleware

The Smart Mirror framework is divided into functional frontend and backend modules. In order to use the ROS2 middleware for communication, the backend modules are implemented as ROS2 nodes. The ROS2 framework has a long list of dependencies, containing Rosdep and OpenCV among others, and is not officially supporting the RISC-V architecture in the current stable version “Humble Hawksbill”. Hence, some packages have to be customized to allow building ROS2 for the given architecture. One obstacle at the moment is the compilation of the mimick_vendor package, which generates assembler code for unit testing depending on the CPU architecture. Another problem at the moment is the compilation of the Qt GUI packages due to dependency issues. The rest of the ROS2 framework has been compiled and installed successfully.

8.1.3 Node.js and Dependency Packages

The Smart Mirror software framework uses the MagicMirror project as a frontend. MagicMirror is based on Node.js and depends on the electron browser for its GUI features. The desired version of Node.js has already been installed in the QEMU SDV but, at the time of writing this document, the build of the electron browser fails during the MagicMirror installation.

8.2 Contents of the Use Case Repository and the System Image

All relevant ROS2 nodes and installation scripts can be found in the repository of the application use cases, in the folder `SmartMirror`. The folder contains scripts for installing all the dependencies and the current software version of the Smart Mirror. Due to missing dependencies and time constraints, the Smart Mirror is not yet executable, but the contents of the repository shows the rough structure of the software. The installation scripts automate the process of installing all the dependencies in the system image, including preparing the environment, enlarging the image, and installing various packages such as Tensorflow.

The system image contains all the necessary installation to run the tests in the folder `/install-tmp/riscv-install/scripts`. TensorFlow and ROS2 have been compiled and installed within the system image. The built TensorFlow 2.9.0 C and the C++ libraries are installed in `/usr/local/lib` with the corresponding headers in `/usr/local/include`. A CMake module for TensorFlow has been installed on the system in order to integrate the libraries into a CMake build environment. Furthermore, the compiled python 3.9 package of TensorFlow has been installed via the python package manager. The ROS2 version "Humble Hawksbill", compiled without unit testing and Qt GUI, has been installed in the `/opt/ros/humble` directory.

The TensorFlow framework can be tested by executing the script `test-tensorflow.sh`. Testing the C++ API includes the download, compilation and inference of an object recognition example based on the "EfficientDet D3" model, provided by the TensorFlow model zoo and trained on the "COCO 2017" image dataset. In addition, the script tests if the TensorFlow python module can be imported successfully.

Both the repository of the application use cases and the system image are available in the B2DROP repository.

9. Porting of the Surveillance Border Control Use Case to the QEMU SDV

The Surveillance Border Control use case consists of using drones flying at a high altitude to capture and analyze high-resolution images along maritime borders to detect boats, as illustrated in Figure 4.

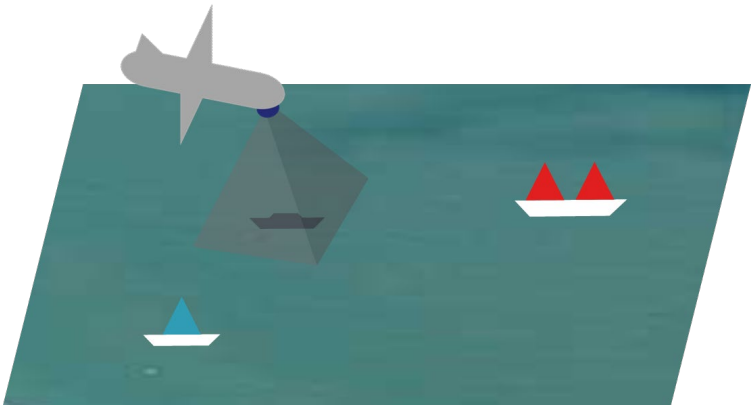

Figure 4: Illustration of a drone patrolling along a maritime border with boats to be detected.

The target system for the use case is the final version of the eProcessor platform involving the eProcessor chip connected to an FPGA -based off -chip CNN (Convoluti onal Neural Network) accelerator, which is responsible for the execution of the majority of the computations. For this deliverable, a mobile -scale state-or-the-art CNN topology has been considered for the initial port on the QEMU SDV. CNNs such as MobileNet-V1[6] are perfectly suited for low-power systems relying on software implementations. This deep neural network was introduced in 2017 with specific features to target embedded applications (new hyperparameters and types of layers).

The typical way of using CNNs is to train them in a supervised manner on annotated databases. In this deliverable we have used the MASATIv2[7] dataset. MASATI stands for MAritime SATellite Imagery dataset and consists of 7,389 images belonging to seven classes (land, coast, sea, ship, multi, coast-ship, and detail). The dataset was built by annotating pictures gathered from the Microsoft Bing Maps website. Then these pictures were labeled using the LabelImg[8] tool.

The MASATI dataset can also be used to illustrate boat detection by merging the seven existing classes into two main classes (Ship and Non-Ship). The Ship class comprises ship, multi, coast-ship and detail sub-classes, while the Non-Ship class is made of land, coast and sea sub-classes. This new partition results in a dataset comprising 4157 images for the Ship class and 3232 images for the Non-Ship class. Figure 5 shows a sample of the MASATIv2 dataset with a ship not far from the coast. Thus, the main class of this sample is "Ship" and the sub-class in "coast-ship".

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." CoRR abs/1704.04861 (2017)

[7] MASATI dataset (v2) - MAritime SATellite Imagery dataset, https://www.iuii.ua.es/datasets/masati/, Gallego, Antonio-Javier, Antonio Pertusa, and Pablo Gil. 2018. "Automatic Ship Classification from Optical Aerial Images with Convolutional Neural Networks" Remote Sensing 10, no. 4: 511. https://doi.org/10.3390/rs10040511

[8] https://github.com/tzutalin/labelImg

Figure 5: Example of a picture containing a ship not far from the coast (from MASATIv2 dataset).

9.1 Installed Dependencies and Tests

For the surveillance border control application use case, the N2D2 (Neural Network Design & Deployment[9]) open source framework has been used to prototype the implementation. This tool is a complete AI framework enabling to build embedded applications based on deep neural networks (DNNs), from the training phase to the deployment on the target, since it is able to generate native parallel source codes of the supported deep neural networks. For this deliverable, N2D2 has been used to generate the native C++ source code of the DNN used for boat detection. The generated source code is already parallelized using OpenMP pragmas. Figure 6 shows the flow of the source code generation using N2D2.

[9] https://github.com/CEA-LIST/N2D2

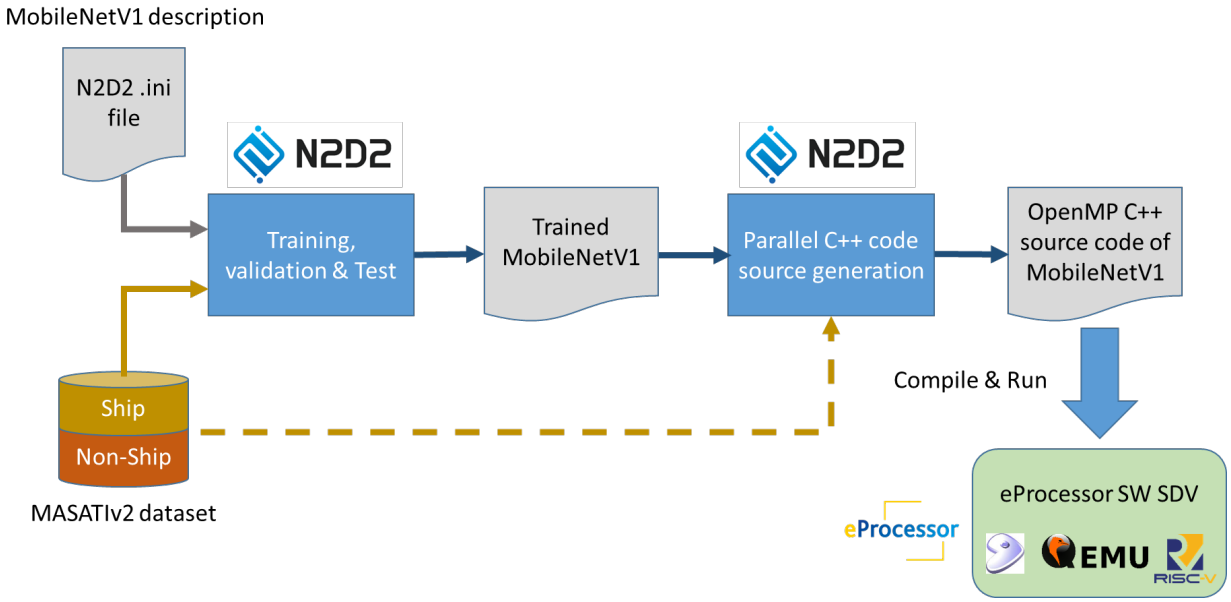

Figure 6: Flow of MobileNetV1 source code generation using N2D2, including the application deployment on the QEMU SDV.

An input file describing the CNN (MobileNetV1 for t his particular implementation of the use case) has been used in the form of a specific .ini file (native file format for N2D2) to perform the training phase (including validation and test sub -steps). The provided dataset is MASATIv2 in the form of Ship/Non Ship partition. The most complex version of the MobileNetV1 network has been chosen, with the "alpha" parameter set to 1.0. An accuracy of ~95% has been reached on the test dataset. Then, the resulting CNN and its parameters have been exported using the N 2D2 framework. The CNN export process consists in generating the C++ source code of the neural network together with quantized weights. For the purpose of this deliverable, 16-bit integer post-training quantization has been performed on the trained MobileN etV1 CNN.

The resulting files have been modified to include eProcessor use case specific features, and then they have been transferred to the QEMU SDV. The source code does not have dependencies with any external library, and the compilation only requires an appropriate gcc/g++ compiler with support for OpenMP. A slight modification to the Makefile generated from N2D2 has been required to change the " --march" option to a value that is compatible with the RISC-V toolchain.

The resulting binary application is able to recognize normalized pictures containing boats in a database or a specific picture provided to it by the "-image" parameter. Figure 7 shows the execution of the binary on the test dataset on the QEMU SDV.

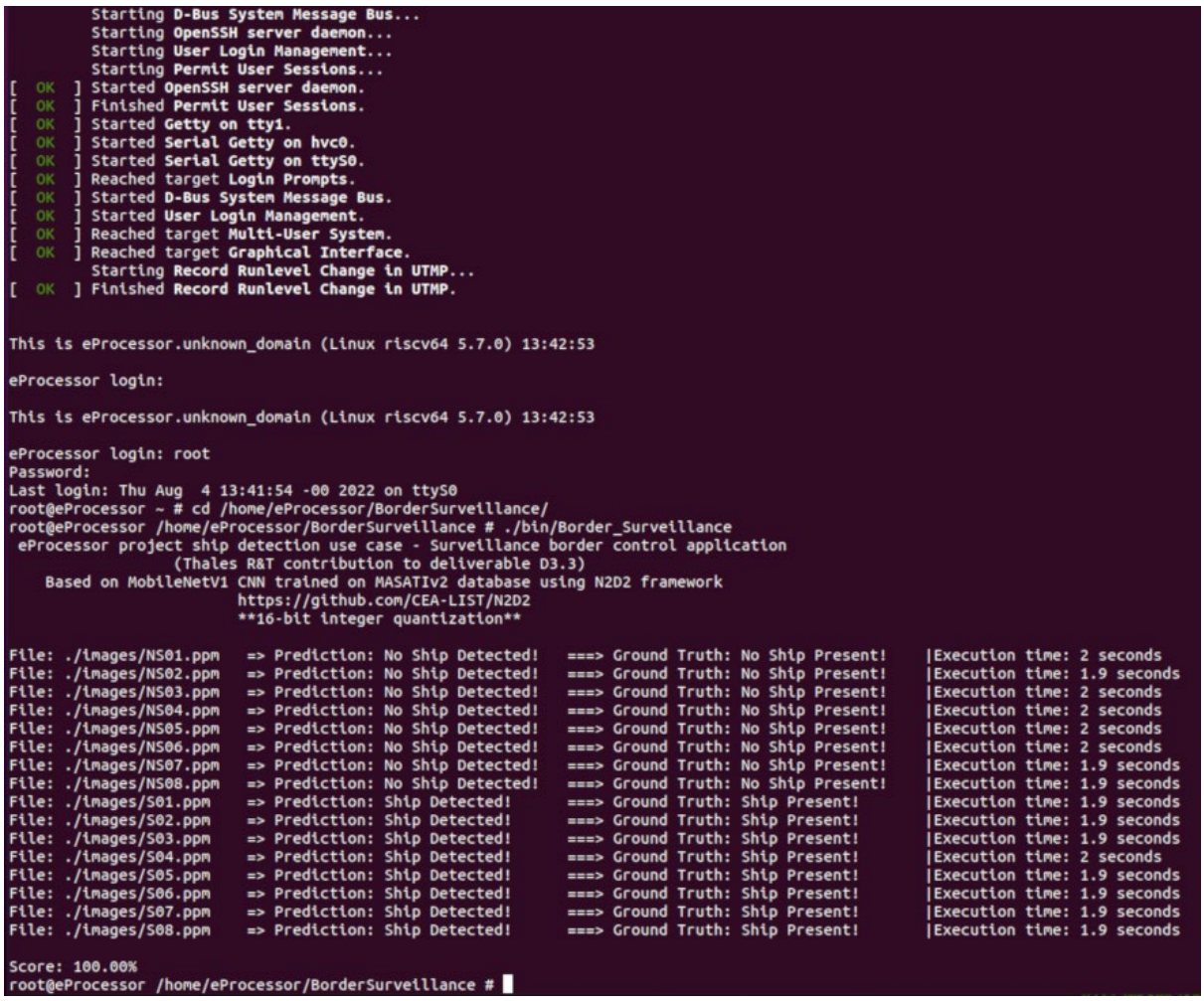

Figure 7: Execution of the Surveillance Border Control use case on the QEMU SDV.

9.2 Contents of the System Image

Alongside this document we release the binary of the Surveillance Border Control application. The binary and a set of images for testing can be found in the system image, in the `/opt/BorderSurveillance` folder. The system image is available in the B2DROP repository.

10. Conclusions

This deliverable describes and releases the initial porting of the eProcessor use cases and microbenchmarks on the eProcessor architecture. The initial porting has been performed on the two SDVs that are being developed in the project, namely the QEMU SDV and the FPGA SDV.

On the one hand, the FPGA SDV allows running simple bare-metal applications on the actual eProcessor core that is being developed in the project. Given the current constraints and time limitations, we have ported one microbenchmark and we have demonstrated that it can be successfully executed on the FPGA SDV. After doing this initial porting effort, the next

steps are to re-generate all the microbenchmarks, run them in the FPGA SDV, analyze their performance, and provide feedback to the hardware developers to tune microarchitectural parameters and guide design decisions.

On the other hand, the QEMU SDV allows emulating a complete system, including a full Linux OS, libraries and applications. Given the requirements of the application use cases, we have chosen this SDV to start their porting. At this initial porting stage, we have built a fully functional system image that contains all the libraries required by the applications, we have adapted the codes and the compilation scripts of the application use cases, and we have been able to successfully run the first tests. The next steps will consist of finishing the porting of the applications use cases that have only been partially ported, migrating the system image to the FPGA SDV when it has full system support, running the application use cases on the FPGA SDV, analyzing their performance, and providing feedback to the hardware developers to tune microarchitectural parameters and guide design decisions.

The work reported in this deliverable has been performed by the WP3 “Software Applications Use Cases, Specifications and Evaluation” under Task 3.4 “Profiling, benchmarking, analysis, and tuning of applications use cases on the emulated RISC-V system”. This deliverable contributes to the project milestone MS3 “Single-core Tapeout: Dissemination and exploitation reports, Use-case applications, Single-core hardware and software (OS and tools for FPGA emulation, chip RTL freeze, and verification, FPGA emulation, Tapeout, PCB, first firmware and BSP release).”. The work performed in this deliverable has been carried out by the four partners that participate in Task 3.4. In particular, FORTH has contributed the FPGA and the QEMU SDVs; BSC has contributed the initial porting of the microbenchmarks, of the HPC use cases, of the Bioinformatics use cases, and of the DeepHealth Toolkit use case; UNIBI has contributed the initial porting of the Smart Mirror use case; and THALES has contributed the initial porting of the Surveillance Border Control use case.

Alongside this document, in this deliverable we do a software release of four files:

- A set of scripts (YAAFRV) that automate the process of setting up and invoking the QEMU SDV.
- A system image for the QEMU SDV that contains the operating system, the libraries required to run the application use cases, and the binaries and inputs of the application use cases.
- A snapshot of the repository of the application use cases that contains the source code of the application use cases.
- A microbenchmark that can be executed on the FPGA SDV.

The four aforementioned files that accompany this document are uploaded to the B2DROP repository of BSC, and can be accessed on the following link and password:

- Link: https://b2drop.bsc.es/index.php/s/KtkNDRjdneGsLbQ
- Password: 2PCz3fX9yN