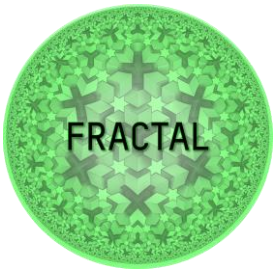

Deliverable

D3.4 Software node and services architecture

Deliverable Id:	**D3.4**
Deliverable Name:	**Software node and services architecture**
Status:	**Approved**
Dissemination Level:	**Public**
Due date of deliverable:	**28.2.2022**
Actual submission date:	**31.3.2022**
Work Package:	**WP3**
Organization name of lead contractor for this deliverable:	**Offcode Oy**
Author(s):	**Antti Takaluoma**
Partner(s) contributing:	**Alexander Flick, PCL2** **Jérôme Quévremont, Thales** **Kévin Eyssartier, Thales** **Jaume Abella, BSC** **Edurne Palacio, Ikerlan** **Lauri Loven, UOULU** **Leticia Pascual, Solver (SML)**

Abstract: This deliverable (D3.4) is the second of a series of deliverables that describe the software work for the FRACTAL project software nodes. This is the second of three deliverables on software node, and it will be updated throughout the project with D3.2 (M12), and D3.6 (M20).

ECSEL Joint Undertaking
Electronic Components and Systems for European Leadership

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056

Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.

Contents

History

Version	Date	Modification reason	Modified by
0.12	30.3.2022	Final formating changes	Antti Takaluoma
0.11	30.3.2022	Updates based on internal review and various optimization to document.	Antti Takaluoma
0.10	11.3.2022	Moving chapters	Antti Takaluoma
0.9	1.3.2022	Ready for review by Matti Vakkuri and Igor Bisio	Wp3 meeting
0.8	28.2.2022	Proposal to be approved	Antti Takaluoma
0.7	28.2.2022	Reviews and updates	Alexander Flick
0.6	25.2.2022	Added "big pictures"	Jaume Abella (BSC)
0.5	23.2.2022	Reviews and updates	Jérôme Quevremont
0.4	17.2.2022	Proposal, prepared to project internal review	Antti Takaluoma (OFFC)
0.3	31.12.2021	Draft, filled D3.2 data to new structure, yellow text to be revisited	Antti Takaluoma (OFFC)
0.2	30.12.2021	Draft, adding content	Antti Takaluoma (OFFC)
0.1	29.12.2021	Draft, basic structure revisited	Antti Takaluoma (OFFC)

1 Summary

The main objective of the FRACTAL project is to "create a cognitive edge node enabling a fractal Edge that can be qualified to work under different safety-related domains". Furthermore, it is stated in the DoA that "This computing node will be the basic building block of intelligent, scalable and non-ergodic IoT". As such the hardware node is a central part of the FRACTAL project around which 28 partners collaborate, investigate and industrial partners develop their use cases.

This deliverable (D3.4) is the second of a series of deliverables that describe the software work for the FRACTAL project hardware nodes. These documents will be delivered throughout the project with D3.2 (M12), D3.4 (M18), and D3.6 (M20). These three deliverables are also paired with the "hardware node and services" deliverables D3.1, D3.3 and D3.5.

The FRACTAL project brings together many partners (28) both from industry and academia, working on varied and challenging topics as well as eight industrial use cases. It was already a challenging task to provide a set of solutions for the hardware node in this context and combined with restrictions around COVID and worldwide supply disruptions for electronic components, partners in WP3 had to face additional challenges.

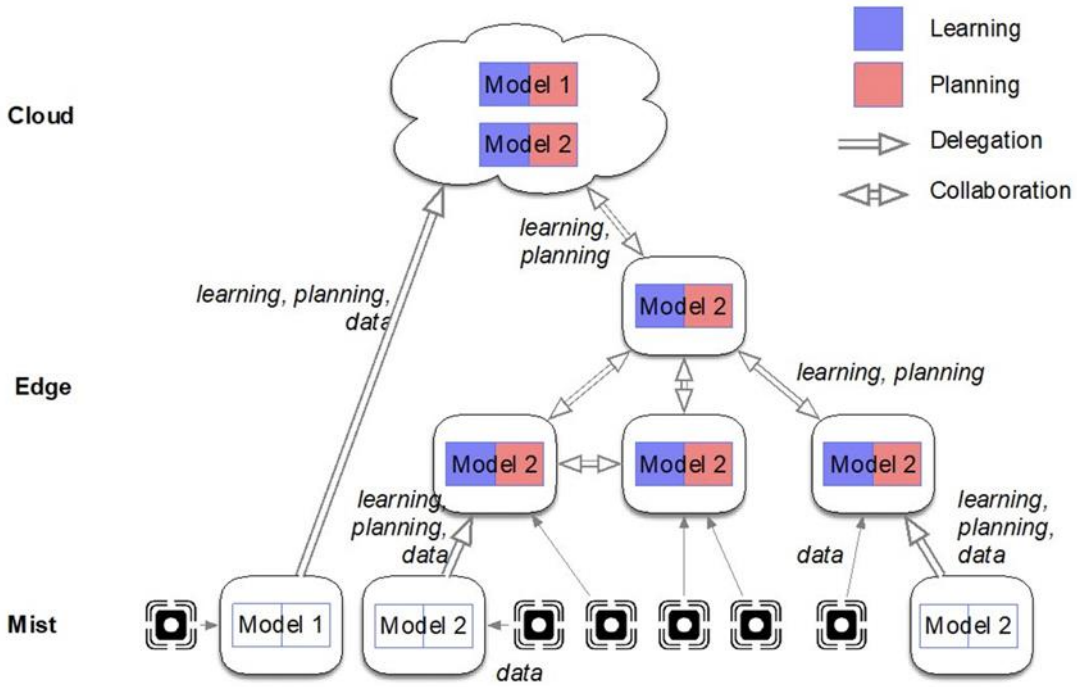

Figure 1. A schematic drawing of a possible FRACTAL system deployment using three different tiers of FRACTAL hardware nodes with different capabilities (drawing from WP5 technical meetings).

In the project the following options for the hardware nodes were used:

1. **Edge node** based around the Xilinx VERSAL ACAP (Adaptable Compute Acceleration Platform)
2. **Low-end node** (also as mist node) -- based around the open-source RISC-V based PULP platform

Additionally, some specific platforms (e.g., Ariane/CVA-6, NOEL-V) are used to demonstrate some specific technology concepts and use cases.

Note, in D3.2 also terms **customizable node** and **commercial node** were used. For now, on the Fractal domains are described by terms: **cloud**, **edge node** (Versal) and **low-end node** (Pulp). In case of cases where other platforms were used, their context will be clarified on text.

The organization of the deliverable is as follows.

Chapter 2 provides a general introduction to the Fractal framework and the stake holders using it. Chapter 3 summarizes WP3 technical relations to other WPs. Chapter 4 looks WP3 requirements for the Pulp platform point of view. Chapter 5 looks WP3 requirements for the Versal platform point of view. Chapter 6 introduces the special WP3 cases that were demonstrated by other platforms. Chapter 7 summaries and referates the Fractal use cases.

2 Introduction

FRACTAL is a system that offers a framework for developing modern distributed applications. Distributed applications can be executed in embedded nodes -- near the individual processes, centrally at the cloud, or as distributed into both of these domains. The supporting Framework should offer seamless connectivity, application integrity and the required security and safety. As addition the FRACTAL framework offers also an integrated AI tools also to the domains. Additionally, FRACTAL offers application specific hardware accelerations to the embedded nodes.

In the project proposal, we identified four strategic objectives of FRACTAL to reach this goal:

- **Objective 1**: Design and Implement an Open-Safe-Reliable Platform to Build Cognitive Edge Nodes of Variable Complexity. This part is mainly being addressed as part of WP3.
- **Objective 2**: Guarantee extra-functional properties (dependability, security, timeliness and energy-efficiency) of FRACTAL nodes and systems built using FRACTAL nodes (i.e., FRACTAL systems), which has determined the tasks of WP4
- **Objective 3**: Evaluate and validate the analytics approach by means of AI to help the identification of the largest set of working conditions still preserving safe and secure operational behaviors, which is the topic of WP5
- **Objective 4**: To integrate fractal communication and remote management features into FRACTAL nodes, which will be covered by WP6.

Looking outside, FRACTAL can be seen from various points of view. Most important are the end-user view and the application owner views.

2.1 Application owner view

Applications are developed according to the developer's business logic. Initially the business owner will assume that on the market the application will offer an added value to the end customer(s). To benefit from this added value, business owners need strategies to enter the market and secondly keep and improve this position.

2.1.1 Support for business logic

Framework should offer freedom to implement various application scenarios according to their business opportunity. While the business case (market) develops the application developments should be easy to deploy.

Main purpose of FRACTAL framework is to minimize this work, without limiting too much the application developer freedom.

Fractal Use Cases described at D3.3, chapter 5.

2.1.2 Support for development and testing

Framework should offer tools for actual development, testing and verification. Mainly these tools are "standard" development tools, but the Framework itself should have specific tools to identify (and prevent) unwanted behavior of application specific components.

While in some extreme cases embedded electronics are custom developed, in most cases the framework should offer seamless hardware acceleration.

Yet another important aspect of distributed applications is application integrity. All parts of the application must be consistent with each other. Framework should offer tools for safe application deployment and ensure that parts of the distributed application are genuine.

2.1.3 Support for commissioning, deployment and provisioning

When the end-users are attracted by the business logic, the application is ramped to a specific end-user. This may require physical installations on site and/or configurations to the cloud. Application logic – by support of the framework – should distribute the configurations to end-user specific nodes.

Some cases there may be needs to collect end-user specific information – e.g., billing of further marketing needs – this information transfer must be secure.

2.2 End-user view

For the end-user (or the customers of the end-user) the distributed application integrates directly to their processes (**Fractal Use Cases described at D3.3, chapter 5**). Depending on the application, availability, operational safety, and information security are typically important aspects.

3 WP3 and related WPs

WP3 is focusing on development of nodes – both the physical HWs and the necessary firmware. There are VERSAL based high-end nodes and low-end PULP based nodes. Both platforms have toolsets for application software and HW acceleration.

3.1 WP3 and WP4

The WP4 looks at the FRACTAL framework from point of safety -- how to manage physical defects on mechanics and electronics and the exceptional cases on software. Thus, these are highly related both to node hardware and node firmware, WP4 is deeply related to both WP3 nodes. Some exceptional cases are not necessarily possible to demonstrate on these platforms, so special platforms may be used.

3.1.1 Supporting FRACTAL developments on safety

Today safety is mainly based on process and system assessments. As such, it is not a plain software feature, but more like process and documentation issue.

3.1.2 Supporting FRACTAL developments on security

Two approaches are differentiated in the developments covering security related features.

3.1.2.1 Linux based systems

Linux offers good tools for security. With HW support those can be strengthened to meet the requirements derived from specific use cases.

Additionally, for the VERSAL node (Linux based system as well), those use case applications that require device-level security could implement boot image encryption and authentication, functionalities that are natively supported by VERSAL.

3.1.2.2 RTOS based systems

By nature, RTOS based systems have little native security features. With dedicated HW support those can be strengthened to meet the requirements, but in most cases security features are application specific implementations.

3.1.3 Supporting FRACTAL developments on low power

Two approaches are differentiated in the developments related to low consumption features.

3.1.3.1 Linux based systems

Linux offers good tools for low power operations. For further needs the RTOS can be utilized. RTOS runs in additional processor or preferably one of the system cores. When low power requires Linux to switch itself off, it yields the responsibility to the RTOS. RTOS keeps processing events and, when defined conditions are met, it wakes up the Linux.

Additionally, for the VERSAL node (Linux based system as well), as shown in Figure 6 a centralized Platform Management Controller (PMC) that handles device management control functions is available. A flexible management control could be done through this PMC. This platform management handles several scenarios and allows the user to execute power management decisions through its framework (equivalent to what it is done in Linux, which provides basic power management capabilities like CPU frequency scaling).

However, some limitations apply. Because of the heterogeneous multi-core architecture of VERSAL, individual processors can't make autonomous decisions about power states of individual components or subsystems. Instead, a collaborative approach is taken, where a power management API delegates all power management control to the platform management controller. This PMC is the key component in coordinating the power management requests received from the other processing units, and the coordination and execution from other processing units through the power management API. This framework manages resources such as power domains, power islands, clocks, resets, pins and their relationship to CPU cores, memory, and peripheral devices.

Therefore, the natively provided power management API would be used for VERSAL node, since this platform management framework abstracts the complexity associated to administrate the power-management of a multiprocessor heterogeneous system.

3.1.3.2 RTOS based systems

By nature, RTOS based systems offer good tools low-power operations. The RTOS level low-power features are typically extended with specific support by the underlying HW (processor).

Another additional layer of low power is typically obtained by application architecture. Event-based application structure is by nature easier for low-power than applications that are coded based on infinity loops, however this is not in scope of framework.

3.2 WP3 and WP5

WP5 focuses on integrating AI to the Fractal framework. While AI is in the scope of whole project, the WP3 has some special concerns related to implementation in the low resource environment.

3.2.1 Supporting FRACTAL developments on AI

Several alternative methods are studied for deploying AI/ML models on FRACTAL nodes. First, a model may be pre-built, that is, trained by a third-party actor, downloaded from a public repository, and uploaded to the node. Second, a model may be learned from data available to a node, possibly augmented with annotations which indicate the expected model output for each data point. Third, a few nodes may co-operate to train a model, e.g., with a federated learning approach.

In each case, the model may need updating due to model drift, that is, the accuracy of the model output slowly degrading. In such cases, new training data must be collected

(and possibly annotated), and the model updated to reflect the data. Further, the model update cycle must be managed such that model quality is monitored and update launched when necessary. Tools and methods conducting model lifecycle management are commonly referred to as MLOps.

Inference-time, when the model is turning input data into model output, federated approaches may improve the quality of the outputs in some use cases. For example, if a number of nodes each employ an independently trained (i.e., with different data) but otherwise identical models, the models may be used as an ensemble, with the same input data fed to all of them, and the results combined into one.

WP5 is studying all above approaches in close co-operation with WP3, focusing on theoretical study of distributed learning and inference, the FRACTAL cloud platform, the architecture and orchestration of the FRACTAL network, as well as the AI methods required to fulfill the requirements of the use cases.

3.2.2 LEDEL to develop and execute AI-based models in a FRACTAL node

In the Figure 4 we can observe the scope of the task in WP3 in the context of LEDEL in the FRACTAL project, which is the adaptation of the EDDL to become LEDEL. Such adaptation consists of compiling the EDDL in a RISC-V platform, reassuring all the libraries and dependencies that it needs are also available and fully functional in the reduced instruction set architecture. The platform chosen for this aim is NOEL-V. This platform is scheduled to be available before the end of the year.

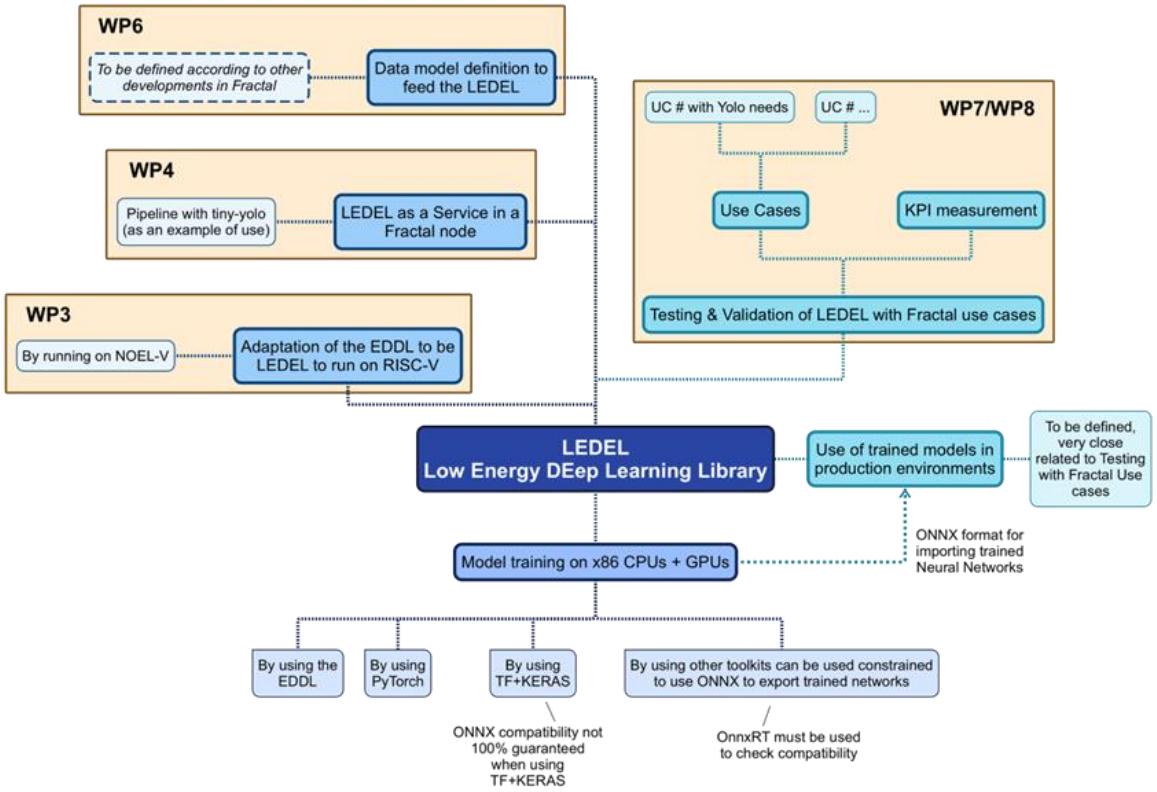

Figure 2 LEDEL development in FRACTAL

Thus, in order to check that the LEDEL could be ported to this hardware, we have used an emulated environment for the RISC-V architecture based on QEMU software. For this purpose, we have used an already created and compiled Linux Debian image named "Artifacts"

https://gitlab.com/api/v4/projects/giomasce%2Fdqib/jobs/artifacts/master/download?job=convert_riscv64-virt

from the project repository

https://gitlab.com/giomasce/dqib#debian-quick-image-baker-dqib

Once the image has been installed and running, EDDL has been compiled in this RISC-V virtualized environment. All the dependencies have worked completely fine. And a few simple tests have been executed checking their proper behavior.

One can train a model using the EDDL on a computer without limitation of resources and exporting it using the ONNX format. Afterwards, the model can be imported by the LEDEL in a FRACTAL node, and then used to infer from data received in the node.

As an example, it has been possible to train a simple model for the MNIST digit dataset, and then use it for inference. Obviously, as all the infrastructure is being emulated, this execution process has been quite slow. Also, it has been possible (i) to train this model

using an "outside" computer, (ii) to save it in ONNX format, (iii) to import it in the emulated machine and (iv) to infer.

The use of this pre-baked image of Linux running on RISC-V emulated platform has allowed us to check if the portability of the EDDL to this architecture was possible. Furthermore, it has given us the advantage of moving forward with T4.1 (LEDEL as a service in a FRACTAL node), and now we are able to test the deep learning model for the UC7.

Currently, this intermediate solution is being documented and packed using a docker. Next steps involve compiling and use the LEDEL in real hardware platform NOEL V.

3.2.3 Support for diverse redundancy

BSC's software-only diverse redundancy support builds upon a monitor process creating redundant instances of the application to be run with diverse redundancy (see Figure 1). In particular, the monitor process spawns the redundant execution of the application in two cores, one thread the head one, and the other the trail one. The monitor guarantees that the head thread is at least a given number of instructions ahead of the trail thread, where such number is platform dependent and must be large enough so that the trail thread cannot catch up with the head one between two consecutive checks of the monitor process. The monitor checks periodically the progress of the head and trail threads, and if, eventually, the trail thread is too few instructions behind the head process, the monitor stalls the trail process until the next monitoring check. When, eventually, the staggering (in terms of instructions) between the head and the trail is large enough, or if the head trail finishes its execution, the monitor allows the trail thread to resume execution. This guarantees that the state of the cores where redundant processes run differs at any time, and hence, a fault affecting both cores similarly will produce different errors that will be detected upon comparison of the outcomes of the head and trail threads.

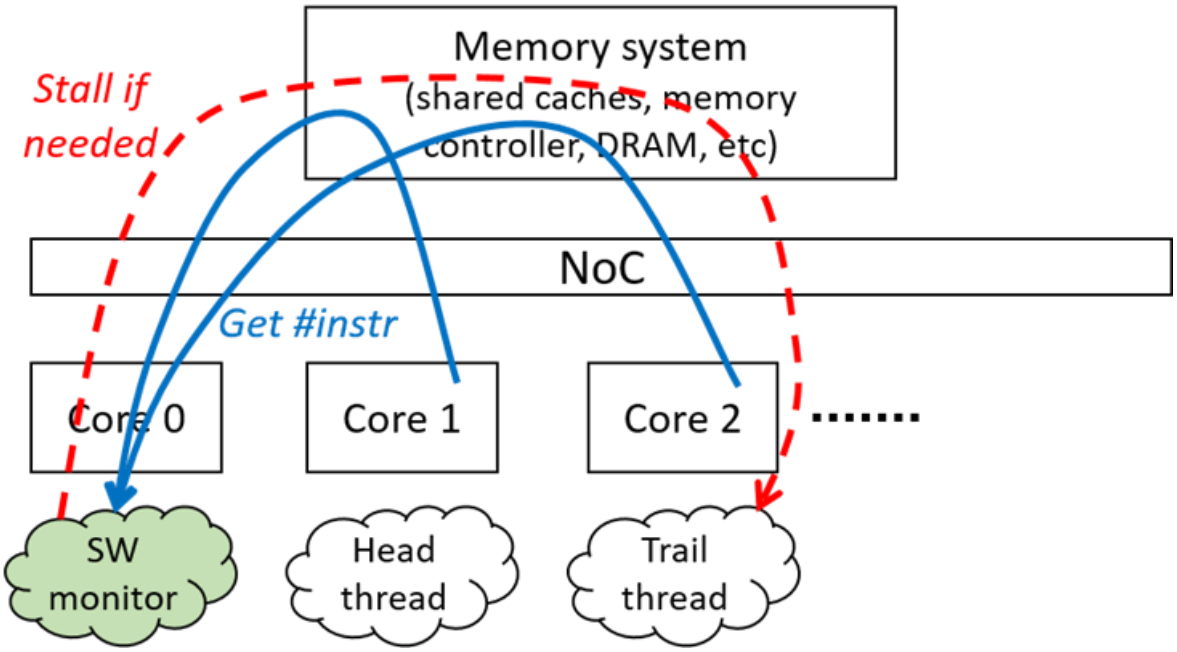

Figure 2: A schematic of the software-only support for diverse redundancy.

BSC software-only diverse redundancy support is deployed on top of the NOEL-V based platform and is intended to run as a Linux library. This type of service has been prototyped in the past for Arm-based platforms with Ubuntu Linux distributions [AKH+20]. Hence, the challenge in FRACTAL is threefold:

1. Porting this service from Arm to RISC-V using a different infrastructure (i.e., a FPGA board interfaced through a host instead of a directly accessible ASIC-based platform), and a different Linux distribution (*Buildroot* instead of Ubuntu).
2. Generating a standalone library easing the integration in use cases, rather than resorting to handcrafted prototyping in ad-hoc experiments as done in [AKH+20].
3. Validate the implementation against a number of relevant test cases prior to its integration in any of the FRACTAL use cases.

Those steps span across WP3 and WP4. In particular, work in WP3 relates to the porting of the basic functionalities on which to build the service, whereas work in WP4 is restricted to the use of those basic functionalities to deliver the service itself.

The first step, namely the porting of this feature from Arm to the particular target RISC-V platform consists of porting the following functionalities: (a) a call to spawn a new thread in a remote core, which will be invoked by the monitor process to create the head and trail threads; (b) a call to reset the instruction count of a remote core, where either the head or trail thread runs; (c) a call to retrieve the number of instructions executed in a remote core, i.e., the cores where the head and trail threads run; and (d) calls to stop and resume the execution of the trail thread, which runs in a remote core. Progress so far has led to the successful porting of those calls, which show to work properly. Building the service on top of those calls is part of WP4.

The second step is mostly within the scope of WP4 and, in the context of WP3, only requires validating that the calls in the first step can be properly encapsulated as part of a library, which we have already validated.

The third step, namely the validation of the overall service, falls within the scope of WP4. However, in the scope of WP3 we have the validation of the individual calls, as well as the tailoring of the staggering (in terms of instructions) between the head and trail threads to guarantee that the trail thread cannot catch up with the head thread. The latter, tailoring, has already been performed successfully. The former, namely the validation of the calls, has been successfully completed to some extent, but further tests are required. In any case, WP4 progress is possible with the current state of WP3 work.

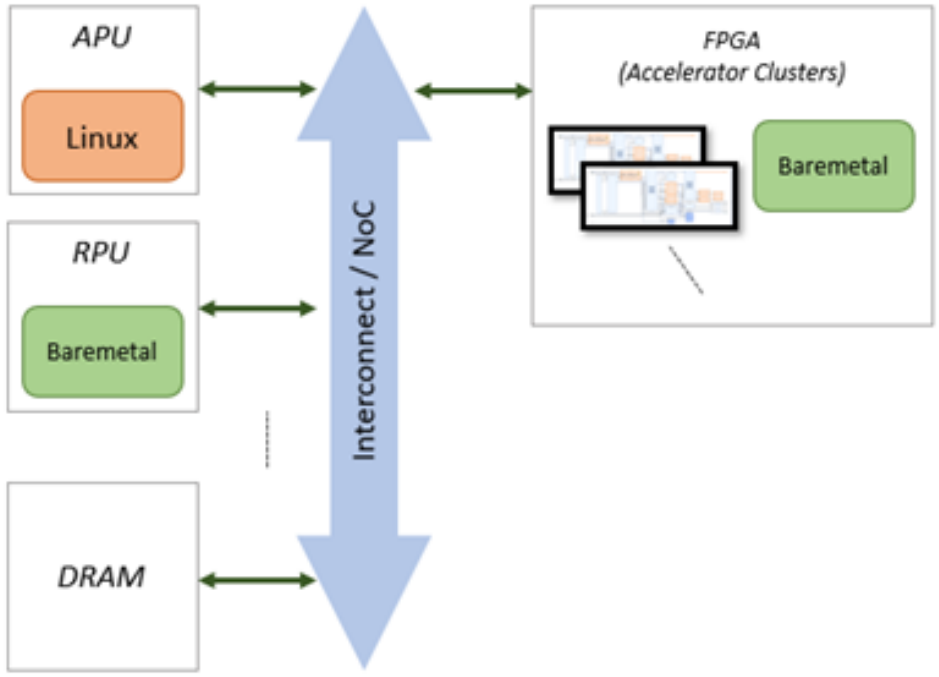

Figure 3: Software architecture for memory interference study

In order to simplify our analysis process on APU, we compiled a Linux kernel image based on *PetaLinux*. We decided to combine the *PetaLinux* system with a custom root file system based on the Ubuntu 20.04.2 distribution to take advantage of its rich ecosystem of software packages. To quantify the interference on host cores, we implemented two micro-benchmarks, which are capable of carrying out sequential and random memory traffic patterns towards the DRAM.

Both benchmarks are tuned to maximize the number of cache misses, to ensure the issued requests are in fact serviced from the DRAM (and not intercepted by the cache hierarchy). For the sequential access pattern, the memory reads are performed with stride equal to the L2 Cache Line Size. For the random-access pattern, the stride is randomic, but always a multiple of the cache line size. Typically, this pattern exhibits a higher average miss

latency, as the prefetching mechanisms in the DRAM itself (e.g., row buffers) are bypassed.

These two memory access patterns represent the worst case for realistic patterns that can occur in a real-life scenario. Our micro-benchmarks are modeled after the *lmbench* test suite http://lmbench.sourceforge.net/.

As for the RPU interference study, we used the same benchmarks used for APUs, but recompiled for bare metal. Finally, for the SmartDMA component, used in this case as a traffic generator, we implemented a simple standalone application, which allows the softcore to control the DMA.

3.3 WP3 and WP5

3.3.1 Supporting FRACTAL developments on cognitive awareness

There may be some software services for providing cognitive awareness to FRACTAL nodes. Those components may include libraries, drivers or software blocks to interface the hardware accelerators implemented in the nodes, which may be connected over different interfaces to the main processing unit (e.g., AXI/APB, shared memory).

With reference to the accelerator for age and gender recognition under development to be part of FRACTAL nodes, the application will be composed by the model and a Flask python server to provide REST API to external services and machines. All the software services will be packed inside a single docker image ready to acquire images and return predicted values. The only requirement to run the services will consist in the availability of the docker daemon in the operating system, together with the required hardware resources to load the model in main memory and to perform the inference.

3.4 WP3 and WP6

WP6 focuses on safe and secure orchestration of the distributed application domain -- how to manage the deployment and management of the distributed application and the data. Like WP5, also here the WP3 nodes need to offer required interfaces and resources while there are limitations on planforms.

3.4.1 Supporting FRACTAL framework consistency

WP6 will introduce methods for ensuring the framework consistency. How to ensure that all components of the framework are always consistent between each other. And how the framework raises and processes exceptions if any inconsistency happens.

3.4.2 Supporting FRACTAL application consistency

WP6 will introduce methods for ensuring the applications in the framework are consistent. While keeping the complex framework consistent, the evolving application brings another layer of potential problems in play.

4 Role of different platforms in FRACTAL Use Cases

The comprehensive discussions with all FRACTAL partners during the preparation of D2.1 "Platform specification (a)" showed a number of issues with our initial approach regarding how FRACTAL software nodes will be demonstrated as part of the use cases.

Work on Pulp SW nodes focuses on developing a software system for the low-end FRACTAL platforms. These will be limited in memory and storage but will benefit from price and energy consumption. Readers must understand that these FRACTAL low-end platforms enable a huge market segment of devices that cost a few euros and/or run years with a set of AAA-batteries.

As stated, the **Commercial Node** software components are already provided by Xilinx. Therefore, this document gathers the information related to those software components that may need some customization or integration effort to be adapted to FRACTAL node requirements.

It is a fact that the **Customizable Node** (PULP) will have much less resources available than the **Commercial Node**, e.g., onboard processor performance and volatile/non-volatile memory will be multiple decades smaller. Due to these facts, the high-end programming tools, such as Java/Python, are not necessarily available. However, the software node will offer POSIX standard APIs and C/C++ standard development tools for application development.

Despite the limitations above, the **Customizable Node**, if carefully designed, will meet application specific performance easily Figure 1. presents the three tiers of the system architecture. If a node exists in mist tier, it is a good candidate to be a Pulp-based low-end node.

Another limitation on **Customizable Nodes** is caused by the high level of optimization of the node's hardware. Having a complete node software framework for all the platforms is out of scope of this project. Some of the use case features may need to be demonstrated at multiple (different) HW platforms.

As the use case is a concrete demonstration for the use case provider, it should not be surprising that the main goal of the use case provider is to make sure that the use case can run without issues within the FRACTAL project. As a result, some project partners expressed rather extensive requirements for their own use cases in order not to be limited by the hardware capabilities in the future, and some others expressed interest in using systems that they are more familiar with. In practice, this has led to several use case providers stating the need for a symmetric multi-core system running a standard Linux distribution.

In all cases, these requirements are perfectly understandable and most of them could be implemented using the commercial node of FRACTAL, the Xilinx VERSAL platform. As outlined in chapter 4.2, the basic customizable node has been targeted towards simpler IoT applications and lacks the power to fulfil several of these requirements. At first sight

this creates an apparent imbalance of utilization between the commercial and the customizable node.

FRACTAL partners have discussed various approaches to provide a solution and have decided on a number of measures to make sure that the ideas developed as part of FRACTAL are validated on common platforms that are available to all project partners. From those discussions come the following recommendations.

- There are several use cases (see Section 5) that are content to use the FRACTAL hardware nodes as provided.
- FRACTAL has identified three tiers of FRACTAL hardware nodes: low (Mist), medium (Edge), high (Cloud). Node versions that all share similar interfaces and interact with each other Figure 1. shows an illustration of such an organization where simpler nodes are acquiring data and delegating more complex tasks to nodes with higher complexity. The figure is meant as an example, and different allocations of tasks are currently under discussion within WP5/6. In this model, the industrial node covers the higher-end version, while the customizable node is seen as the lower-end version. As described chapter 4.3, partners have suggested several alternatives for the medium-end nodes.
- Some partners are relying on their prior work and experience to implement some of their contributions. Most of these are based on hardware systems that are similar and/or compatible with FRACTAL nodes but have some differences. These include implementations in earlier models of Xilinx MPSoC platforms than the VERSAL as well as other openly available RISC-V systems. Out of practical considerations, FRACTAL partners have added these as additional platforms to the initially identified hardware nodes.

It was also recognized that official FRACTAL nodes could be instrumental for research aspects involving developments in WP4/5/6 and the experience from these explorative works could then be used to evaluate the potential of these developments in use cases that consider more traditional solutions. As a concrete example, novel safety solutions with hardware support could be explored on a small scale in the customizable node as part of WP4. The results of this exploration could then be used to directly estimate the gains achievable by this approach in a use case that employed an alternative hardware node. The work done throughout the first part of the project allowed partners to realize different possibilities and showed that the key point was that all developments from FRACTAL technical work packages should be accessible for all FRACTAL partners. While the initially identified Hardware Nodes cover a large range of the specification spectrum, partners could also make use of additional hardware nodes as long as this work could be used/verified/evaluated by all partners.

4.1 Customizable node (RISC-V based PULP)

The customizable node in FRACTAL is a RISC-V based PULP platform (particularly the PULPissimo microcontroller) which is further described in D3.3. For the sake of completeness, Figure 5 depicts PULPissimo as part of the FRACTAL big picture.

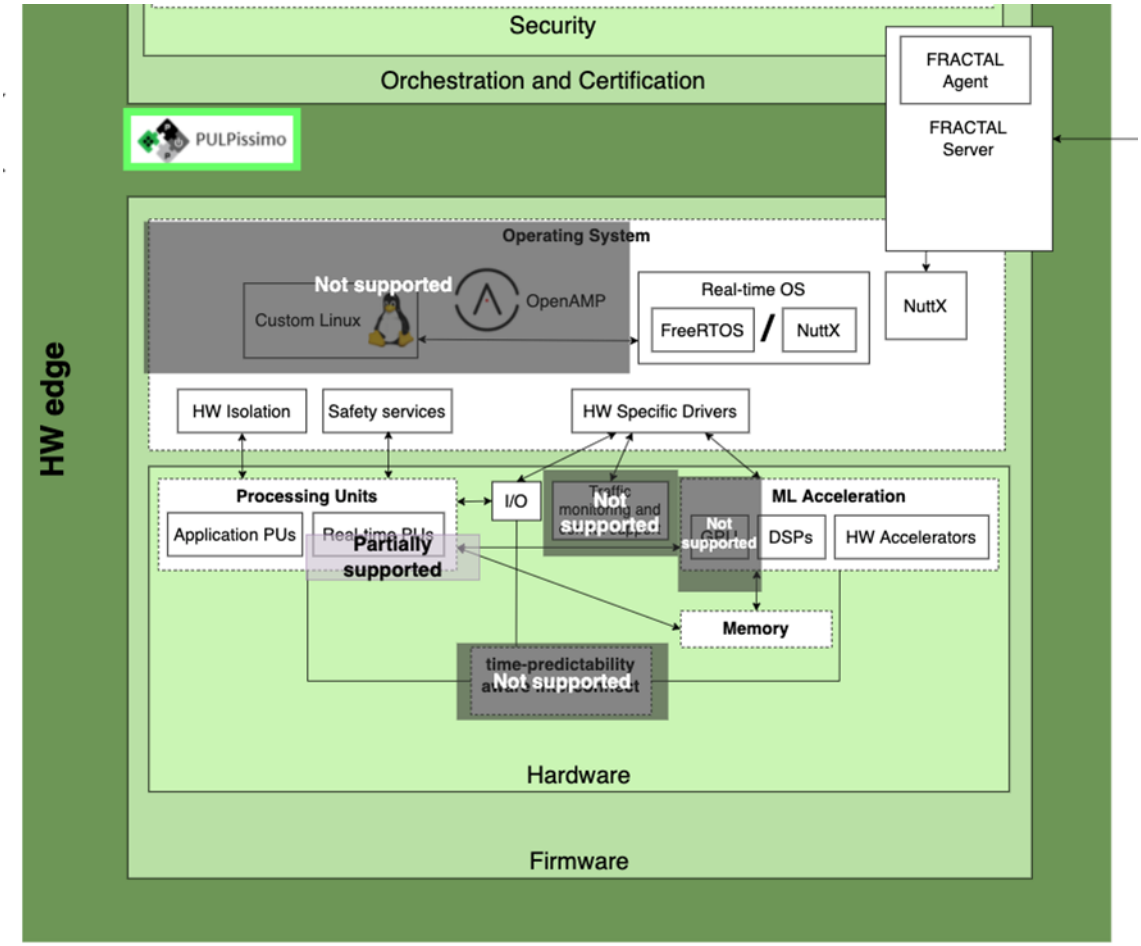

Figure 4: PULPissimo as part of the FRACTAL big picture

PULPissimo is a RISC-V based system, with resources adopted to IoT workloads – RAM (volatile) up to few megabytes and flash (non-volatile) some hundreds of megabytes. System clock speeds are typically below 1GHhz and often the processor may lack the memory protection modules.

RISC-V is based on proven RISC principles from the 80's. Recently it has had additional benefits, as an open ISA. This enables developments of both open and commercial applications. This is interesting both for Industry and Academia as it lowers the barriers to share developments on the ISA between various partners and benefits the cumulative research results. The RISC-V licensing terms make it possible to develop open-source hardware

As a platform optimized for the IoT applications, the PULP platform has a much lower power budget in the milliwatts or even microwatts range. This is especially interesting for applications where battery powered nodes are expected to operate for years.

PULPissimo systems have FPGA based implementations, where all digital parts (including the processor) exist as FPGA code, allowing the peripheral and HW-accelerators optimized purely to the application. Potentially parts of the application logic may be implemented in hardware and even altered in runtime. An interesting option is the implementation of AI engine primitives by HW.

By converting the FPGA design to the ASIC, the cost of PULP based systems (in high volumes) can go down to the cents.

For software point of view, the advanced operating systems such like Linux are out of scope due to the limitations of the underlying platform. On the other hand, plain bare-metal applications will be too complex to be managed by the Fractal framework. There exists numerous RTOS's, but for FRACTAL Pulp nodes the open source Nuttx RTOS has been chosen. Main benefit of Nuttx is the POSIX compatibility. This RTOS offers primitives such as threads, devices and sockets. Due this the libraries and the applications may develop to be fully Linux compatible – some existing Linux libraries and application can be just compiled into the Nuttx. Physical communication devices can be integrated into the sockets. By utilizing the IP-stack, the local and wide area networks can be seamlessly hidden below IP-networking. This again eases the application development and isolates the network configuration. Existing authentication and encryption methods can be utilized. As result, the application development does not require any RTOS specific code.

For the development the standard GNU C/C++ development environment is available for the application developer. Also, tools such as GDB/JTAG offer test/debug features on platform.

Due to HW limitations – mainly RAM/flash -- tools such as java and python are typically not available (some limited versions do exist).

Due the limited memory resources -- the containers and hypervisors does not make sense on Nuttx. By increasing the processor complexity and sizes of memories, these problems can be solved, but when moving in that direction, the better solution is to switch to the Linux.

4.1.1 Pulp onboard resources

4.1.1.1 Key management

Key security element of safe software deployment and authentication is secure key management. To do this in a safe way an additional hardware module is required.

4.1.1.2 Hardware acceleration

There are two typical approaches for HW acceleration in Pulp platform.

RISC-V offers a mechanism to introduce new instructions to the processor. Utilizing this mechanism makes the acceleration totally seamless to the application software. Drawback is the requirement of a customized compiler. Less general the acceleration is, less practical this approach is.

Another typical approach is to build a custom peripheral that offers a memory mapped register interface to the software.

One interesting case is the HW acceleration of AI-primitives.

If the Pulp is running on FPGA platform, software may upload different functions to FPGA, thus it can dynamically change the behavior of the HW acceleration. By utilizing this mechanism, the parts of application software can be executed on HW – however this is outside scope of this document.

4.1.2 Safety considerations on Nuttx Pulp – WP4

Refencing following publication:

RISC-V for Real-time MCUs - Software Optimization and Microarchitectural Gap Analysis

https://ieeexplore.ieee.org/document/9474114

Abstract:
Processors using the RISC-VISA are finding increasing real use in IoT and embedded systems in the MCU segment. However, many real-life use cases in this segment have realtime constraints. In this paper we analyze the current state of real-time support for RISC-V with respect to the ISA, available hardware and software stack focusing on the RV32IMC subset of the ISA. As a reference point, we use the CV32E40P, an open-source industrially supported RV32IMFC core and FreeRTOS, a popular open-source real-time operating system, to do a baseline characterization. We perform a series of software optimizations on the vanilla RISC-V FreeRTOS port where we also explore and make use of ISA and micro-architectural features, improving the context switch time by 25% and the interrupt latency by 33% in the average and 20% in the worst-case run on a CV32E40P when evaluated on a power control unit firmware and synthetic benchmarks. This improved version serves then in a comparison against the ARM Cortex-M series, which in turn allows us to highlight gaps and challenges to be tackled in the RISC-VISA as well as in the hardware/software ecosystem to achieve competitive maturity.

4.1.3 AI processing on Nuttx Pulp -- WP5

On this project the AI scenarios on Pulp node are based preloading models to the Node. The learning of model is done elsewhere (i.e Fractal cloud) and machine learned model is uploaded to node(s). This is demonstrated by UC3.

4.1.3.1 Software approach

In the case of low-end nodes, the lack of high-level tools such as python and Java and the limited RAM memory make the plain SW approach challenging. Infesting cases would be a hybrid solution where software loads pre trained modules to be executed in HW.

4.1.3.2 Hardware approach

A very interesting case would be the HW acceleration of AI-primitives. This would result in a case where models can be loaded to software, but the actual processing happens in HW.

Refencing following (Fractal) publication:

Ternarized TCN for µJ/Inference GestureRecognition from DVS Event Frames:

https://ieeexplore.ieee.org/document/9406333
Abstract:

Heavily quantized fixed-point arithmetic is becoming a common approach to deploy Convolutional Neural Networks (CNNs) on limited-memory low-power IoT end-nodes. However, this trend is narrowed by the lack of support for low-bitwidth in the arithmetic units of state-of-the-art embedded Microcontrollers (MCUs). This work proposes a multi-precision arithmetic unit fully integrated into a RISC-V processor at the micro-architectural and ISA level to boost the efficiency of heavily Quantized Neural Network (QNN) inference on microcontroller-class cores. By extending the ISA with nibble (4-bit) and crumb (2-bit) SIMD instructions, we show near-linear speedup with respect to higher precision integer computation on the key kernels for QNN computation. Also, we propose a custom execution paradigm for SIMD sum-of-dot-product operations, which consists of fusing a dot product with a load operation, with an up to 1.64 × peak MAC/cycle improvement compared to a standard execution scenario. To further push the efficiency, we integrate the RISC-V extended core in a parallel cluster of 8 processors, with near-linear improvement with respect to a single core architecture. To evaluate the proposed extensions, we fully implement the cluster of processors in GF22FDX technology. QNN convolution kernels on a parallel cluster implementing the proposed extension run 6 × and 8 × faster when considering 4- and 2-bit data operands, respectively, compared to a baseline processing cluster only supporting 8-bit SIMD instructions. With a peak of 2.22 TOPs/s/W, the proposed solution achieves efficiency levels comparable with dedicated DNN inference accelerators and up to three orders of magnitude better than state-of-the-art ARM Cortex-M based microcontroller systems such as the low-end STM32L4 MCU and the high-end STM32H7 MCU.

For additional information see:

https://pulp-platform.org/pulp_sw.html
https://pulp-platform.github.io/pulp-dsp/tutorial-index/

4.1.4 Application orchestration on Nuttx Pulp – WP6

Fractal application deployment is based on containers. While Nuttx lack resources to run containers as such, the problem is solved by having special containers on cloud (or high-end edge nodes), that connects to specific nodes and upload specific binaries to Pulp nodes. While most of this work is done in scope of WP6 here is just described the services that node offers.

This further studied on D6.2.

4.2 Versal node

The Xilinx VERSAL ACAP is expected to be deployed as part of the VCK190 Evaluation Kit board, which provides support for several I/O interfaces and memory devices. For completeness, Figure 5Figure 5 shows the VERSAL platform in the FRACTAL big picture.

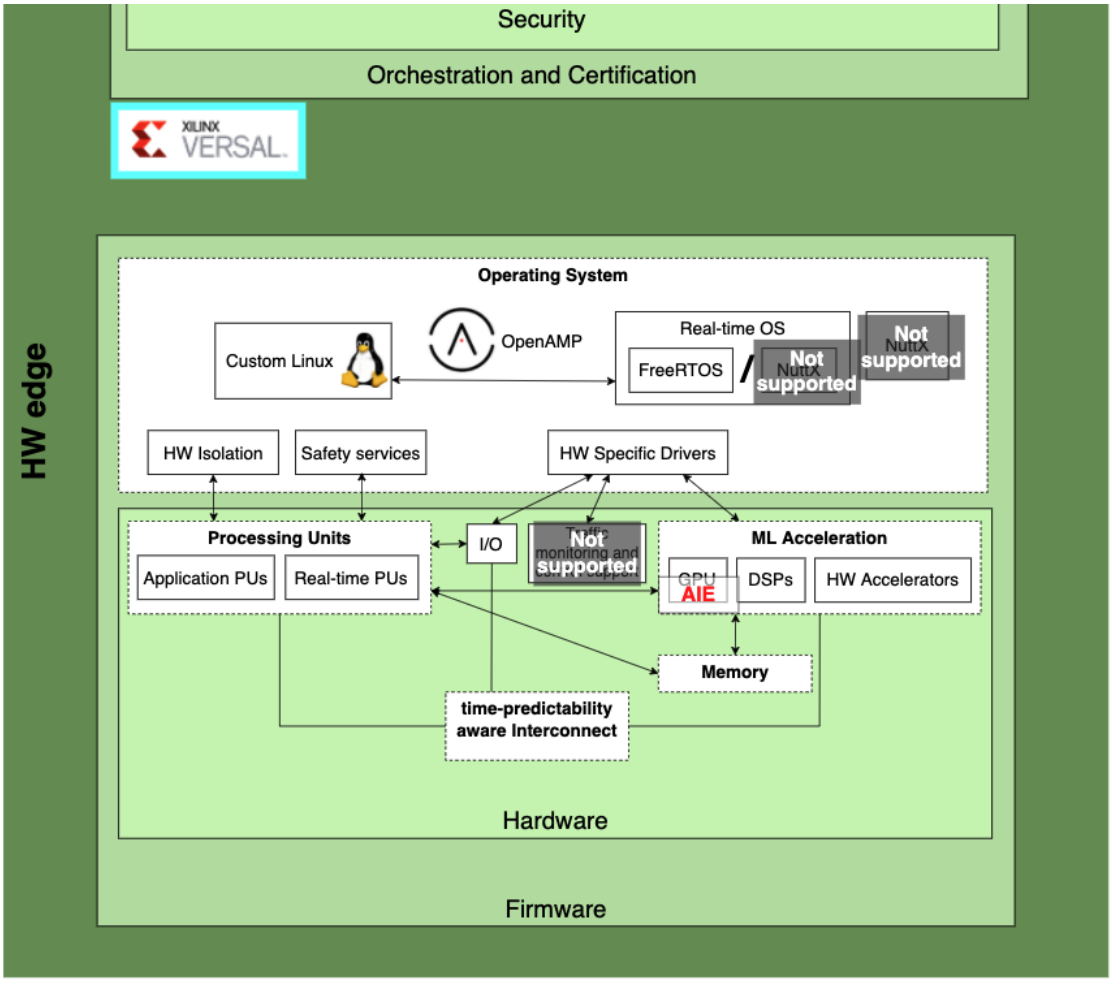

Figure 5: The VERSAL platform in the FRACTAL big picture

The VERSAL architecture (Figure 6Figure 6) combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC).

Like the earlier Xilinx Zynq MPSoC products, VERSAL ACAP devices still offer the two main components:

- **Processing System** (PS) consists of a dual high-performance ARM Cortex A72 cores that can run Linux or other operating systems. This system is augmented by a dual-core ASIL-C certified real-time processing subsystem based on Arm Cortex R5F cores. Together these systems address the needs of most modern computing needs using a traditional programming interface.

- **Programmable Logic** (PL) allows this system to be augmented by hardware accelerators customized to a particular compute function.

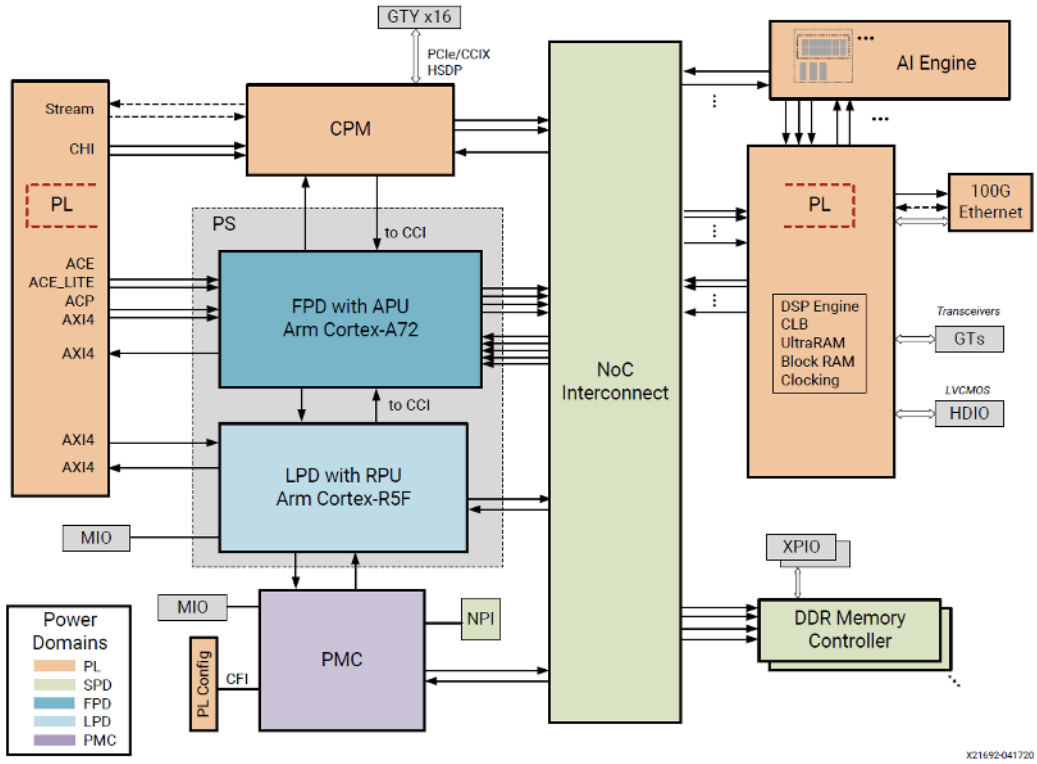

Figure 6. Top level schematic of Xilinx VERSAL ACAP

On one hand, VERSAL designs are enabled by the Vitis™ tools, libraries, and IP. The Vitis IDE lets the developer program, run, and debug the different elements of VERSAL AI Engine application, which can include AI Engine kernels and graphs, PL, high-level synthesis (HLS) IP, RTL IP, and PS applications. Vitis offers two development approaches:

- **Accelerated Flow.** It allows to build a software application using the OpenCL or the open-source Xilinx Runtime (XRT) native API to run the hardware kernels on accelerator cards, or on a Linux-embedded processor platform. The Vitis tool includes the v++ compiler for the hardware kernel on all platforms, the g++ compiler for compiling the application to run on an x86 host, and Arm® compiler for cross compiling the application to run on the embedded processor of a Xilinx device.
- **Embedded Flow.** It provides a complete environment for creating software applications targeted for the embedded processors. It includes a GNU-based compiler toolchain, C/C++ development toolkit (CDT), JTAG debugger, flash programmer, middleware libraries, bare-metal BSPs, and drivers for all the Xilinx IPs. It also includes a robust IDE for C/C++ bare metal and Linux application development and debugging.

On the other hand, the Peta Linux tools (built on top of the Yocto Project) offer everything necessary to customize, build, and deploy embedded Linux solutions on Xilinx processing

systems. Tailored to accelerate design productivity for SoC devices, the solution works with the Xilinx hardware design tools to facilitate the development of open-source Linux systems for VERSAL devices.

FRACTAL nodes will use the tools provided by Xilinx to build the system as stated in the different use cases. This deployment will include OS or system software customization (e.g., hypervisor, bootloader, kernel) to match the requirements of FRACTAL nodes, and file system creation, including all the software packages and configuration stated in the platform description requirements (e.g., Python, JAVA, net-tools, others). It will also consider the development of use case specific high-level applications or low-level drivers required by custom building-blocks (e.g., security related modules).

It is important to mention that even though the developments will be very use case specific, they will share a common development stack, based on the options and considerations stablished by Xilinx (

Figure 7).

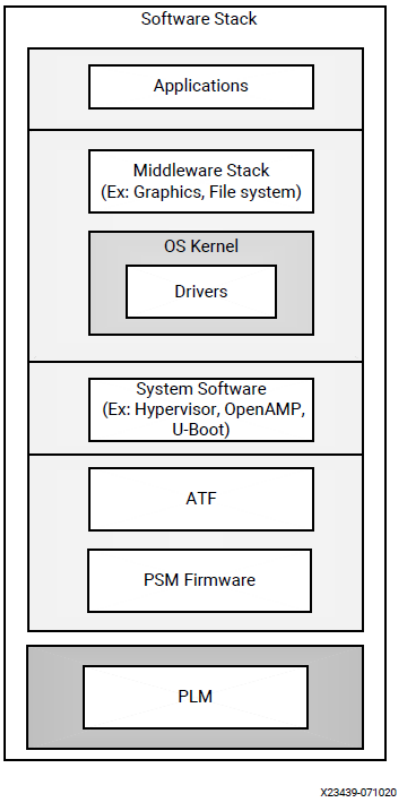

Figure 7: VERSAL Linux development stack

As VERSAL provides many different implementation possibilities, in the beginning of the project the main approach was not only to analyze, understand and determine the requirements coming from every use case, but also the proposed roadmap in order to achieve use case objectives. Reference software architecture of a cognitive edge computing node with FRACTAL properties will be defined and a common repository of

generic qualified components will be set up. Particular attention will be paid on providing flexible computing nodes, that are reusable by others and that efficiently support the software on providing acceleration for the learning part.

The reference software architecture design for the VERSAL based FRACTAL node will be described in the following deliverable (D3.6), so that it reflects the customization and integration made on top of the FRACTAL nodes based on the software components provided by Xilinx.

4.2.1 Versal onboard resources

Native Linux applications that include the Xilinx runtimes for different hardware targets can still be built as native applications and run against the support of the runtime of the Versal Linux host. This may be beneficial in nodes where the FRACTAL orchestration and system interface level is only used to provide data or model and otherwise has low interaction with the application itself. Such applications can make use of all features on the board as the device provides the complete underlying hardware description.

The FRACTAL Edge Software stack uses microservices to build up an application. As applications in the node need to be exchanged or scheduled from the system level context aware scheduling, it is desirable to fetch these from the image store and thus run these applications in a containerized version.

To achieve this, a docker container is devised that is based on an Ubuntu Linux and holds the full addition of libraries to satisfy the respective runtime, as an example XRT or VART. In such a docker environment the application can run and reach out to the devices through the kernel. The scaling of this approach will be further investigated, but it is required to control the scope and reach of the application.

4.2.2 Safety considerations on Versal – WP4

The Versal reference platform is being set up in two base designs. While one of these is not imposing restrictions on actual computation cores, the second version is intended to build the certifiable platform along D2.3.

This safety focused setup is guided by the separation capabilities of the Versal ACAP architecture. As already pointed out in Chapter 3 Versal devices offer a hardware abstraction of core-related power functionality and allow access to these through the PMC. While the PMC exposed features can be used rather freely, this is not reasonable for a proper safety approach.

To satisfy these safety concerns, the separation of the elements is considered from ground up and the services on the FRACTAL node level need to settle to a particular central provider that actually then as a proxy call into the PMC. In the safety focused platform all control requests for power and scaling the actual low-level accesses are collected on one of the RPUs. This RPU will carry out all PMC related transactions on behalf of the node. This platform setup also incorporates a secure boot mechanism to ensure the guaranteed boot into the safety environment. The exposure of the service interface to the RPU is carried out within the respective WP4 tasks.

The support for the time triggered scheduling and node-level and system-wide features as from WP4, are attributed to the RPU in the designs where these real-time capabilities are required. In this setup the Adaptive Time-Triggered Network Interface is controlled by this core to provide the scheduling services on top. This RPU can unify the node control transactions with this service or push transactions to the respective other RPU if it is available in the design.

The communication between the heterogenous cores, from the Fractal Edge Software Linux to the RPU is carried out using OpenAMP. To cater for such a setup, the RPU projects are deploying FreeRTOS.

4.2.3 AI processing on Versal -- WP5

The Versal based platform, specifically with the VCK190 and the VCKC1902 device offers specific ML acceleration means by deploying the AI Engines. These are supported by the DPU IP hardware structure in the reference platform design.

In the Versal platform BSP for Linux the supporting device tree elements for all particular features of the device are made available and the respective Vitis AI runtime (VART) libraries are installed. The effective pre-trained model deployment is carried out through a proprietary Xilinx toolchain, Vitis AI, that is capable of processing ML models from a variety of toolchains, like Tensorflow and PyTorch, but also reading from ONNX models.

The analysis and translation of such models yields a combination of hardware configuration information and embedded code artifacts for multiple computation engines. The translated models themselves are exchangeable and may be retrieved from the model repository on demand. The translation itself can also be carried out as a service in the FRACTAL cloud.

A translated model is deployed through the VART runtime in a Linux application. Such an application is typically triggered through the orchestration level of the Fractal Edge software and can be added to a specific application container.

4.2.4 Application orchestration on Versal – WP6

The FRACTAL reference design based on Versal ACAP is set up to include the basic application support along the WP6 proposal. Current basic Petalinux setup has been operating Docker and Kubernetes (MicroK8s) and Mosquitto. The full set is planned to use Prometheus and Juju. All additional orchestration features are deployed in respective containers.

4.3 Other nodes

On Fractal the Versal and the Pulp platforms were selected to use as node HW-platforms. However, to demonstrate some specific developments partners have chosen to use some other platforms.

During this study performed in the first part of the project, several use cases stated the need for more traditional RISC-V based systems (capable of running single-core or SMP Linux) which resulted in some additional hardware nodes being added. The software capabilities of these additional RISC-V nodes will be covered in this section.

4.3.1 NOEL-V

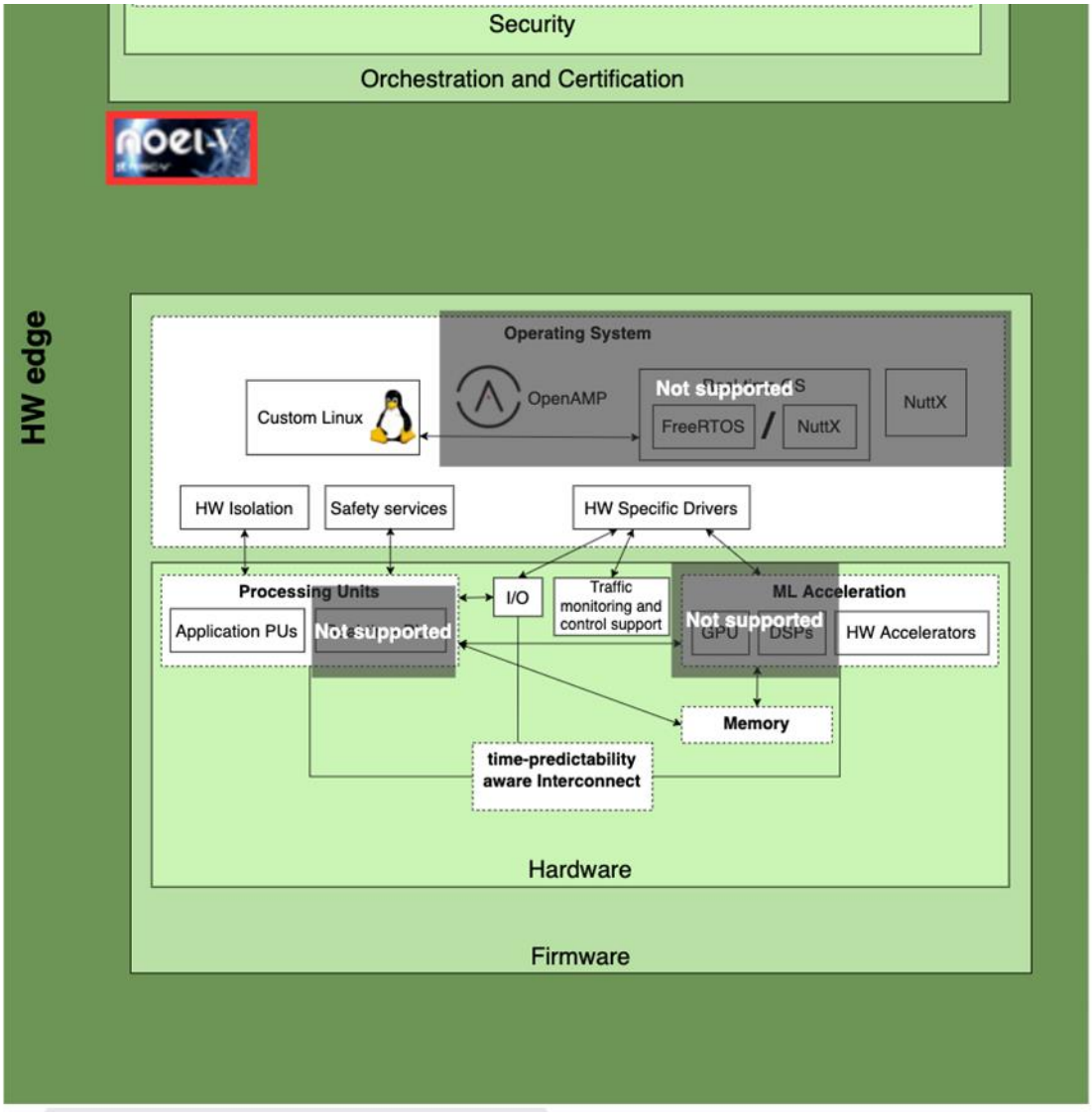

Figure 9 NOEL-V as part of the FRACTAL big picture

The NOEL-V based SoC builds upon the GPL platform provided by the H2020 SELENE project (https://www.selene-project.eu/). The SELENE SoC has been synthesized in a Xilinx Virtex UltraScale VCU118 FPGA and the original NOEL-V SoC is also available for the KCU115, although it can be ported to other boards. For completeness, Figure 9 shows the NOEL-V based SoC as part of the FRACTAL big picture.

The NOEL-V SoC supports memory management units, and implements Translation Lookaside Buffers (TLBs), both for data and instructions, locally in each core. The SoC also provides support for cache coherence. Those features allow booting SMP Linux and RTEMS operating systems among others and allow sharing data across cores.

The Linux image, on which current developments are performed, has been built with buildroot (2021.02LTS), and the required sources are provided by Cobham Gaisler at https://www.gaisler.com/index.php/downloads/sw-noelv-downloads.

The platform implements the RV64I RISC-V ISA along with the G, C and H extensions.

Standard software tools for compiling, debugging, and the like are supported since the platform adheres to the RISC-V standard.

4.3.2 ARIANE/CVA6

Ariane/CVA6 supports Linux, both 32-bit on CV32A6 and 64-bit on CV64A6. In a first step of the FRACTAL project, Linux has been ported to CV32A6 with recent versions of the various components (BBL, Buildroot 2021.5.rc1, Linux kernel 5.10.7).

CV64A6 has supported 64-bit Linux for a longer time and work has been performed to update to the same versions as CV32A6. Compilation is supported by GCC 9.3, and software simulation is supported by Spike. As CVA6 is aligned with RISC-V standard extensions, we can expect software support by other generic tools, such as Clang/LLVM, without further port.

Since D3.2 release, the support of popular components has been added: UBoot as an alternative to BBL (Berkely Boot Loader), as well as the OpenSBI firmware. The support of Yocto embedded Linux image builder, as an alternative to Buildroot, is being worked upon.

On top of the Linux operating system, the Ariane/CVA6-based node will support the FRACTAL software components depicted in the figure below to deliver the services needed in UC4. On the AI side, only inference will be supported.

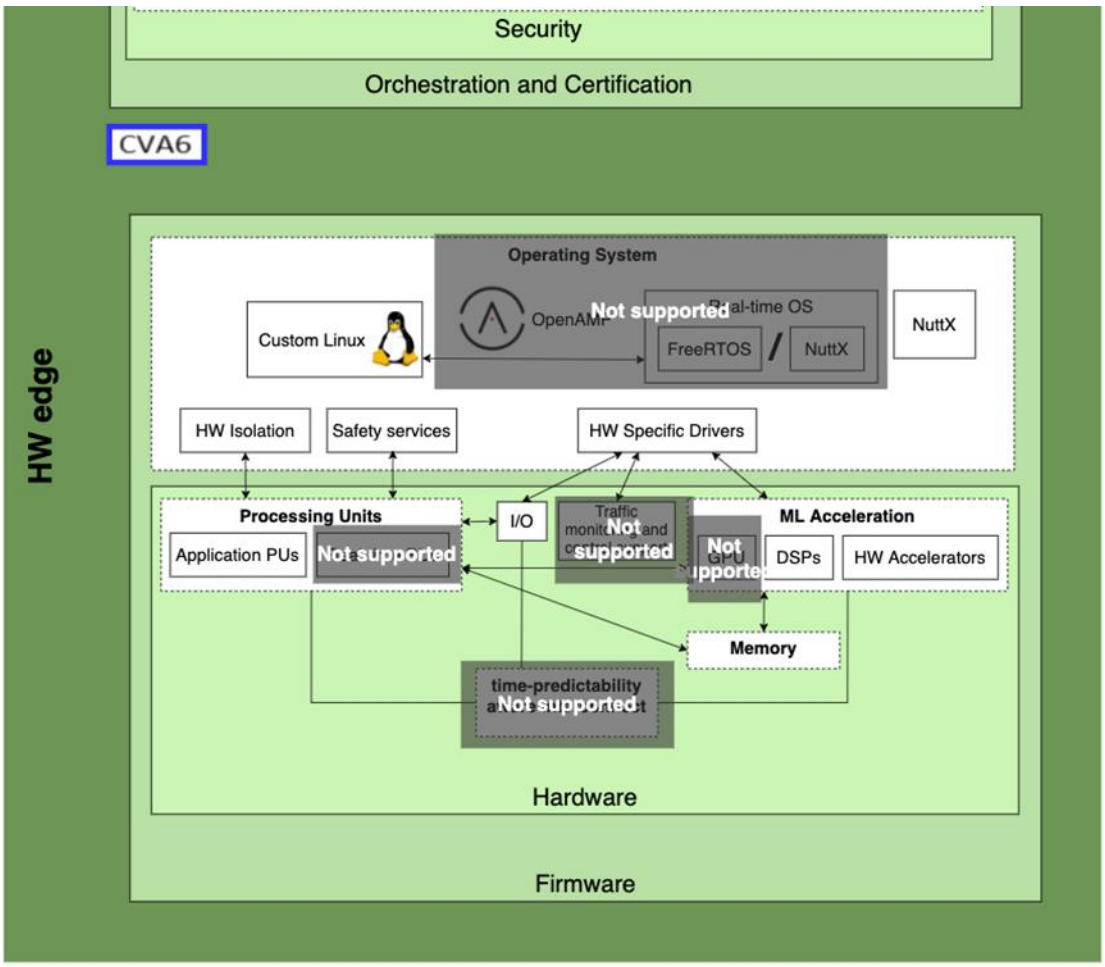

Figure 10 CVA6 as part of the FRACTAL big picture

4.3.3 Yet additional platforms

Some special (technical) cases may be implemented and/or demonstrated by yet other platforms (i.e. Raspberry PI). Those cases will be presented here.

4.3.3.1 Asymmetric multiprocessing AMP

Yet another interesting option is to utilize both Linux and *Nuttx*. Cost efficient way to do this is to use Asymmetric Multi Processing (AMP) in a multicore processor. One for the cores is running Nuttx, while others run Linux. Main benefits are real-time processing. As Nuttx is RTOS, it can process the real-time deadlines better than Linux. Another benefit is the low-energy operation. Linux can put to deep sleep, while Nuttx handles basic operations and when required wake up the Linux. This is not possible to demonstrate on a PULPissimo platform.

5 Interaction of UCs with FRACTAL nodes

The FRACTAL cloud SW development has begun in top-down approach. The main AI-applications are under development and aim to demonstrate with simulated nodes.

The FRACTAL Pulp node SW development progresses in by bottom-up approach, where the basic features are ramped up. This includes the board OS-wake-up, simple LED-Blink demonstrations and setting up SW-repositories and tools. While development progresses the service layers AI, Connectivity, Security will be integrated. The final phase of integration is the application – use case integration, where node SW and cloud SW are integrated as complete solution.

On FRACTAL Versal the use case development can begin after the FRACTAL adaptation interfaces are agreed.

The detailed interaction between FRACTAL SW nodes and use cases will be described in subsequent versions of this deliverable (D3.4, D3.6). In document **D3.1_Preliminary_HW_node** (chap 5) presents the resource-based allocation of use cases on FRACTAL nodes.

6 Conclusions

The two main hardware nodes (commercial and customizable) are being made available to all FRACTAL partners. Following discussions regarding the design requirements, at least in the first phase of the project, it was seen that partners would benefit from additional hardware nodes that fall in between the two default options. WP3 partners are discussing providing such solutions in agreement with other partners from technical WPs 4/5/6 as well as the UCs.

7 Next steps

In the first year of the project, the goal was to create an alignment between partners and allow the node supporting partners to understand the requirements as well as the use case and technical partners to see both the capabilities and limitations of the available systems. As the work in technical WP's intensifies, the interaction between the partners will increase due to the design, integration and implementation of the results of the WPs in the HW nodes presented in this deliverable.

8 Risks and Mitigation plans

The **D3.3_Intermediate_HW_node** (chapter 6.2) presents the current set of risks.

9 Bibliography

[AKH+20] S. Alcaide, L. Kosmidis, C. Hernandez and J. Abella, "Software-only based Diverse Redundancy for ASIL-D Automotive Applications on Embedded HPC Platforms," 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2020, pp. 1-4, doi: 10.1109/DFT50435.2020.9250750.

[AAA+21] J. Abella et al., "Security, Reliability and Test Aspects of the RISC-V Ecosystem," 2021 IEEE European Test Symposium (ETS), 2021, pp. 1-10, doi: 10.1109/ETS50041.2021.9465449.

10 List of Abbreviations

ACAP Adaptive Compute Acceleration Platform (relates to VERSAL)
AI Artificial intelligence
APB Advanced Peripheral Bus
API Application Programming Interface
APU Application Processing Unit
ASIC Application-specific integrated circuit
AXI Advanced eXtensible Interface
BBL Berkeley Boot Loader
BSP Board Support Package
CDT C/C++ Development Toolkit
CLang C Language
CPU Central processing unit
DMA Direct Memory Acces
DoA Description of Action
DRAM Dynamic/Distributed random-access memory
EDDL European Distributed Deep Learning Library
FPGA Field-Programmable Gate Array
GCC GNU Compiler Collection
GDB GNU Debugger
GPL General Public License
GNU GNU is Not Unix
HLS High-Level Synthesis
HW Hardware
IDE Integrated Development Environment
IoT Internet of Things
IP Intellectual Property
ISA Instruction Set Architecture
JTAG Joint Test Action Group
LED Light-Emitting Diode
LEDEL Low Energy DEep Learning Library
LLVM Former initialism of Low Level Virtual Machine. Concept currently expanded.
MLOps Compound of "machine learning" and the continuous development practice
MPSoCMultiprocessor System-on-Chip
NoC Network-on-Chip
ONNX Open Neural Network eXchange
OpenCL Open Computing Language
OS Operating System
PL Programmable Logic
PMC Platform Management Controller
POSIX Portable Operating System Interface
PS Programmable System

PULP Parallel Ultra Low Power
QEMU Quick EMUlator
REST Representational state transfer
RISC-V Reduced Instruction Set Computer
RPU Real time Processing Unit
RTEMS Real-Time Executive for Multiprocessor Systems
RTOS Real Time Operating System
SMP Symmetric Multi-Processing
SoC System on a Chip
SW Software
TLB Lookaside Buffer
UC Use Case
WP Work Package
XRT Xilinx Runtime