Deliverable

D3.2 Preliminary Fractal Software node and services

Deliverable Id:	**D3.2**
Deliverable Name:	**Preliminary Fractal Software node and services**
Status:	**Draft**
Dissemination Level:	**Public**
Due date of deliverable:	**31. August 2021**
Actual submission date:	**2021**
Work Package:	**WP3**
Organization name of lead contractor for this deliverable:	**Offcode Oy**
Author(s):	**Antti Takaluoma**
Partner(s) contributing:	**Jérôme Quévremont, Thales** **Kévin Eyssartier, Thales** **Jaume Abella, BSC** **Edurne Palacio, Ikerlan** **Lauri Loven, UOULU** **Leticia Pascual, Solver (SML)**

Abstract: This deliverable (D3.2) is the first of a series of deliverables that describe the software work for the FRACTAL project hardware nodes. This is the first of three deliverables on software node, and it will be updated throughout the project with D3.4 (M18), and D3.6 (M20).

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
0.1	26.7.2021	Composed, from project sources.	Antti Takaluoma (OFFC)
0.2	9.8.2021	NOEL-V platform introduced, BSC safety-related features included	Jaume Abella (BSC)
0.3	17.08.2021	VERSAL node information added + some formatting issues fixed	Edurne Palacio (IKER)
0.4	19.08.2021	AI	Lauri Loven (OULU)
0.5	20.08.2021	Refactored and updated	Antti Takaluoma (OFFC)
1.0	1.9.2021	Updated based on Reviewer comments	Antti Takaluoma (0FFC)

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 (D3.2) is the first of a series of deliverables that describe the software work for the FRACTAL project hardware nodes. This is the first of three deliverables on software node, and it will be updated throughout the project with 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 a large number of partners (28) both from the 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.

In the original plan two main options for the hardware node were foreseen:

- **Commercial node** based around the Xilinx VERSAL ACAP (Adaptable Compute Acceleration Platform)
- **Customizable node** based around the open-source RISC-V based PULP platform

These two main nodes continue to form the backbone of the developments and implementations for the FRACTAL project, but to accommodate practical needs and requirements of project partners additional (related) platforms were also leveraged when necessary.

This document focuses on the software developments to be done for **customizable nodes** (PULP and additional RISC-V related platforms), but it also gathers the software level customizations and integrations required to adapt the **commercial node** software components provided by Xilinx to the FRACTAL nodes.

The organization of the deliverable is as follows. Section 2 provides a general introduction to the software and hardware components placed on top of FRACTAL hardware. Following the discussions around D2.1 "Platform Specification (a)" the section also clarifies the role of the hardware platforms in FRACTAL for the use cases. Section 3 summarizes the current state of the software in each of the mentioned hardware platforms, and Section 4 describes how the technical developments in work packages WP4/5/6 make use of the software components being described in this deliverable. Section 5 describes how use cases will utilze FRACTAL nodes. Section 6 provides conclusions for the first year of FRACTAL activities on Software Node development.

2 Introduction

Note to the reader: This document has common (similar) text with the **D3.1_Preliminary_HW_node.** This is due the fact that these documents are highly related. While reading both documents this can be annoying, but it eases the reader of one document.

FRACTAL is an ambitious project to design a cognitive edge node that is capable of learning how to improve its performance against the uncertainty of the environment. In the project proposal, we had 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.

While the development of both the hardware and software of the node architecture is the primary goal of WP3 that addresses (Objective 1) it can be seen that the node plays an essential part of the developments of the other objectives as the technical developments in WP4/5/6 are expected to be demonstrated around the hardware nodes described as part of this deliverable in several use cases.

From the inception of the project, a key aspect was to identify a hardware platform that could lead to commercialization within time and cost limitations of the project. It was important to offer a mature platform to end-users for the integration and assessment of their use cases already at the start of the project, as well as a relatively short path towards commercialization of the FRACTAL approach. Project partners had already identified Xilinx VERSAL platform supported through FRACTAL partner PLC2 as the most suitable SoC system as the **commercial hardware node**. As a commercial system with significant resources, Xilinx VERSAL is able to fulfill the performance requirements of even the most challenging use case and project considered within FRACTAL while offering a mature development environment based around an industry standard design flow.

While the commercial platform offers many advantages, especially in short-term commercialization efforts, such a platform presents some well-established solutions, and the customization options are limited within the processing system (known as PS within a Xilinx MPSoC system). In order to increase customization, practically without any additional constraints, a **customizable node** designed around the open RISC-V instruction set architecture (ISA) and based on the open-source PULP platform (maintained by partner ETHZ) was added. This customizable node allows FRACTAL partners a powerful and flexible starting point for the development of custom nodes and a viable path for longer-term product development without an early commitment to a proprietary ISA and platform. The customizable node was also offered in an extremely flexible FPGA-based development platform where resources in the node, as well as their organization, can be adapted as needed to enable a larger range of tradeoffs.

The status of both of these platforms will be discussed in Section 3 with some detail. Moreover, at the beginning of the project, especially during the work done for "D2.1 Platform specification", it became clear that especially in the initial stages of the project the official FRACTAL nodes had to be augmented by other complementary platforms as well. There are two main reasons for this:

- Supply chain issues, partially as a result of global response to COVID related restrictions, has limited the availability of Xilinx VERSAL platforms, with few partners having access to the node within the first year. This is of course a temporary issue and all FRACTAL partners that need access to a Xilinx VERSAL platform are expected to get one sooner than later.
- While the customizable node offered an interesting alternative, especially as it supported a RISC-V based open-source solution, some of the use case requirements were geared towards higher end solutions that exceeded the capabilities of the IoT based platform made available. Some partners realized they could use other RISC-V based platforms and still remain compatible with the FRACTAL platforms in the longer term. We will describe these platforms and their rationale in Section 3.3.

A further discussion centered around how the FRACTAL software nodes would be demonstrated in use cases is discussed in the following subsection 2.1 while the individual solutions currently in plan for use cases will be covered in Section 5.

2.1 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 Section 3.2 the basic customizable node has been targeted towards simpler IoT applications and lacks the power to fulfill 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

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 allocation 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 in Section 3, partners have suggested several alternatives for the medium-end nodes.

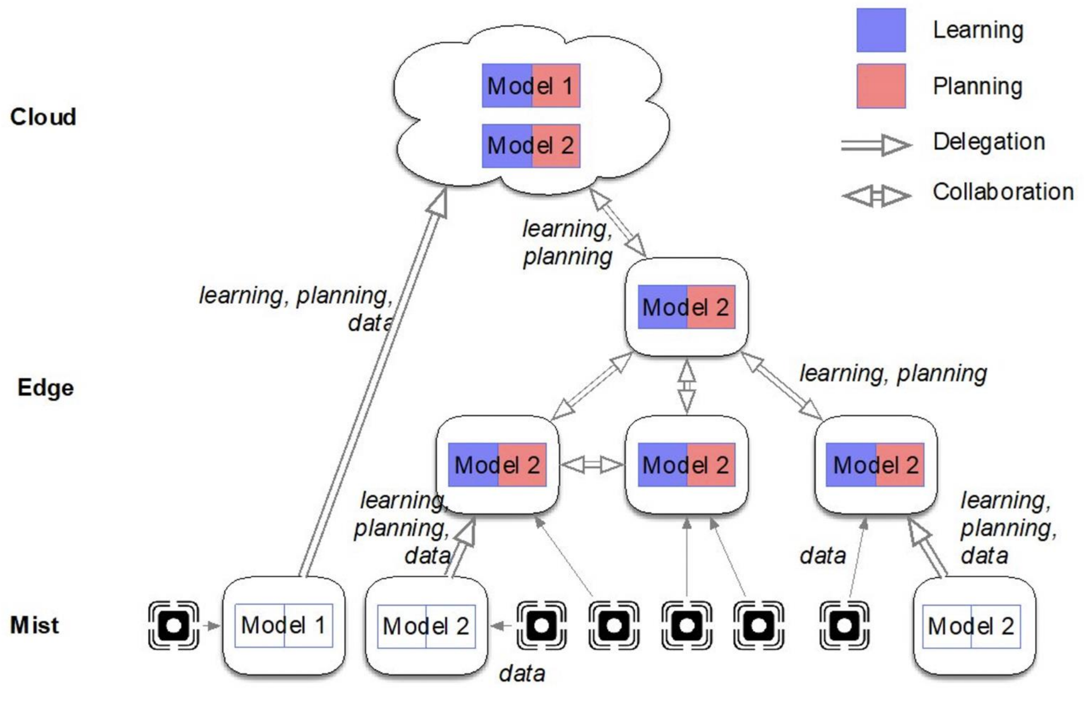

Figure 1. A schematic drawing of a possible FRACTAL system deployment using three different tiers of FRACTAL hardware nodes with different capabilities (drawing from WP5 technical meetings).

- 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.

3 Software to the FRACTAL nodes

As stated earlier, a great part of the work to be done for the FRACTAL software nodes is focusing on enabling and adapting the low-end optimizations to them.

Figure 1 presented the three main categories of the FRACTAL nodes: Cloud, Edge and Mist. The cloud computing platform is considered abstract from software point of view, but the Edge and Mist have technical constrains to be considered based on the requirements derived from FRACTAL use cases and other technical work packages, and the fact that they are embedded HW platforms

- In most of the cases the Edge nodes are implemented by **Commercial Node** (Xilinx Versal)
- The Mist nodes may be implemented by **customizable node** (Pulp).

Additionally, some other platforms can be used.

In principle, adequate abstraction/adaptation SW layers hide the HW differences from the FRACTAL applications. However, when HW is lacking resources, the only solution is to limit FRACTAL node capabilities. Two methods are used:

- Offer a POSIX compatible OS for the HW platforms that are not able to run Linux, so some of the applications can be compiled to target
- Offer an asymmetric multiprocessing for systems where Linux lacks real-time performance or low-power operations (to be considered in subsequent versions of the deliverable).

If we look at the project use cases from the demonstration point of view, the Xilinx **commercial hardware** is more attractive than the Pulp **customizable node**, due to the fact of provided onboard resources and development tools – that is, the demonstrations are much faster and easier to implement. However, if we think of commercializing these applications, the Pulp platform offers much cost and energy efficient outcome. As the reader may have guessed, the onboard resources are limited in the **customizable node** and development will require additional steps.

This section describes the state of the node's software used within FRACTAL, which would be updated in subsequent versions of this deliverable (D3.4, D3.6).

3.1 Commercial node (Xilinx VERSAL)

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.

The VERSAL architecture (Figure 2) combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC).

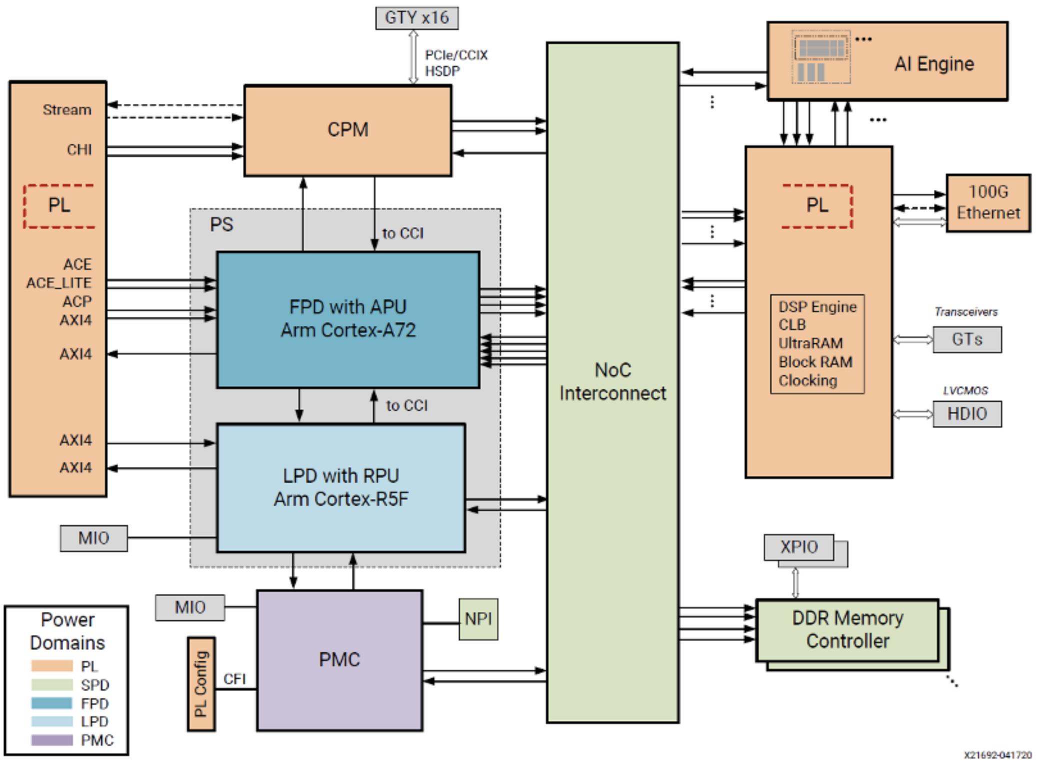

Figure 2. Top level schematic of Xilinx VERSAL ACAP

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.

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 3).

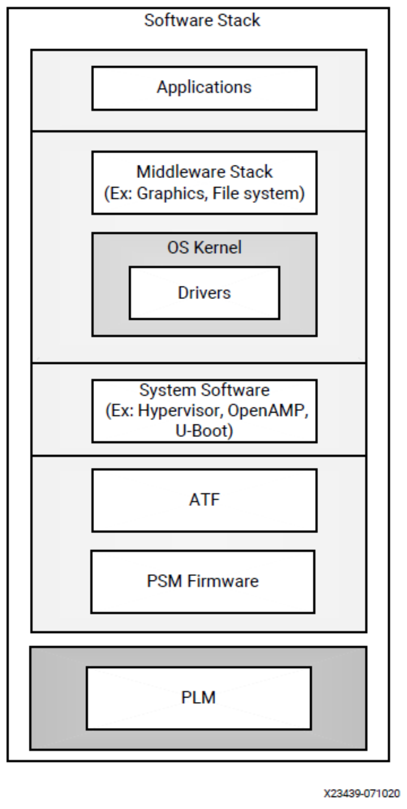
Software Stack
Applications
Middleware Stack (Ex: Graphics, File system)
OS Kernel
Drivers
System Software (Ex: Hypervisor, OpenAMP, U-Boot)
ATF
PSM Firmware
PLM
X23439-071020

Figure 3: 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 following deliverables (D3.4 and 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.

3.2 Customizable node (PULP based)

A POSIX compatible Nuttx RTOS will be integrated to the selected Pulp platforms (See Section 3. at D3.1_Preliminary_HW_node.xxx). As such it will offer standard GNU C/C++ development environment for the application developer.

For developer tools such as GDB/JTAG offer test/debug features on platform.

While the PULP HW is highly customizable, various accelerations can be easily added. The SW Node support for these accelerations is often application specific and may drop outside FRACTAL scope, however service level accelerations such as AI (WP4) are studied.

3.3 Other customizable nodes

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.

3.3.1 NOEL-V

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.

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.

3.3.2 ARIANE/CVA6

Ariane/CVA6 supports Linux, both 32-bit on CV32A6 and 64-bit on CV64A6.

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 is ongoing 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.

4 Supporting FRACTAL developments on safety, security, low-power and cognitive awareness

4.1 Supporting FRACTAL developments on low-power

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

4.1.1 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 off itself, 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 2, 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 allow 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.

4.1.2 RTOS based systems

By nature, RTOS based systems offer good low-power operations. In addition to Nuttx system level low-power features, extra features may be coded in application, but also in HW.

4.2 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.

4.2.1 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 4: A schematic of the software-only support for diverse redundancy. Figure already included in [AAA+21]**¡Error! No se encuentra el origen de la referencia.**). In particular, the monitor process spawns the redundant execution of the application in two cores, being 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.

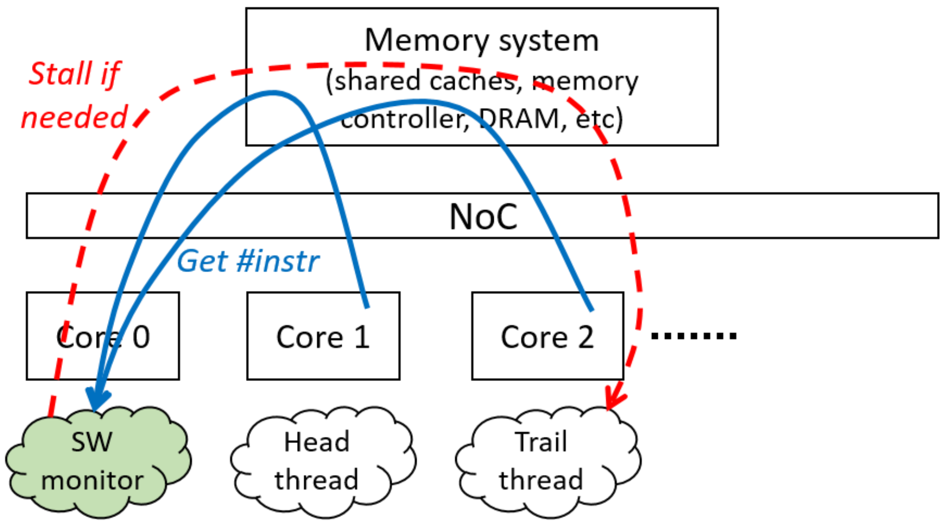

Figure 4: A schematic of the software-only support for diverse redundancy. Figure already included in [AAA+21]

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. The 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, the 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.

The activities carried out by UNIMORE within T3.4 concern the realization of a software infrastructure that allows the study of memory interference on the reference platforms, that are the Zynq UltraScale + and Versal ACAP. The software architecture adopted is the following:

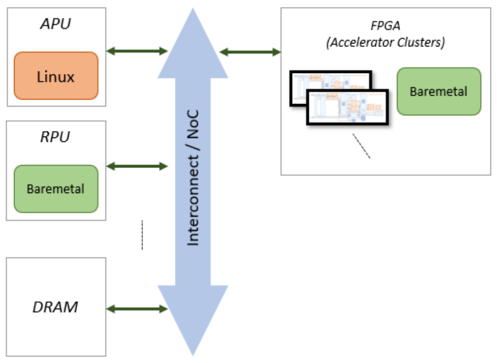

Figure 5 Software Architecture Adopted

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.

4.3 Supporting FRACTAL developments on security

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

4.3.1 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.

4.3.2 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 most cases security features are application specific implementations.

4.4 Supporting FRACTAL developments on AI

A number of 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 number of 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.

4.5 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.

4.5.1 LEDEL to develop and execute AI-based model in a FRACTAL node

In the Figure 6 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 on 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.

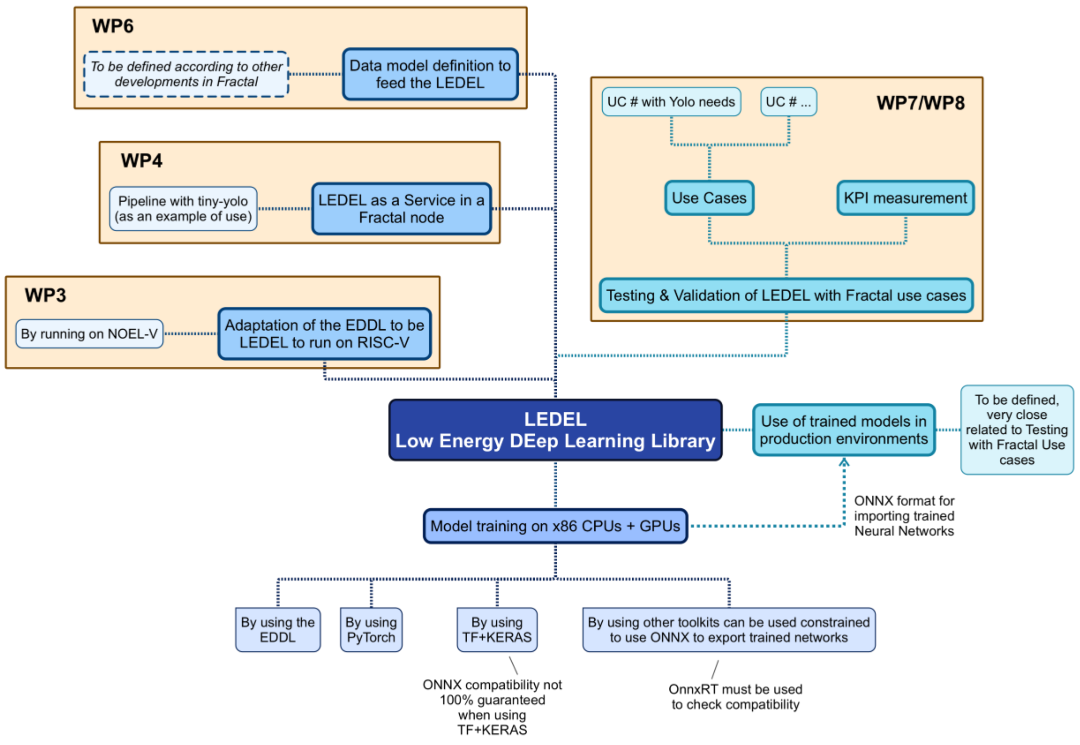
WP6
To be defined according to other developments in Fractal
Data model definition to feed the LEDEL
WP4
Pipeline with tiny-yolo (as an example of use)
LEDEL as a Service in a Fractal node
WP3
By running on NOEL-V
Adaptation of the EDDL to be LEDEL to run on RISC-V
UC # with Yolo needs
UC # ...
WP7/WP8
Use Cases
KPI measurement
Testing & Validation of LEDEL with Fractal use cases
LEDEL
Low Energy DEep Learning Library
Use of trained models in production environments
To be defined, very close related to Testing with Fractal Use cases
ONNX format for importing trained Neural Networks
Model training on x86 CPUs + GPUs
By using the EDDL
By using PyTorch
By using TF+KERAS
By using other toolkits can be used constrained to use ONNX to export trained networks
ONNX compatibility not 100% guaranteed when using TF+KERAS
OnnxRT must be used to check compatibility

Figure 6 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/downlo ad?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 work completely fine. And some few simple tests have been executed checking their proper behavior.

The LEDEL model follows a strategy similar to the one explained before in Subsection 4.4. 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 using 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 docker. Next steps involve to compile and use the LEDEL in real hardware platform NOEL V.

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 demonstrations with simulated nodes.

The FRACTAL Pulp node SW development progress 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 progress the service layers AI, Connectivity, Security will be integrated. 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 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 in providing such solutions in agreement with other partners from technical WPs 4/5/6 as well as the UCs.

6.1 Next steps

The FRACTAL PULP node SW development is done by bottom-up approach. First the the basic features are ramped up. This includes the board OS-wake-up with simple LED-Blink demonstrations and setting up SW-repositories and tools. Secondary target will be the integration/porting of service layers, such as AI, connectivity, security. Finally, the integration of use cases can begin.

Fractal Versal platform the work is focusing on adaptation layers and developer introductions.

On Versal (and on additional HW platforms) the use case development can begin immediately in parallel.

6.2 Risks and Mitigation plans

The ***D3.1_Preliminary_HW_node (chap 6.1)***, presents a set of risks and challenges (performance/cost/availability) related on the FRACTAL HW nodes. Same challenges are valid also on node SW. Limited access to actual node HW will delay the adaptation SW development and integration, thus it could potentially delay the project. By utilizing other HW platforms and simulations the work can be started.

7 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.

8 List of figures

9 List of tables

N/A

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 Access
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
MPSoC	Multiprocessor 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