

EnerMan

Energy Efficient Manufacturing System Management

D2.1 – Preliminary version of EnerMan Data Collection and Management Components

Date : 28/02/2022

Deliverable No : 2.1

Responsible Partner : Industrial Systems Institute/Research Center ATHENA

Dissemination Level : Public



HORIZON 2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **958478**



Short Description	
This deliverable gives an overview of the EnerMan project preliminary activities of WP2 T2.1 and T2.4. It consists of 5 sections that provide a description of the EnerMan edge/end node architecture and execution environment, of the preliminary applications that can be executed in this environment including security applications on cyber-attack prevention and detection as well as a demonstration of some of the described applications in action.	

Project Information	
Project Acronym:	EnerMan
Project Title:	ENERgy-efficient manufacturing system MANagement
Project Coordinator:	Dr. Ing. Giuseppe D'Angelo CRF giuseppe.dangelo@crf.it
Duration:	36 months

Document Information & Version Management			
Document Title:		Preliminary version of EnerMan Data Collection and Management Components	
Document Type:		Report and Demonstration	
Main Author(s):		Apostolos Fournaris (ISI) Alexander El-Kady (ISI) Evangelos Haleplidis (ISI) Alexander Gkillas (ISI) Aris Lalos (ISI)	
Contributor(s):		Giannis Morianos (TSI) Andreas Miaoudakis (STS) Panagiotis Rodosthenous (ITML) Mina Marmpena (ITML)	
Reviewed by:		Panagiotis Katrakazas (MAGGI) Dominik Leherbauer (FHOOE)	
Approved by:		Marco Costantino (CRF)	
Version	Date	Modified by	Comments
0.1	10/11/2021	Apostolos Fournaris (ISI)	ToC and first draft
0.2	12/12/2021	Alexander El-Kady (ISI) Evangelos Haleplidis (ISI)	Initial input by ISI
0.3	15/01/2022	Aris Lalos (ISI) Alexander Gkillas (ISI) Panagiotis Rodosthenous (ITML) Mina Marmpena (ITML)	Input on Section 3 and 5
0.4	17/01/2022	Giannis Morianos (TSI) Andreas Miaoudakis (STS)	Input on section 4

0.5	03/02/2022	Apostolos Fournaris (ISI)	Input on section 2 and section 5
0.6	08/02/2022	Apostolos Fournaris (ISI)	Final inputs have been provided
0.9	14/02/2022	Apostolos Fournaris (ISI)	Final draft submitted for internal review
1.0	21/02/2022	Panagiotis Katrakazas (MAGGI) Dominik Leherbauer (FHOE)	Review comments provided by the reviewers
1.1	25/02/2022	Apostolos Fournaris (ISI) Evangelos Haleplidis (ISI)	Final version of the deliverable
1.2	28/02/2022	Kubra Yurduseven (INTRACT)	Format Control

Disclaimer

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. The publication reflects the author's views. The European Commission is not liable for any use that may be made of the information contained therein.

Table of Contents

Executive Summary	9
1. Introduction	10
2. EnerMan end nodes/edge Architecture and execution environment	12
2.1. Overall Execution environment Concept	12
2.2. Edge/End Node Architecture Enabling the Execution Environment.....	12
2.3. Conceptual Usage of the EnerMan edge/end node Execution Environment.....	14
3. Data Processing and Assisted Intelligence.....	17
3.1. Data Harmonization	17
3.1.1. The role of data harmonization in EnerMan.....	17
3.1.2. Data harmonization targets	17
3.1.3. Technical implementation	18
3.2. Federated Learning Based Machine Status Detection	21
3.2.1. Federated Learning Introduction	21
3.2.2. Horizontal Federated Learning.....	21
3.2.3. Federated learning on non-IID Data	22
3.2.4. Personalization layer approaches	23
3.2.5. Clustering based approaches	24
3.2.6. Federated learning for machinery fault diagnosis	24
3.2.7. Problem Formulation.....	24
3.2.8. FedAvg and Network architecture	25
3.2.9. Experimental Setup	25
3.2.10. Results.....	26
3.3. Indoor Industrial space Mean Radiant Temperature distribution (MRT) estimation using infrared thermography images	27
3.3.1. Different Scenarios examined in literature.....	27
3.3.2. Mean Radiant Temperature Calculation using the IR camera	28
3.3.3. Calculation of View Factors.....	28
3.3.4. Case study	29
4. Edge Node Security Aspects	30
4.1. Attack Threat Model	30
4.1.1. ICS threats	30
4.1.2. Threat Agents	31
4.1.3. Security Requirements	33
4.1.4. Security Architecture	34

4.2.	Security Mechanisms	35
4.2.1.	Cybersecurity Attack Detection	35
4.2.2.	Cybersecurity Attack Prevention.....	36
5.	Demonstration report	41
5.1.	Execution Environment Hardware accelerated application Demonstration.....	41
5.1.1.	Platform Creation with Linux system	41
5.1.2.	Python script to run.....	42
5.2.	Intelligent Data Processing Demonstration using software reconfiguration	45
5.3.	AI Industrial Intrusion Detection Security Demo	49
5.3.1.	Board set up	49
5.3.2.	Set up the host	49
5.3.3.	Train a quantized MLP with Brevitas	50
5.3.4.	Import model into FINN and compare it with Brevitas execution	52
5.3.5.	Synthesis of the accelerator and generation of the bitfile	52
6.	Conclusion	55
7.	REFERENCES	56
APPENDIX 1.	EnerMan Execution Environment creation Workflow	57

TABLE OF FIGURES

Figure 1 Initial EnerMan Data Aggregator architecture 11

Figure 2. The implementation setup for the Data Collection and Control Plane..... 14

Figure 3 Conceptual usage of EnerMan edge node and interaction with the remaining EnerMan framework 15

Figure 4 EnerMan data collection stages: (A) use-case, (B) WP2-T2.1 (edge), (C) WP3-T3.1 (cloud) .. 17

Figure 5 Examples of data representation inconsistencies that need to be addressed by the data harmonization component. Multiple sheets with different granularity of the same measurements, different indications for missing values, diverse timestamps format, date and time split in different columns..... 18

Figure 6 The data harmonization modules are integrated in the Data Aggregator edge node and communicate with the Big Data Analytics Engine in the cloud..... 19

Figure 7. Preliminary Data Model sample 20

Figure 8. A demonstration of data partition in horizontal federated learning. In this example the two clients contain five personal features, namely name, age, sex, height and weight. However, each client has data for different persons. 22

Figure 9. An illustration of horizontal FL with personalization layers. Only the base layers (filled blocks) are uploaded to server for the global model aggregation..... 233

Figure 10. The proposed fault diagnosis deep learning model..... 25

Figure 11. Description of the CWRU dataset. 25

Figure 12. The proposed fault diagnosis deep learning model..... 26

Figure 13. MRT distribution maps in a pedestrian space [4] 28

Figure 14. View factors calculation 28

Figure 15. Modelled enclosed room [3] 29

Figure 16. MRT distribution maps [3]. The wall surfaces are divided into smaller surfaces with different sizes to determine the optimal value regarding the accuracy of the MRT calculation..... 29

Figure 17. The EnerMan Security Architecture 35

Figure 18. Logical representation of SNORT architecture flow..... 36

Figure 19. OAuth authentication and authorization flow..... 38

Figure 20 Hardware Security Token edge-node cryptography based cyberattack prevention setup.. 40

Figure 21. Python code for executing the vadd application and IP core 43

Figure 22. Terminal output of the register map..... 44

Figure 23. Terminal output of the final output of vadd 455

Figure 24. The ULTRA96 Board that is used in this Demo 46

Figure 25. Training loss per iteration..... 50

Figure 26. Test accuracy per iteration 51

Figure 27. The exported ONNX model in Netron 52

Figure 28. The steps towards the bitfile generation 53

Figure 29: Block design architecture in Vivado 53

Figure 30. Terminal Output..... 54

LIST OF ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
BDAE	Big Data Analytics Engine
CPS	Cyber-Physical System
CPSoS	Cyber-Physical System of Systems
CWRU	Case Western Reserve University
DL	Deep Learning
DMZ	DeMilitarized Zone
DoA	Description of Action
DoS	Denial of Service
ENISA	European Union Agency for Cybersecurity
FIFO	First In First Out
FL	Federated Learning
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HLS	High Level Synthesis
HST	Hardware Security Token
HTTP	HyperText Transfer Protocol
I2DS	Industrial Intrusion Detection System
ICS	Industrial Control Systems
IDS	Intrusion Detection System
iDSS	Intelligent Decision Support System
IETF	Internet Engineering Task Force
IFCA	Iterative federated clustering algorithm
IID	Independent and Identically Distributed
IIoT	Industrial Internet of Things
IP	Intellectual Property
IPS	Intrusion Prevention System
IR	InfraRed
JWT	JSON Web Token
MITM	Man In The Middle
ML	Machine Learning
MLP	MultiLayer Perceptron
MPSoC	MultiProcessor System on a Chip

MRT	Mean Radiant Temperature
OAuth	Open Authorization
ONNX	Open Neural Network Exchange
OT	Operational Technology
PKCE	Proof Key for Code Exchange
PKI	Public Key Infrastructure
PL	Programmable Logic
PLC	Programmable Logic Controller
PS	Processor System
QNN	Quantized Neural Network
ReLU	Rectified Linear Unit
REST	Representational state transfer
SCADA	Supervisory Control and Data Acquisition
SoC	System on a Chip
TLS	Transport Layer Security
UNSW	University of New South Wales
WP	Work Package
XRT	Xilinx RunTime
XSS	Cross-Site Scripting

EXECUTIVE SUMMARY

This deliverable is focused on the preliminary activities of WP2 and especially in the activities of Task 2.1 (T2.1) and Task 2.4 (T2.4). Initially we provide an introduction of the overall approach in the WP2 and then in section 2 we present the analysis on Execution Environment of the EnerMan edge/end node along with the node's architecture that supports such an environment. Afterwards, in section 3 the preliminary applications that have been developed in WP2 as those have been prescribed in Task 2.2 are being briefly presented however, we do not provide thorough analysis on them since there is a dedicated deliverable on T2.2 on M18. Similarly, we do not deliberate on the activities of T2.3 since there is a dedicated deliverable report on M18 for that task. In section 4 we focus our analysis explicitly on the security aspects that are linked with the EnerMan edge layer based on the activities of Task 2.4. Finally, in Section 5 we demonstrate the usage of the EnerMan execution environment for various scenarios and showcase how some applications described in Section 3 and Section 4 are implemented in action. The deliverable is concluded with an appendix that presented the custom design flow that was used in order to create the EnerMan execution environment on the edge/end node that is implemented on a Xilinx MPSoC device.

1. INTRODUCTION

The second work package of the EnerMan project is focused on the data collection and data processing at the end devices and edge level of the EnerMan framework. This means that in this WP we are going to research, design and implement all the relevant components of the EnerMan data collection and Control plane on the edge of the industrial manufacturing infrastructure. Also, we are going to create the necessary computing and execution environment that will allow the appropriate deployment, execution, and efficient operation of such components. Our goal in this WP is to create an EnerMan intelligent Cyber-Physical System (CPS) end node that will act as an end device or/and as a data aggregator for a series of in-field devices (machines) within the industrial environment. Apart from simple data collection, the EnerMan node should be able to perform intelligent operations that can support specialized industrial functionalities (e.g., Predictive maintenance or intelligent temperature measurement or energy consumption local data predictions) to pre-process and fine-tune data that are going to be forwarded to the EnerMan system layer (as a private or a public cloud big data analytics engine etc.). We adopt, as an execution environment, embedded system solutions that have Multiple Processor System on Chips with dedicated FPGA fabric that can offer custom to our needs hardware acceleration and hardware level (programmable logic) reconfiguration, to support flexibility in the execution of the various end/edge node operations and also to offer a high level of efficiency. Apart from the above, in WP2 we also considered the need for an edge device-based control loop mechanism that will collect the needed configuration from the EnerMan system layer (e.g., from the intelligent Decision Support System, iDSS) and forward it to the factory automation processes (e.g., PLCs or other control (actuation) devices). Given that reconfiguration is also supported by the EnerMan edge node (software and hardware based) the control loop should also be able to offer control of the EnerMan edge node functionality and how such functionality can be reconfigured over time (during operation) according to the EnerMan platform suggestions.

Following the Description of Action (DoA) in the General Assembly of the EnerMan project, the WP2 is meant to provide the necessary execution environment for performing data collection and processing at the edge of the industrial infrastructure so that we can deploy in such an environment the EnerMan software agents aiming to do holistic data processing using diverse sensing modalities for specific industrial functions as those are specified by the EnerMan user requirements. Of course, goal of WP2 is also to create such software agents (envisioned as small software programs executed in the EnerMan execution environment). Apart from those actions in WP2 we are designing and implementing the edge IIoT level support mechanism for the EnerMan flexible control loop and protect the overall data collection mechanism against security and privacy breaches using dedicated security operations aiming to act as proactive (to prevent security breaches) and reactive (to detect security attacks) measures.

As can be seen in Figure 1 the WP2 designed and developed edge node is going to act as an EnerMan intelligent CPS node that will collect sensor data and as a data aggregator that will harmonize and preprocess such data to be ready for the big data analysis performed at the cloud level of the EnerMan architecture. During the initial user requirements and architecture requirements phase of the EnerMan project (performed in WP1) it became apparent that the two roles can be merged into a unified CPS component (the EnerMan intelligent node) that can include both roles originally described in the DoA document (i.e., data collection/aggregation, data harmonization and preprocessing).

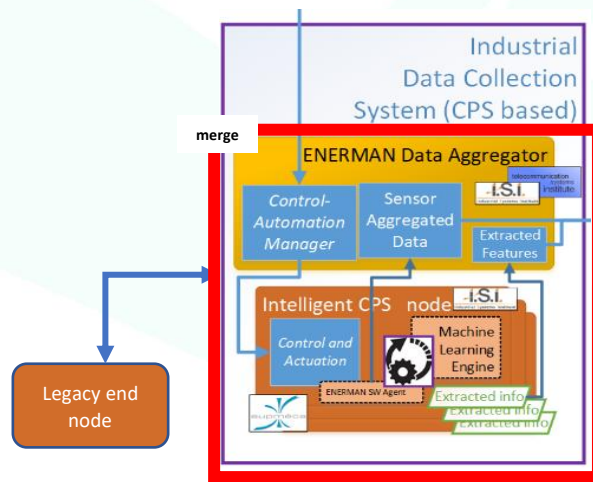


Figure 1. Initial EnerMan Data Aggregator architecture

The data that we currently consider at the edge level (either raw data or data after preprocessing) can be the following

- **Machine energy consumption**
- **Multiple sensory data (temperature, pressure, humidity, etc.) from existing pilot deployed sensors**
- **Machine functionality status for predictive maintenance (machine faulty state)**

The above activities of WP2 span in 4 tasks (Task 2.1 to Task2.4). Task 2.1 is focused on setting up the appropriate execution environment and the research and development of how to efficiently deploy the Edge node intelligence and security functionality into the execution environment. Task 2.2 is focused on the design and development of the appropriate intelligence algorithms (Machine Learning based) to be deployed in the EnerMan intelligent edge node. Task 2.3 is about the realization of the EnerMan control loop and the reconfigurability that is supported by the EnerMan architecture. Task 2.4 is about the establishment of all the data security functionality that will prevent security breaches of the collected, processed, and transmitted data from the edge node to the EnerMan cloud solutions. Given that according to the EnerMan workplan there are dedicated deliverables for Task 2.2 and Task 2.3, in the deliverable (D2.1) we report the preliminary activities of Task 2.1 and Task 2.4. We also provide a brief description of the Task 2.2 activities (that are currently in progress) since those are linked with the Task 2.1 activities and the structures that are implemented there.

Thus, in this deliverable, we focus on the activities that are in progress till Month 14 of the project (note that WP2 and the relevant tasks are concluded in M18). These activities are:

- The research, design, and development of the appropriate execution environment in the EnerMan CPS intelligent edge node
- The capabilities that this execution environment can provide at the current state of the project
- The algorithms that are currently designed and under deployment in the created execution environment
- The security functionality that such an environment can currently support

Finally, in the deliverable, we provide in a tutorial-like fashion the workflow to be followed to deploy an application on the developed execution environment including hardware acceleration/reconfiguration support.

2. ENERMAN END NODES/EDGE ARCHITECTURE AND EXECUTION ENVIRONMENT

2.1. Overall Execution environment Concept

One of the fundamental activities of WP2 (reflected in T2.1) is to design the appropriate execution environment for the EnerMan end nodes and data aggregators that will enable the easy deployment and usage of the EnerMan edge intelligence as well as the control loop reconfiguration (designed in T2.2 and T2.3). Given that we are aiming to provide highly efficient data processing at the edge as well as the maximum possible reconfiguration, it is imperative that we structure the execution environment to include mechanisms that will be able to offer such services.

In general, the execution environment of the EnerMan end/edge node should include several components that will allow the hardware and software support of the EnerMan edge functionality. Given that the EnerMan project aims at providing reconfiguration of the EnerMan edge functionality, the EnerMan execution environment should also be able to support such service at the hardware level (using FPGA programmable logic) and at the software level. Thus, before actually deploying specific algorithms (as those are specified in T2.2) we need to provide an execution environment that can allow the easy deployment, configuration and reconfiguration of such algorithms in hardware and in software. The EnerMan software agents, which constitute, the operations to be executed in the EnerMan end/edge nodes will rely exclusively on the capabilities of such execution environment.

2.2. Edge/End Node Architecture Enabling the Execution Environment

We envision the EnerMan intelligent Edge node as a heterogenous embedded system device that can perform multiple activities within the manufacturing infrastructure in an efficient manner. This edge device should be able to collect the data that are provided to it by the various sensors existing inside the industrial domain, harmonize those data in order to be compatible with the data expectations required by the EnerMan big data analytics engine and in parallel also process those data so as to offer the EnerMan platform as well as the operator appropriate information that will help them make informed decisions on the optimal energy sustainability options. This indicates that the Data collection and Processing should be manifested in the EnerMan intelligent edge node with various ways and in an efficient manner. Furthermore, the node should be versatile enough to offer a broad range of services that may vary from time to time and from industry to industry. This highlights, apart from high efficiency, the need for flexibility in the offered operations of the node. To keep efficiency at high level regardless of the configuration of the EnerMan edge node we opt for support of reconfigurability both in hardware and software. This gives the ability to change, at a reasonable degree, the functionality of the EnerMan intelligent edge node with respect to the underlined industry needs without having to redesign/remanufacture the node itself. In practice, this means that the execution environment of the EnerMan intelligent edge node should be able to handle such reconfigurability and efficiency features. From a hardware perspective, we consider as a best match to the above requirements, the use of MultiProcessor System on Chip (MPSoC) embedded system devices that can host in its core, multiple processors (usually multicore processors) to achieve efficiency but also specialized hardware components for specific applications (e.g., Graphic Processing Units (GPU) or real time processors). To further support hardware reconfigurability, we consider embedded system MPSoCs that fathom in their SoC architecture, reconfigurable hardware programmable logic in the form of an FPGA fabric. The latest FPGA manufacturer solutions are more than capable of supporting the above-described setup. In EnerMan we opt for Xilinx manufacturer devices focusing on the Xilinx Zynq Ultrascale+ MPSoC design as this is realized in two low-end, low-cost embedded system boards i.e., the Xilinx ZCU 104 or 102 development board and the Avnet ULTRA96 development board. The

advantage of such development boards is that they can be used both as means for prototyping as well as in the context of the actual final implementation of the EnerMan intelligent edge node.

In Figure 2 the overall architecture and concept of the EnerMan intelligent Edge node is presented. The node will accept and process data from the distributed sensors using a platform that operates both in the software and hardware domains. These data are processed by the MPSoC unit that runs a Linux-based OS, i.e., PetaLinux, on its software side and uses its PL to implement specific types of IP cores, i.e., functional modules, with optimized processing and energy consumption metrics, on its hardware side. Naturally, the OS is equipped with all the necessary firmware for software to hardware communication provided by the Xilinx Runtime (XRT) library that accompanies the embedded OS distribution. On top of the OS, however, we have implemented and configured additional reconfigurability/flexibility features to support the EnerMan node requirements. To achieve software reconfigurability we introduce in the execution environment Docker based containerization, i.e., Docker containers, that will allow for the introduction to the node, input from the other EnerMan planes such as the Management plane as well as support of functionality that cannot be offered by the Xilinx supported PetaLinux OS. In other words, the data collection and control plane, will support an interactive relationship between itself and the other EnerMan framework components such as the other planes as well as the sensors and actuators.

Furthermore, given that many Machine Learning and Deep Learning core software libraries are implemented in the python programming language, in the EnerMan execution environment we integrate the PYNQ python library offered by the Xilinx tool into the PetaLinux OS environment as an alternative mechanism of supporting hardware (FPGA PL) reconfiguration. Note, that existing Xilinx solutions offer hardware reconfiguration through PetaLinux either using the native Xilinx runtime (XRT) or using PYNQ python Jupyter notebooks, not both. We have managed to employ both such approaches in the EnerMan edge/end node execution environment. Beyond that, we have managed on top the above two hardware reconfiguration approaches, software reconfiguration through Docker containers.

As shown in Figure 2 the EnerMan edge/end node architecture consists of 4 different layers. The first layer, hardware layer, includes the FPGA fabric (the programmable logic) on which using the EnerMan execution environment capabilities (through the XRT or the PYNQ python library) we can deploy the hardware part (the designed IP Cores) of an EnerMan application (as will be demonstrated in section 5 of this deliverable). Above the hardware layer, there is the processor layer that is executing the software part of an EnerMan application (e.g., the EnerMan software agents). The EnerMan applications themselves are deployed using the PetaLinux OS environment that constitute the OS layer of the EnerMan edge/end node. On top of this layer, we have built the overall EnerMan execution environment functionality that as mentioned in the previous paragraphs includes the Docker container support and the PYNQ library support that has been adapted to be used in parallel with the XRT environment. The above capabilities rely on the EnerMan application backbone that includes the necessary core, backend, functionality to enable the described execution environment. Finally, this application layer, includes the necessary software functionality to securely transmit the collected data or relevant extracted data using the TLS1.3 protocol that is enhanced with quantum – safe security capabilities to support long lasting and strong security.

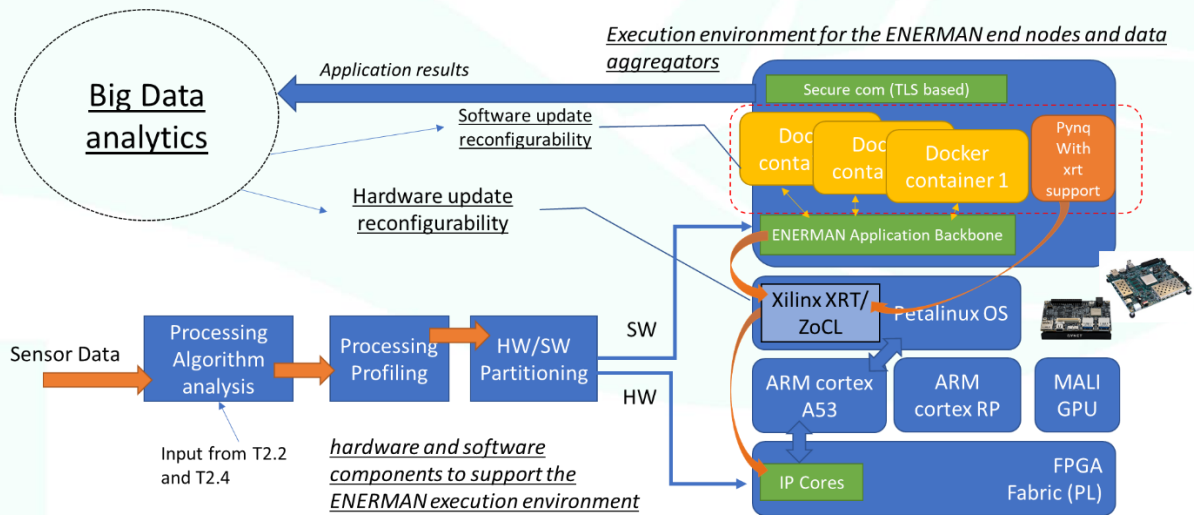


Figure 2. The implementation setup for the Data Collection and Control Plane

To fully employ the efficient execution of an algorithm of the EnerMan software agent (in the form of an executable application) we follow the process shown in the lower left side of Figure 1. At design time we obtain the algorithm software implementation and place it in a controlled environment where we use various computation profiling tools like control flow graph analyzers showing the dependence between functions of the algorithmic implementation and their time delay or flame graphs showing the memory usage and memory depth of such function during execution. The goal is to identify computationally demanding or slow execution functions (or software code in general) and following a crude hardware /software partitioning reassign the execution of such functions on dedicated hardware Intellectual Property (IP) Cores that are deployed within the EnerMan edge node execution environment.

2.3. Conceptual Usage of the EnerMan edge/end node Execution Environment

The EnerMan edge node and its execution environment is used in accordance with the overall EnerMan framework/platform as shown in Figure 3. The EnerMan edge/end node collects input from the pilots in the form of datasets (the format varies from pilot to pilot), processes that input and forwards the result to the system layer of the EnerMan framework (deployed in public or private cloud). Typically, the consumer of the EnerMan edge/end node results is the EnerMan Big Data Analytics Engine. Also, the EnerMan edge/end node is acting as an enabler of control configurations that stem from the EnerMan intelligent Decision Support System (iDSS) by forwarding those configurations to the pilot site infrastructure. In practice, given the EnerMan pilot site constraints (and not to disturb the actual factory production lines/process) the actuation configuration is forwarded to the Factory human personnel for evaluation (acting as suggestions).

Most importantly in Figure 3, the currently identified preprocessing applications (acting as the EnerMan edge/end node software agents) are being shown and their interaction with the overall EnerMan framework is briefly presented. Typically, each pilot collects data from the in-field sensors deployed in the factory and stores them in some data collection point. The EnerMan edge node acting as data aggregator interacts with that collection point and consumes such data by initially transferring them (in a synchronous or asynchronous way depending on the pilot needs) to edge node storage area and then harmonizing them using a data harmonization component. This component's goal is to prepare the pilot data in order to be fit for the EnerMan platform, mainly for the EnerMan big data analytic engine. The EnerMan edge node data harmonizer produces datasets that can be either directly forwarded to the Big Data analytics Engine or can be further preprocessed within the EnerMan

edge/end node. The preprocessing performed at the edge is associated with specific ML/DL based operation that follow the federated learning model. Each EnerMan edge/end node in such case acts as a client of an AI federation that has a local ML/DL model. This local model is trained using the data that are provided locally to each EnerMan edge/end node and when data classification is made the accuracy of the results is limited due to the volume and quality of this local dataset. However, to improve the overall classification (or even prediction) accuracy periodically the client local models are forwarded to a Federated Learning central server that compose the local models into a Federated global model that can provide considerably more accurate classification compared to single local models. The global model is then shared with all clients thus updating their local models and the training process resumes while classification is taking place. The overall process is further analyzed in section 3 where we also describe a demonstration of the approach for machine health status classification (in the overall concept of predictive maintenance) as part of a Deep Anomaly Detection classification concept.

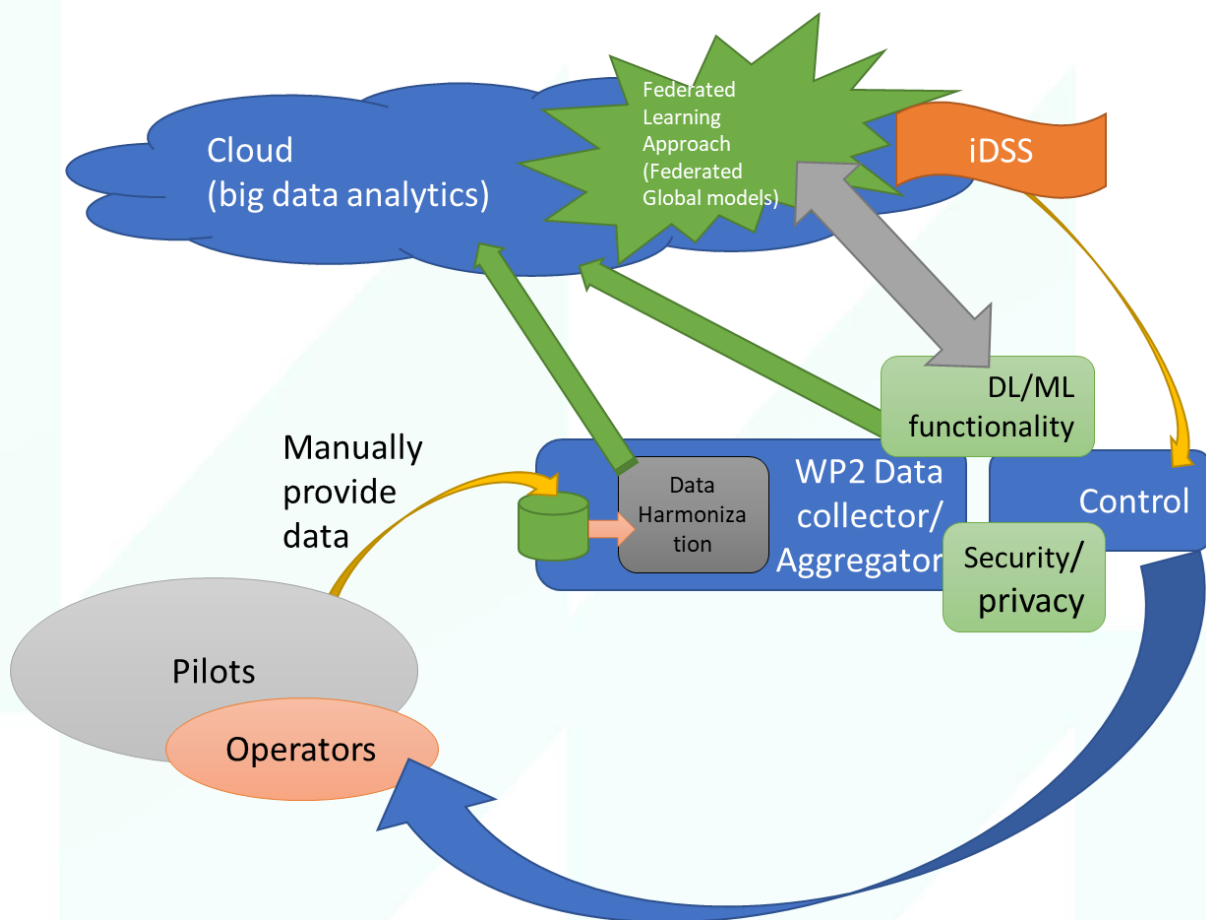


Figure 3 Conceptual usage of EnerMan edge node and interaction with the remaining EnerMan framework

Apart from the processing of datasets from pilot's sensors, the EnerMan edge/end node can also process data that are collected for specific reasons from sensors of the edge/end node itself. Infrared thermal camera images or videos from the manufacturing space is a good candidate for the capabilities of the EnerMan execution environment. Using such data, we can infer the Mean Radiant Temperature distribution on a given industrial manufacturing space and eventually visualize this information in the EnerMan visualization framework.

Notably, the harmonized data as well as the data produced after preprocessing at the edge are transmitted to the rest of the EnerMan framework securely. Furthermore, we also make sure that the

EnerMan edge/end node is protected against security attacks that can potentially maliciously modify (or poison) the data processed within the EnerMan edge/end node. In general, the EnerMan execution environment is supporting several security services focusing on cyberattack prevention and detection. More information on the EnerMan edge/end node security is provided in section 4 of this deliverable.

Finally, the EnerMan execution environment can handle control-configurations that are provided by the EnerMan iDSS. This activity is out of scope for this deliverable and will be analyzed in detail in the dedicated deliverable for T2.3 (i.e., Deliverable 2.4)

3. DATA PROCESSING AND ASSISTED INTELLIGENCE

3.1. Data Harmonization

Data harmonization describes the preparation of raw data originating from various sources with the goal to provide a more standardized and uniform data representation. It is an important stage in a data preparation pipeline since the raw data are produced in custom formats which are diverse and pose a challenge for the application of more advanced preprocessing technics (e.g., interpolation, resampling, clustering). The harmonization can be achieved by identifying similarities in the various data sets, retaining critical requirements, and generating a common standard. Data quality checks are also critical at this stage to ensure the data integrity and validity before data features are propagated to the next stage of preprocessing.

3.1.1. The role of data harmonization in EnerMan

Data harmonization as a component of the EnerMan platform aims to bridge data collection requirements between the edge node and the cloud infrastructure, addressed in WP2 and WP3 respectively. As demonstrated in Figure 4, data collection begins at the end-user's premises (Figure 4: A), with customized representations built internally to serve the needs of the organization (different PLCs or recorders). Subsequently, EnerMan edge nodes apply the harmonization pipelines (Figure 4: B) that ensure that the data stored in the cloud infrastructure (Figure 4: C) have an aligned representation that will allow to process them with more generic methodologies across pilots and use cases. Thus, the data harmonization component facilitates the data collection between the end users and the Big Data Analytics Engine (BDAE) in the cloud (WP3-T3.1), by implementing a data pipeline from raw data provided by the use cases to structured time-series data, available for downstream tasks.

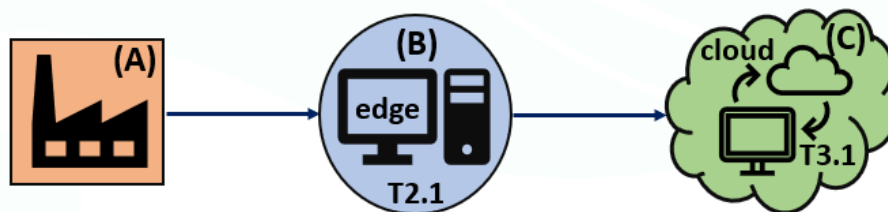


Figure 4 EnerMan data collection stages: (A) use-case, (B) WP2-T2.1 (edge), (C) WP3-T3.1 (cloud)

3.1.2. Data harmonization targets

In this section we will discuss some concrete data aspects from the EnerMan raw datasets that the data harmonization component aims to transform into a uniform representation across use cases.

File formats: Raw datasets are provided in diverse formats, e.g., .xlsx, .csv. Moreover, .xlsx files often consist of multiple sheets corresponding to various aspects of the same use case and process (e.g., same measurement with different time aggregation). The harmonization task aims to convert the files into a uniform format suitable for modelling frameworks and database ingestion processes (.csv or .parquet depending on the batch sizes).

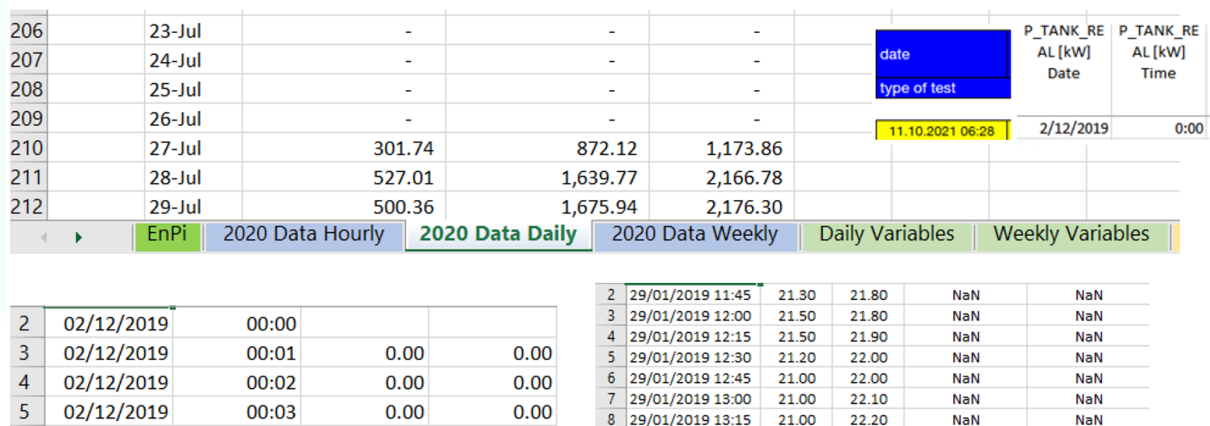
Data Schema: Attributes' names are often defined in forms that are not suitable for big data processing, e.g., they contain spaces, are split in multi-row header or are not in English. Furthermore, the data schema is not always readily transferable to a tabular format since there might be multiple headers to a single sheet or merged cells might occur.

Missing data indication: Missing value indications vary across use cases and organizations, e.g., empty cell, null, '0' or 'N/A', 'NaN' etc. Data harmonization must detect all these different representations and transform them into a uniform type, recognizable from a programming point of view.

Measurement units: Units are not always explicitly indicated. In case a data set has the same type of measurement (for example temperature) the harmonization layer needs to ensure that the unit of measurements is always the same (e.g., Celsius).

Timestamps: Another source of variation is related to the time zone of the timestamps of each measurement. It is not always evident whether the initial data collection used the local time zone or Coordinated Universal Time (UTC). Furthermore, organizations, use cases and processes utilize diverse timestamp formats (depending on their PLC or recording configurations), or even separate date and time columns breaking the timestamp in two parts. The harmonization pipelines need to change these dates into the same time zone and in unique format to improve data usability and integrity.

Metadata information: Raw datasets are not accompanied by metadata information that would be necessary for advanced preprocessing and analytics in the cloud. Examples of such information are the expected datatypes for each measurement, min-max limits or descriptive statistics for numerical values, expected classes for categorical values, expected granularity or sampling rate, localization, measurement type (e.g., sensor, actuator), how should missing values be interpreted for a particular measurement and if they should be acceptable.



Row	Date	Value 1	Value 2	Value 3	Time	Missing	Missing
206	23-Jul	-	-	-			
207	24-Jul	-	-	-			
208	25-Jul	-	-	-			
209	26-Jul	-	-	-			
210	27-Jul	301.74	872.12	1,173.86			
211	28-Jul	527.01	1,639.77	2,166.78			
212	29-Jul	500.36	1,675.94	2,176.30			

Sheet	Time	Value 1	Value 2	Missing	Missing
2	29/01/2019 11:45	21.30	21.80	NaN	NaN
3	29/01/2019 12:00	21.50	21.80	NaN	NaN
4	29/01/2019 12:15	21.50	21.90	NaN	NaN
5	29/01/2019 12:30	21.20	22.00	NaN	NaN
6	29/01/2019 12:45	21.00	22.00	NaN	NaN
7	29/01/2019 13:00	21.00	22.10	NaN	NaN
8	29/01/2019 13:15	21.00	22.20	NaN	NaN

Figure 5 Examples of data representation inconsistencies that need to be addressed by the data harmonization component. Multiple sheets with different granularity of the same measurements, different indications for missing values, diverse timestamps format, date and time split in different columns.

3.1.3. Technical implementation

Data Harmonization is implemented as a python package which can be integrated to the EnerMan at the edge node's Data Aggregator and is configured to communicate with the Big Data Analytics Engine (WP3-T3.1). A higher-level depiction of the architecture is presented in Figure 6. The main components of the Data Harmonization package are the following: dedicated harmonizers for each use case, a data loader class, a library of harmonization utilities, and a library for data quality checks.

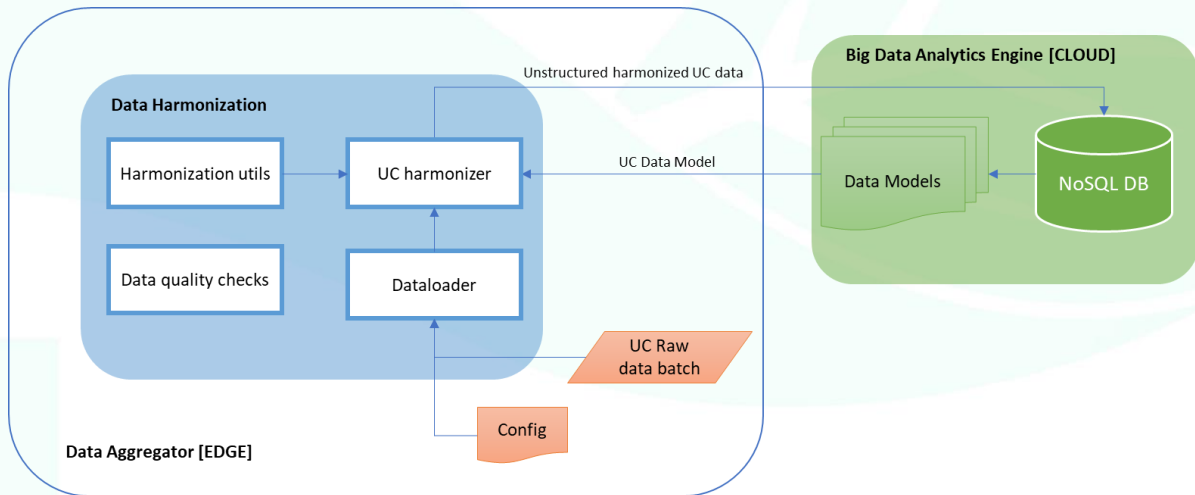


Figure 6 The data harmonization modules are integrated in the Data Aggregator edge node and communicate with the Big Data Analytics Engine in the cloud.

Data harmonization at the edge level is necessary as described in Section 3.1.1 to support the Big Data Analytics Engine, a centralized EnerMan component that is responsible for handling advanced preprocessing tasks. The data harmonization package is designed to be a lightweight solution adapted to the use case specific characteristics and separated from the cloud infrastructure which involves processes that require data that have been previously standardized. This setup enhances modularity of the EnerMan solution and a balance between specialization and generalization, addressed at the edge and in the cloud, respectively.

More concretely, the communication entails fetching the use case specific Data Model from the Big Data Analytics Engine to the edge, and in the other direction, transferring the harmonized datasets to the cloud. Data Model prototypes are designed and developed in WP3-T3.1 (a centralized approach is taken since they are expected to be used also from downstream tasks that only communicate with the Big Data Analytics Engine). A preliminary form of a Data Model is depicted in Figure 7.

```

▶ CRF_BodyshopScheduling:      {...}
▶ CRF_BodyshopTemperatures:    {...}
▼ CRF_PaintshopTankData:
  pilot_partner:                "CRF"
  dataset_name:                 ""
  dataset_path:                 ""
  process:                      "PaintshopTankData"
  format:                       "xlsx"
  missing_indication:           "NaN"
  datetime_format:              "dd/mm/yyyy hh:mm:ss"
▼ collection_site:
  country:                      "Italy"
  tz_name:                      "Europe/Rome"
▼ header:
  first_data_row:               0
▼ column_names:
  0:                            "P_TANK_REAL_[kW]_Date"
  1:                            "P_TANK_REAL_[kW]_Time"
  2:                            "P_TANK_REAL_[kW]"
  3:                            "P_TANK_CAL_[kW]_Calculated"
  4:                            "P_TANK_CHECK_[%]_GAP_between_real_and_calculated"
  5:                            "T_REAL_[C]"
  6:                            "Qv_EX_[m^3/h]"
  7:                            "T_in_EX_[C]"
  8:                            "T_out_EX_[C]"
  9:                            "T_SPreg_[C]"
▼ column_dtypes:
  0:                            "object"
  1:                            "object"
  2:                            "float64"
  3:                            "float64"
  4:                            "float64"
  5:                            "float64"
  6:                            "float64"
  7:                            "float64"
  8:                            "float64"
  9:                            "float64"
▶ CRF_PaintshopAHUData:      {...}
    
```

Figure 7. Preliminary Data Model sample

The components of the harmonization package are the following:

Harmonizers: A python class designed for each use case's data profile. It takes a use case specific Data Model as an argument and provides methods for loading the data, preprocessing it with the harmonization functions that correspond to the specific use case, and applying the quality checks on

the harmonized data. The Data Model contains information to be used by the class methods, e.g., the expected feature names, data types, localization information.

Data loader: A python class which is called from the harmonizers and handles the loading of the data. It provides methods to enable loading different data formats.

Harmonization utilities: A library of preprocessing functions to harmonize the data. For example, there are functions to replace the missing data indication with a standardized form, to transform local timestamps to UTC+00 and datetime formats to ISO 8601 (yyyy-MM-dd'T'HH:mm:ssZ). It also provides functions to transform attributes' names and data types according to a given list provided by the Harmonizer class as described in the Data Model.

Data quality checks: A library which contains functions to run basic data quality checks. For example, functions that examine whether the data schema matches the Data Model schema, the data types are the expected ones, min and max values whenever defined are not exceeded.

The final harmonized datasets are stored in an unstructured database in the cloud where they are further preprocessed in the final layer of preprocessing (WP3-T3.1) to derive time-series data.

3.2. Federated Learning Based Machine Status Detection

3.2.1. Federated Learning Introduction

Traditional distributed deep learning approaches demand a large amount of private data to be processed and aggregated at central servers during the model training stage by employing some suitable distributed optimization algorithm. However, this distributed process suffers potential data privacy leakage issues. On the other hand, federated learning (FL) has been emerging as a promising approach for decentralized model training focusing on the data privacy aspect of the problem. In particular, it allows the clients to train and acquire an accurate globally trained model without sharing any private data with other users.

The scope of the federated learning scheme is to obtain a global model, say θ (Eq. (2)) that can minimize some aggregated local function $f_k(\theta^k)$ (Eq. (1)), where x denotes the data feature, y is the data label, n_k is the local data size, $n = \sum_{k=1}^{CxK} n_k$ is the total number of sample pairs, C stands for the participation ratio assuming that not all local clients participate in each round of model updates and k is the client index.

$$f_k(\theta^k) = \frac{1}{n_k} \sum_i^{n_k} \text{loss}(x_i, y_i; \theta^k) \quad (1)$$

$$\min_{\theta} f(\theta) = \sum_{k=1}^{CxK} \frac{n_k}{n} f_k(\theta^k) \quad (2)$$

Federated learning can be divided into two major categories, namely the horizontal and vertical FL based on the characteristics of data distribution across the participants [1]. In this study, we focus on the horizontal federating learning scheme.

3.2.2. Horizontal Federated Learning

Horizon FL or homogeneous FL relates the case where the local training data of clients share the same feature space but have different sample space. Figure 1 exemplifies this scenario, where client 1 and client 2 share the same personal features (feature space) but they contain data for different persons.

Features

Name	Age	Sex	Height	Weight	Label
Person A	24	Male	178	78	1
Person B	61	Female	165	64	0
Person C	44	Male	182	89	1
Person D	17	Female	159	52	0
Person E	11	Male	137	36	1
Person F	33	Female	171	60	0

Samples

} **Client 1**
} **Client 2**

Figure 8. A demonstration of data partition in horizontal federated learning. In this example the two clients contain five personal features, namely name, age, sex, height and weight. However, each client has data for different persons.

The FedAvg algorithm [2] is a typical case of the horizontal FL providing an efficient methodology to train a global model without sharing any client's data. In more details, the global model, say θ and the local models θ_k share the same deep learning architecture with different model parameter values, since each local model is independently optimized based on its local data. After the local training, the models are uploaded to the server. Considering that all local models have the same structure, the server aggregates the local models and generates the corresponding global model θ . Algorithm 1 summarizes the FedAvg approach.

```

1: Server:
2: Initialize global model  $\theta_0$ 
3: for each communication round  $t = 1, 2, \dots, T$  do
4:   Select  $m = C \times K$  clients, where  $C \in (0, 1)$ 
5:   for each Client  $k = 1, 2, \dots, m$  in parallel do
6:     Download  $\theta_t$  to Client  $k$ 
7:     Do Client  $k$  update and receive  $\theta^k$ 
8:   end for
9:   Update global model  $\theta_t \leftarrow \sum_{k=1}^m \frac{n_k}{n} \theta^k$ 
10: end for
11:
12: Client  $k$  update:
13: Replace local model  $\theta^k \leftarrow \theta_t$ 
14: for local epoch from 1 to  $E$  do
15:   for batch  $b \in (1, B)$  do
16:      $\theta^k \leftarrow \theta^k - \eta \nabla L_k(\theta^k, b)$ 
17:   end for
18: end for
19: Return  $\theta^k$ 
    
```

Algorithm 1. FedAvg. K is the total number of clients; B is the size of mini-batches, T is the total number of communication rounds, E is the local training epochs, and η is the learning rate.

3.2.3. Federated learning on non-IID Data

Horizontal federated learning approaches, such as FedAvg algorithm exhibit satisfactory performance on Independent and Identically Distributed (IID) data. FL methods depend on stochastic gradient descent, which is widely employed for training deep learning models achieving good empirical results. The IID sampling of the training data is pivotal to guarantee that the stochastic gradient is an unbiased estimate of the full gradient. However, in real world settings and applications usually the local data of each client is not IID. Non-IID data heavily affect the performance of the horizontal FL models, since the local data distributions are different from the global data distribution, thus the averaged local model parameters may diverge from the global model parameters. In literature there is a plethora of studies that aim to address the challenges that non-IID data impose [1], [3], [4].

3.2.4. Personalization layer approaches

Different from the FedAvg approach, in this case each client is allowed to contain some personalized layers in the deep learning models focused on the local data distribution of the clients. Figure 9 illustrates an example of these networks, where each client model comprises of personalization layers (filled blocks) and base layers. It should be highlighted that only the base layers are uploaded to server for the global model aggregation, thus significantly reducing the communication costs, since only the base layers need to be uploaded to server.

FedPer algorithm [3] constitutes a characteristic case of these approaches. Particularly, in the FedPer model the base layers are shallow neural networks focused on capturing high-level representations of the local data and the personalization layers are deep neural networks aiming to tackle the classification problems. Algorithm 2 summarizes this method.

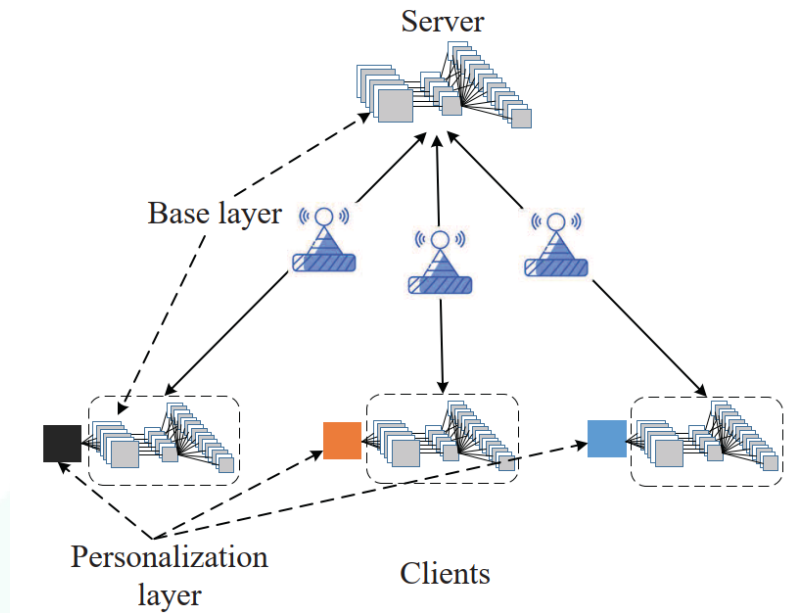


Figure 9. An illustration of horizontal FL with personalization layers. Only the base layers (filled blocks) are uploaded to server for the global model aggregation

```

1: Server:
2: Initialize the shared base model  $\theta_B^0$ 
3: Initialize personalization layers  $\theta_{P_k}^0$ 
4: for each communication round  $t = 1, 2, \dots, T$  do
5:   Select  $m = C \times K$  clients, where  $C \in (0, 1)$ 
6:   for each Client  $k = 1, 2, \dots, m$  in parallel do
7:     Download  $\theta_B^t$  to Client  $k$ 
8:     Do Client  $k$  update and receive  $\theta_B^k$ 
9:   end for
10:  Update base model  $\theta_B^t \leftarrow \sum_{k=1}^m \frac{n_k}{n} \theta_B^k$ 
11: end for
12:
13: Client  $k$  update:
14: Merge base model  $\theta_B$  and personalization layers  $\theta_{P_k}$ 
15: for local epoch from 1 to  $E$  do
16:   for batch  $b \in (1, B)$  do
17:      $(\theta_B^k, \theta_{P_k}^k) \leftarrow (\theta_B, \theta_{P_k}) - \eta \nabla L_k(\theta_B, \theta_{P_k}; b)$ 
18:   end for
19: end for
20: Return  $\theta_B^k$ 
    
```

Algorithm 2. FedPer

3.2.5. Clustering based approaches

Considering the highly non-IID distribution of the clients' data, instead of having only one global model, a client clustering method is proposed to develop a multi-center system by clustering the clients into different clusters. By creating multiple global models this FL framework is able to capture the heterogeneous data distribution of the clients. Nevertheless, note that the data distribution of each user is private. Thus, a clustering methodology is developed that performs the clustering of the users based on the similarity of the loss value, called iterative federated clustering algorithm (IFCA) [4], which is summarized on Algorithm 3. In more details, in this approach the server generates multiple global models and send all models to the participants. The participants train the cluster models based on their local data and compute the loss values of all models. Then, each clients updates the cluster model with the smallest loss and upload it to the server for cluster model aggregation.

```

1: Input: number of clusters  $k$ , any single cluster index  $j \in [k]$ , the total number of communication round  $T$ , number of local epochs  $E$ , mini-batch size  $B$ , learning rate  $\eta$ 
2:
3: for  $t = 0, 1, \dots, T - 1$  do
4:   Server: Broadcast cluster model  $\theta_j^t, j \in [k]$ 
5:   Randomly subsample  $m$  participating clients
6:   for client  $i \in m$  in parallel do
7:     Determine cluster group:  $\hat{j} = \underset{j \in [k]}{\operatorname{argmin}} F_i(\theta_j^t)$ 
8:     Generate one-hot vector  $s_i = \{s_{i,j}\}_{j=1}^k$  with  $s_{i,j} = 1 \{j = \hat{j}\}$ 
9:     option I (gradient averaging):
10:      Compute gradient:  $g_i = \nabla F_i(\theta_j^t)$ 
11:     option II (model averaging):
12:       $\tilde{\theta}_i = \text{ClientUpdate}(\theta_j^t, E, B, \eta)$ 
13:     Send back  $s_i$  and  $g_i$  or  $\tilde{\theta}_i$  to the server
14:   end for
15:   Server:
16:   option I (gradient averaging):  $\theta_j^{t+1} \leftarrow \theta_j^t - \frac{\eta}{m} \sum_{i \in [m]} s_{i,j} g_i$ 
17:   option II (model averaging):  $\theta_j^{t+1} \leftarrow \sum_{i \in [m]} s_{i,j} \tilde{\theta}_i / \sum_{i \in [m]} s_{i,j}$ 
18: end for
19: Return  $\theta_j^T, j \in [k]$ 
20:
21: ClientUpdate( $\theta_j, E, B, \eta$ ) at the  $i$ -th machine
22:  $\theta^i \leftarrow \theta_j$ 
23: for local epoch from 1 to  $E$  do
24:   for batch  $b \in [B]$  do
25:      $\theta^i \leftarrow \theta^i - \eta \nabla F_i(\theta^i, b)$ 
26:   end for
27: end for
28: Return  $\theta^i$ 

```

Algorithm 3. Iterative federated clustering algorithm (IFCA)

3.2.6. Federated learning for machinery fault diagnosis

Machinery fault diagnosis employing condition labelled data constitutes a pivotal tool in modern industries proving numerous benefits such as machine reliability, operation safety and low maintenance costs. Furthermore, several studies have pointed out the strong relation between the early fault diagnosis and the energy consumption [5]. Exploiting the ground-breaking progress of the recent deep learning models, data driven approaches has been widely employed by many industries to perform machinery anomaly detection. Although these data driven methods exhibit great performance accuracy, they require large amounts of high-quality supervised data to optimize an accurate diagnostic model. In real industrial scenarios labelled condition monitoring data are usually difficult and expensive to collect [6].

3.2.7. Problem Formulation

Taking into consideration that different companies and factories contain similar types of working machines, and they usually have their own supervised dataset for fault diagnosis a Federated learning

framework is employed to tackle the above-mentioned problem. Thus, the clients (different industries) can collaborate to develop a global fault diagnosis classifier in the central server, without sharing any sensitive information regarding their personal data. This study is conducted under the following assumptions:

Multiple clients participate in the FL system, and each client has limited training data to effectively develop its fault diagnosis model independently.

The fault diagnosis procedures of all the clients are the same, indicating that different participants share the same label space.

3.2.8. FedAvg and Network architecture

To perform the FL scenario the FedAvg algorithm is used, described analytically in Algorithm 1. At each training round, the fault diagnosis model is locally updated within each client, and then it is uploaded to the server for model aggregation. The architecture of the deep neural network that is shared by the server and the clients is illustrated in Figure 10. The model consists of the following modules. First, two 1-D convolutional layers with filter size of 9 and filter number of 10 are employed to perform the feature extraction process. After flattening, a fully connected layer with 64 neurons that is used to capture the more complex features of the data, and final a fully connected layer is adopted where each neuron stands for the classification confidence of each health condition.

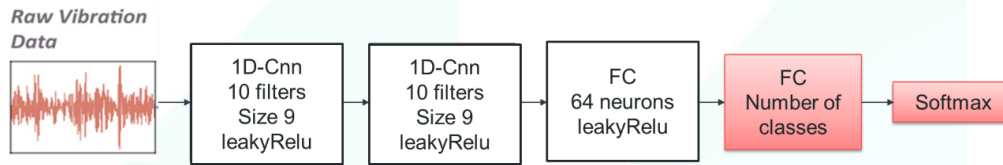


Figure 10. The proposed fault diagnosis deep learning model.

To train the local models each client employs the cross-entropy loss function

$$L_c = -\frac{1}{n_s} \sum_{i=1}^{n_s} \sum_{j=1}^{n_c} 1\{y_i = j\} \log \frac{e^{x_{i,j}^h}}{\sum_{k=1}^{n_c} e^{x_{i,k}^h}},$$

where n_s is the number of local data for the client, $\{x_i, y_i\}_{i=1}^{n_s}$ denotes the labelled samples and the n_c stands for the number of classification classes.

3.2.9. Experimental Setup

Dataset¹: The Case Western Reserve University (CWRU) rolling bearing dataset contains vibration acceleration signals collected from the drive end of the motor and the sampling frequency is 12 KHz. 4 machinery health states are examined, i.e., healthy (H), outer race fault (OF), inner race fault (IF) and ball fault (BF). The corresponding fault diameters are 7, 14, 21 mils. Thus, we have one healthy and three fault modes were classified into ten categories (one health state and 9 fault states) according to different fault sizes. Figure 11 summarizes the under examined health conditions.

Dataset	Descriptions										
	Condition label	1	2	3	4	5	6	7	8	9	10
CWRU	Fault location	N/A (H)	IF	IF	IF	BF	BF	BF	OF	OF	OF
	Fault size (mil)	0	7	14	21	7	14	21	7	14	21

Figure 11. Description of the CWRU dataset.

¹ <https://engineering.case.edu/bearingdatacenter>

Training Parameters: The number of training and testing labelled data was 17987 with dimension, $d=500$. Furthermore, the batch size was set to 16, the number of training epochs for updating the local models was 30, while the number of communication rounds between the server and clients was set to 50. Finally, the Adam optimizer was employed to train the proposed models and the leakyRelu was used as activation function.

3.2.10. Results

To quantify the performance of the under-examined federated learning scenario, extensive numerical experiments were conducted in the context of machinery fault diagnosis. In more details, two scenarios were examined where the first corresponds to the IID case and the second focuses on the non-IID case.

IID Scenario

In this scenario the local data of the clients are independent and identically distributed (IID), thus each client contains labelled data from all the under-examined health conditions (10 in total). Particularly, the performance of the FedAvg algorithm is examined with 5 and 10 clients, where the number of training data per class was limited into the following range {100, 200, 300, 400, 600, 1000}. Figure 12 demonstrates the classification accuracy of the FedAvg scheme. It is obvious that the case with the 5 clients exhibits better performance compared to the scenario with the 10 clients, especially when fewer training data is used.

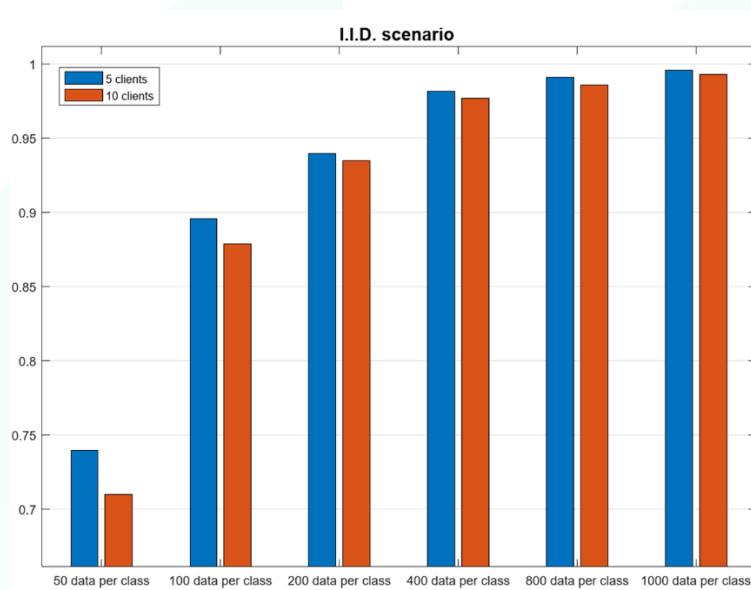


Figure 12. The proposed fault diagnosis deep learning model.

Non-IID Scenario

In this section a more realistic situation is considered, where the local labelled data of different clients are not IID. To this end two non-IID scenarios were examined

- **Scenario 1:** with 10 faulty bearing conditions in total, 9 clients are considered where each clients contains data only from one faulty condition and some data from the healthy condition (category 1).

- **Scenario 2:** 9 clients were considered where each client has data of 3 conditions chosen from conditions [2, 3, 4] or [5, 6, 7] or [8, 9, 10] as described in Figure 11. Moreover, all clients have data from the healthy state.

Table 1. Accuracy per non-IID scenario

Scenarios for non-I.I.D problem	Accuracy
1	0.401
2	0.536

It is evident that the proposed federated learning approach although exhibits very good results (almost 100% accuracy) for the IID. scenario, in the case of the non-IID. scenario the performance drops dramatically. Thus, better, and more suitable approaches for the non-IID. case should be explored. In future work, we aim to implement FL for non-IID data methodologies, such as algorithm [3], [4] and compare their performance with the FedAvg method.

3.3. Indoor Industrial space Mean Radiant Temperature distribution (MRT) estimation using infrared thermography images

The Mean Radiant Temperature (MRT) at a specific point in an indoor space is defined as the uniform temperature of an imaginary enclosure in which radiant heat transfer from the examined point (e.g., object) equals the radiant heat transfer in the actual nonuniform enclosure [1]. Due to the fact that it is difficult to measure MRT values directly, various non-intrusive instruments (such as GTs (globe thermometer) and IR cameras) are employed to derive approximate MRT values [2]. In more detail, the MRT at a point in indoor space can be determined from the temperatures of the surfaces that enclose the point, as shown in equation (1).

$$T_{MRT}^4 = \varepsilon_1 T_{s_1}^4 F_{p-1} + \varepsilon_2 T_{s_2}^4 F_{p-2} + \dots + \varepsilon_n T_{s_n}^4 F_{p-n} \quad (1)$$

where T_{s_i} stands for the temperature of the surface i , F_{p-i} corresponds to the view factor from the target point to surface i and ε_i is the emissivity of surface i , for $i = 1 \dots n$. The view factor is defined as the percentage of radiant energy emitted from on surface to another surface, which reflects the geometric shape and positional relationship between different objects. Note that when wall surfaces have high emittance ε_i can be assumed to be equal to 1 [3], [4].

3.3.1. Different Scenarios examined in literature

In literature numerous studies have explored the potentials of analyzing the MRT distribution under different scenarios and settings. The MRT values can provide valuable information regarding the temperature distribution in both small and large indoor [3], [5] and outdoor spaces [4]. In [3], the authors examined the MRT distribution in a small indoor space with dimensions 5 m × 3 m × 2.5 m. Furthermore, in [5] they measure a wide range of MRT values to understand thermal comfort conditions for a large indoor space. In particular, they conducted experiments in a dome stadium with indoor dimensions 68 m × 160 m × 218 m (height × width × length). However, the MRT can be also employed for large outdoor spaces. For instance, in [4], they estimate the MRT distribution in an outdoor environment located in the teaching buildings district of Guangxi University (see Figure 13).

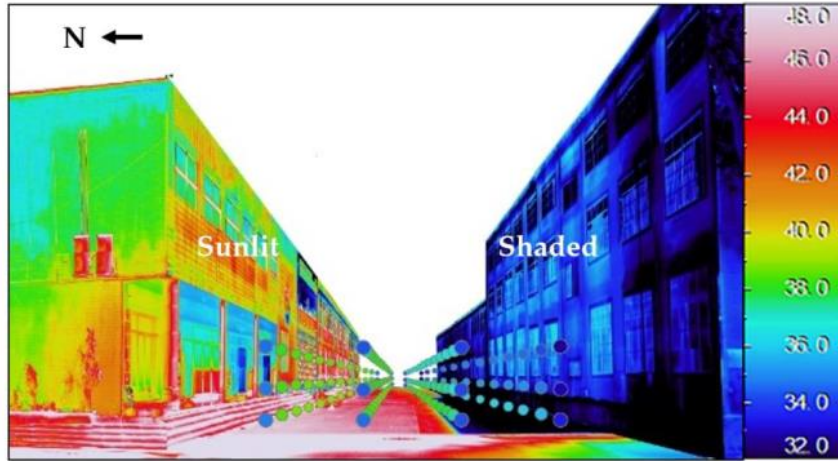


Figure 13. MRT distribution maps in a pedestrian space [4]

3.3.2. Mean Radiant Temperature Calculation using the IR camera

The methodology to estimate the MRT distribution of an indoor space consists of two major stages: (1) calculating the view factors in every point of the room where MRT is to be calculated and (2) measuring the surface temperatures with the IR camera.

3.3.3. Calculation of View Factors

As mentioned before, the view factor represents the fraction of the total radiation emanating from a surface in all possible hemispheric directions across all possible wavelengths, as received by another object of surface [3]. Deriving the view factors requires modeling the indoor space. In particular, the view factor for a very small surface area on the wall, say A_i , from an arbitrary point in space (x, y, z) can be estimated as

$$F_{p-i} = \frac{A'_i}{4\pi} \quad (2)$$

where A'_i is the projection of the surface area A_i to the sphere of radius $r=1$, as shown in Figure 14.

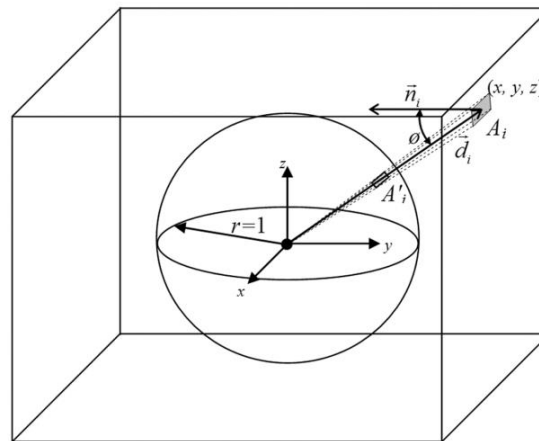


Figure 14. View factors calculation

After that the projection area A'_i is calculated as follows

$$A'_i = \frac{A_i \cos \varphi}{x^2 + y^2 + z^2} = A_i \frac{t}{(x^2 + y^2 + z^2)^{\frac{2}{3}}} \quad (3)$$

where t is the distance from the originating point to the wall. Note that the wall surface area A_i should be adequately small [3].

Having for each surface A_i the corresponding temperature provided from the IR camera, the MRT distribution of the indoor space can be estimated by computing for several points in the indoor space the view factors according to equation (2).

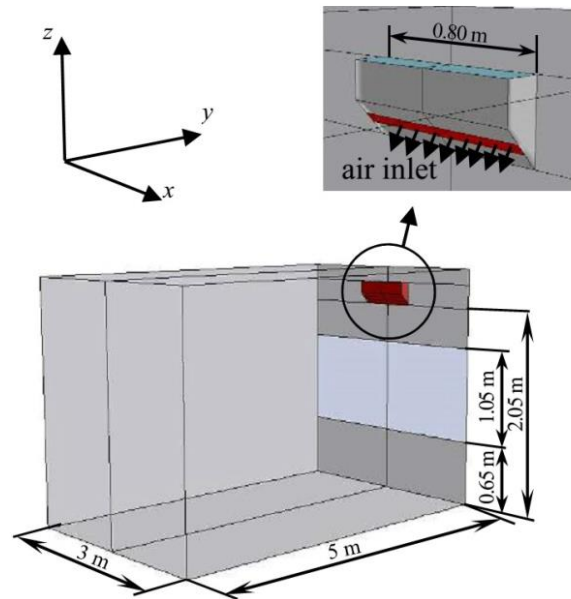


Figure 15. Modelled enclosed room [3]

3.3.4. Case study

Using 3D models of indoor spaces (e.g., Figure 15) and thermal images to estimate the surface temperatures, our goal is to compute based on equation (1) the MRT distribution of industrial environments by implementing small case trials according to the premises of the ISI. Based on the 3D models of the under-examined space and the temperatures of the indoor surfaces provided by the IR camera, our target is to derive detailed maps concerning the MRT distribution. Figure 16 provides an example of these maps. Note that the surfaces can be divided into smaller surface areas A_i (such as 50cm x 50cm, 25cm x 25cm, 10cm x 10cm, 5cm x 5cm) according to the temperature distribution on the wall surfaces, thus providing more accurate results.

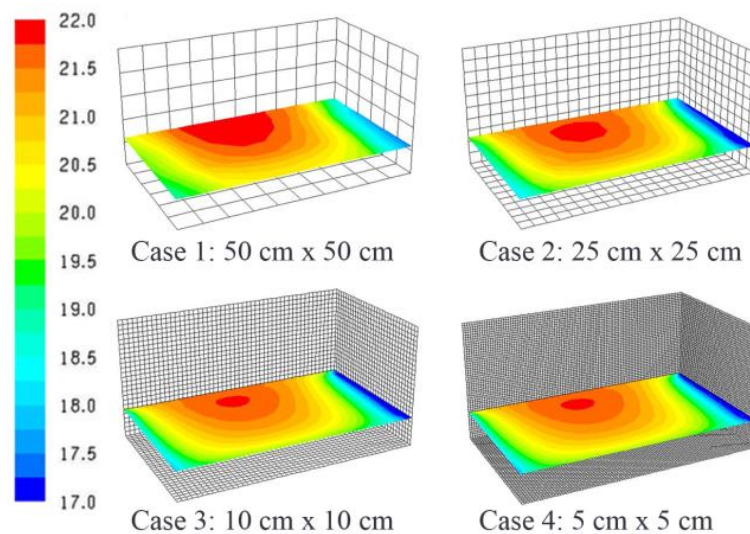


Figure 16. MRT distribution maps [3]. The wall surfaces are divided into smaller surfaces with different sizes to determine the optimal value regarding the accuracy of the MRT calculation.

4. EDGE NODE SECURITY ASPECTS

Traditionally, the Industrial Control Systems (ICSs) that are employed to control an industrial process, often referred to as Supervisory Control and Data Acquisition (SCADA) systems, are based on primitive implementations lacking cyber-security considerations and practices. Reasons for this can be found in the lack of interoperability between different vendors and/or the adoption of proprietary protocols and data formats. Ensuring interoperability between platforms and devices has two major challenges, i.e., their seamless operation and their security. The weakest link in this chain, from a cyber-security perspective, are the endpoints on SCADA systems, i.e., the Programmable Logic Controllers (PLCs) with their sensors and actuators. Not only is their firmware full of flaws with no regular update policy, but also, many of the most popularly used communication protocols lack authentication or encryption [7].

In legacy industrial deployments, the isolation of the SCADA deployments had been a viable option, however, in today's interconnected and technologically mobile world, true isolation is nearly impossible. It is, therefore, crucial that in EnerMan we tackle edge node security aspects efficiently since we are planning to collect data from the use case sites by interconnecting the edge nodes to the targeted ICSs.

4.1. Attack Threat Model

Industry 4.0 and Smart manufacturing companies are subjectable to threats and attacks, which can affect their production, infrastructure, personnel, and their operations in general. These threats and attacks should be considered during the development of EnerMan and should be addressed during design and implementation. Even if the manufacturers are not aware of them or think of them as a second-class priority, these threats can disrupt their operations to such an extent, which could create huge loss of revenue or even leave a whole country out of resources. Characteristic examples are the disruption of the largest petroleum pipeline of the east coast by ransomware, and a major electricity supplier in Johannesburg which was hit again by a ransomware, which resulted in leaving several citizens without electricity.

ENISA developed guidelines about the security in Industry 4.0 and provided security measures, which should be implemented for a factory to be considered secure. They provide a threat taxonomy, which suits the needs of EnerMan, and we argue that EnerMan should address, at minimum, those threats, and measures against them. There are several categories and threats which should be addressed. Below we describe the most important ones, which should be considered during the implementation of the project.

4.1.1. ICS threats

Denial-of-Service (DoS) is an attack meant to disrupt the availability of a machine or a network by making it inaccessible to the legitimate users. DoS attacks flood the target with traffic which is intended to deplete the resources of a machine or a network, so it cannot process any legitimate traffic and packets. In industry 4.0 a DoS attack can target Industrial Internet of Things (IIoT) systems resulting in system unavailability and production shut down. Attackers could also take advantage of many IIoT devices in industrial environments and create a botnet which can later be used to conduct attacks on other infrastructures.

Malware is a piece of malicious software, normally a file or code, which is normally delivered through the network, that infects, steals, monitors or conducts any function that an attacker desires. Usually, an attacker uses a remote-control center to control the malware and it can later send commands to perform unwanted actions. These actions may cause damage to an OT system, operational processes, and related data. Ransomwares, viruses, trojan horses, and spyware are common examples of this

threat. Malwares can affect critical systems such as servers, IIoT devices, mobile devices, cloud computing services etc.

Attackers may also try to modify unwanted and unauthorized data. By compromising OT or production systems such as SCADA, they could alter and tamper data that play an important role in the decision-making of the industry, which could result in inappropriate decisions based on false information. In the same context is the threat of compromised personal and sensitive information. Attackers may try to gain unauthorized access to personal devices or in the company’s cloud server to collect data about the employees and their roles, their performance, their names, salaries etc. This is a privacy leakage issue which could expose personnel information and hurt the trust of the company.

An attacker may try to gain access to an organization by using a brute force attack. The attacker submits many passwords and passphrases with the goal of eventually guessing the correct one, so she can log into the systems of the organization. Brute force is a common attack, which requires low to nontechnical expertise to be conducted. It still poses a big problem among organizations especially the ones that allow the utilization of uncomplicated or default passwords for industrial devices and systems.

A man in the middle attack (MITM) is an attack where the perpetrator positions himself in the middle of a channel between a user and an application in order to eavesdrop or to impersonate one of the two parties. The attacker can just listen to exchanged messages in order to steal company’s sensitive information such as passwords, credentials, confidential files, or can even modify and delete messages to disrupt communication. This threat could have serious implications for the company since data could be leaked, resulting in system compromise, or the operations of the organization could be disrupted by false messages transmitted by the attacker. In a similar context, another threat is the communication protocol hijacking. In this attack the attacker takes control of an existing communication session between two network components. This could lead to sensitive data leakage, password leakage or other confidential information compromise.

Network reconnaissance is the first part of any hacking operation. Attackers try to learn the target environment that can help in the identification of potential attack vectors and exploits on potential vulnerabilities. Reconnaissance efforts can be either passive or active. In passive reconnaissance the attacker listens to the network and services without taking any actions in order to remain undetected. On active reconnaissance, the attacker will actively send packets and communicate with devices and resources, either by scanning or by connecting to them in order to collect as much information as possible. In industry 4.0, such an attack could reveal internal network information like connected devices, network protocols, open ports etc., which the attacker could later utilize to orchestrate an attack to the organization.

4.1.2. Threat Agents

Threats sources for ICSs are located in various groups, including adversarial sources, such as foreign governments, terrorists, competition spies, dissatisfied employees, malicious intruders, as well as system errors and malfunctions, equipment errors or unintended events e.g., accidents and human errors. Table 2 provides an overview of them.

Table 2: Adversarial Threats to ICS [8]

Threat Agent	Description
Attackers	Attackers break into networks for the thrill of the challenge or for bragging rights in the attacker community. While remote cracking once required a fair amount of skill or computer knowledge, attackers can now download attack

	<p>scripts and protocols from the Internet and launch them against victim sites. Thus, while attack tools have become more sophisticated, they have also become easier to use. Many attackers do not have the requisite expertise to threaten difficult targets such as critical U.S. networks. Nevertheless, the worldwide population of attackers poses a relatively high threat of an isolated or brief disruption causing serious damage.</p>
Bot-network operators	<p>Bot-network operators are attackers; however, instead of breaking into systems for the challenge or bragging rights, they take over multiple systems to coordinate attacks and to distribute phishing schemes, spam, and malware attacks. The services of compromised systems and networks are sometimes made available on underground markets (e.g., purchasing a denial-of-service attack or the use of servers to relay spam or phishing attacks).</p>
Criminal groups	<p>Criminal groups seek to attack systems for monetary gain. Specifically, organized crime groups are using spam, phishing, and spyware/malware to commit identity theft and online fraud. International corporate spies and organized crime organizations also pose a threat. through their ability to conduct industrial espionage and large-scale monetary theft and to hire or develop attacker talent. Some criminal groups may try to extort money from an organization by threatening a cyber attack</p>
Foreign intelligence services	<p>Foreign intelligence services use cyber tools as part of their information gathering and espionage activities. In addition, several nations are aggressively working to develop information warfare doctrines, programs, and capabilities.</p>
Insiders	<p>The disgruntled insider is a principal source of computer crime. Insiders may not need a great deal of knowledge about computer intrusions because their knowledge of a target system often allows them to gain unrestricted access to cause damage to the system or to steal system data. The insider threat also includes outsourcing vendors as well as employees who accidentally introduce malware into systems. Insiders may be employees, contractors, or business partners. Inadequate policies, procedures, and testing can, and have led to ICS impacts. Impacts have ranged from trivial to significant damage to the ICS and field devices. Unintentional impacts from insiders are some of the highest probability occurrences.</p>
Phishers	<p>Phishers are individuals or small groups that execute phishing schemes in an attempt to steal identities or information for monetary gain. Phishers may also use spam and spyware/malware to accomplish their objectives.</p>
Spammers	<p>Spammers are individuals or organizations that distribute unsolicited e-mail with hidden or false information to sell products, conduct phishing schemes, distribute spyware/malware, or attack organizations (e.g., DoS).</p>
Spyware/malware authors	<p>Individuals or organizations with malicious intent carry out attacks against users by producing and distributing spyware and malware.</p>
Terrorists	<p>Terrorists seek to destroy, incapacitate, or exploit critical infrastructures to threaten national security, cause mass casualties, weaken the economy, and damage public morale and confidence. Terrorists may use phishing schemes or</p>

	spyware/malware to generate funds or gather sensitive information. Terrorists may attack one target to divert attention or resources from other targets.
Industrial spies	Industrial espionage seeks to acquire intellectual property and know-how by clandestine methods

4.1.3. Security Requirements

To counter the numerous threats that can target Industry 4.0 organizations as those that are considered in the EnerMan project, security mechanisms have to be implemented in order to secure their assets. Smart manufacturing companies should always keep an eye on the level of security and practices they have in place, in order to minimize the threat level and ensure their continuous operation. Below we describe in high level what are the minimum mechanism that we believe industry 4.0 should have in place in order to ensure its secure operations.

Security mechanisms need to be used to ensure the integrity and trust of the data and the devices. Software needs to be verified before start running and ensure that is signed by the actual vendor and not tampered. IIoT devices need to be authorized to run in the network and secure channels needs to be utilized to ensure the integrity of the data and the connections. Cryptographic mechanisms should be used to ensure the data integrity and security. Also, all the production data need to be monitored either they are at rest or on transit to identify potential modifications.

Cloud infrastructure and data stored in the cloud should also be secured in various aspects. Types of cloud utilized by the organization should take into consideration the laws and regulations of the cloud provider's country and points of presence. Single points of failure should be avoided, and critical systems and applications should be identified, so a correct risk assessment can be made.

About the business continuity and recovery, the critical systems and processes should be identified to determine the extent which they can influence the production. Risk assessments should be performed, and procedures should be in place to define a plan in case of a security incident.

Communications between different machines should be secured either they operate inside the factory or if they connect to the internet. Secure cryptographic algorithms should be used to provide authentication, integrity and confidentiality between machines and private keys should be stored securely in a server. Messages should be checked to ensure that they are not tampered or that they are not replayed, and input validation should be in place to ensure that injection commands and cross-site scripting (XSS) is avoided.

Data should be encrypted either they are on transit or at rest and should be categorized based on their criticality. This should be the result of a risk analysis and risk assessment. Encryption and key management should be utilized to ensure that not all users have access to the data and anonymization of personal data should be in place whenever possible, to minimize security implications in case of a security breach.

Access control policies should also be applied to ensure authentication of users and accounts, remote access, and user privileges. Minimal level of authentication should be used across all different IIoT devices. Multi-factor authentication should be used, and default password and usernames should be avoided. The least privilege principle should be applied, and roles should be properly assigned to each person. System users should have different accounts with different privileges, and access control systems and Privilege Access Management solutions should be in place.

Correct security measures should be applied in the network and the protocols used in the factory. Industrial plants networks should be based on pre-defined zoning model with establishment of DMZ zones and control of traffic between zones. IIoT solutions should implement proven-in-use protocols

with known security capabilities, based on broadly accepted standards. Secure environments should be secured for key exchange and key management and proper use of cryptography should be applied to protect confidentiality, integrity and availability of data and information. The security should be strong and insecure protocols should be avoided.

There also should be a monitoring system across the organization to detect anomalies and real-time attacks. Security logs should be collected and analyzed and periodic reviews of network logs, access control privileges and assets configurations should be conducted

4.1.4. Security Architecture

A high-level representation of the EnerMan security architecture is shown in Figure 17. Overall, its purpose is twofold. On the one hand, it is aimed at preventing malicious activities from becoming successful, i.e., it aims at reinforcing the flow data such that they become immune to infection by malicious activity. The second involves the aspect of detection and, in this particular case, what we have is a mechanism for picking out unwanted activity that has managed to become part of the data flow.

The architecture consists of several levels, each of which correspond to a different part of a typical system that is going to use the EnerMan framework. Hence, there is the Industrial Data, which originate from the fringes of the architecture, e.g., sensor modules as edge devices in a factory, the Secure Gateway that is a little further up the hierarchy, i.e., the edge node of the system, and, finally, the cloud server.

At each of these points, as well as in-between, EnerMan is going to implement security features that will setup a strong security mechanism. Hence, starting from the edge devices, EnerMan utilizes an intrusion detection mechanism named I2DS. This is implemented on MPSoC technology using the EnerMan edge/end node execution environment and is positioned right at the entry to the Data Aggregator, also co-hosted at the MPSoC. The I2DS operates on the data that are flowing in from the various edge devices used in the context of the various EnerMan use case providers' industry setups. Having filtered the data and flagged any potentially malicious activity, the data is then processed inside the MPSoC by the data aggregator and, subsequently, it is encrypted for cyber-attack prevention purposes.

The encrypted data are going to fulfil TLS secure session communication protocol requirements, which will assist in the consolidation of a prevention mechanism between the MPSoC (edge devices) and Gateway (edge node) layers of the architecture. Just prior to the introduction of the MPSoC data into the gateway, a second Intrusion Detection System (IDS) mechanism is deployed. Hence, a detection mechanism just prior to the EnerMan gateway ensures that the encrypted data have indeed not been corrupted. Subsequently, an Intrusion Prevention System (IPS) mechanism follows on the IDS-processed data at the gateway-level of the architecture.

Hence, and similar to the security steps followed at the EnerMan edge/end node MPSoC, the EnerMan gateway(s) will encrypt the data that are to be propagated further up the architecture, i.e., the EnerMan cloud devices, by ensuring that the TLS protocol standards are met for prevention purposes in the context of edge node and cloud communication.

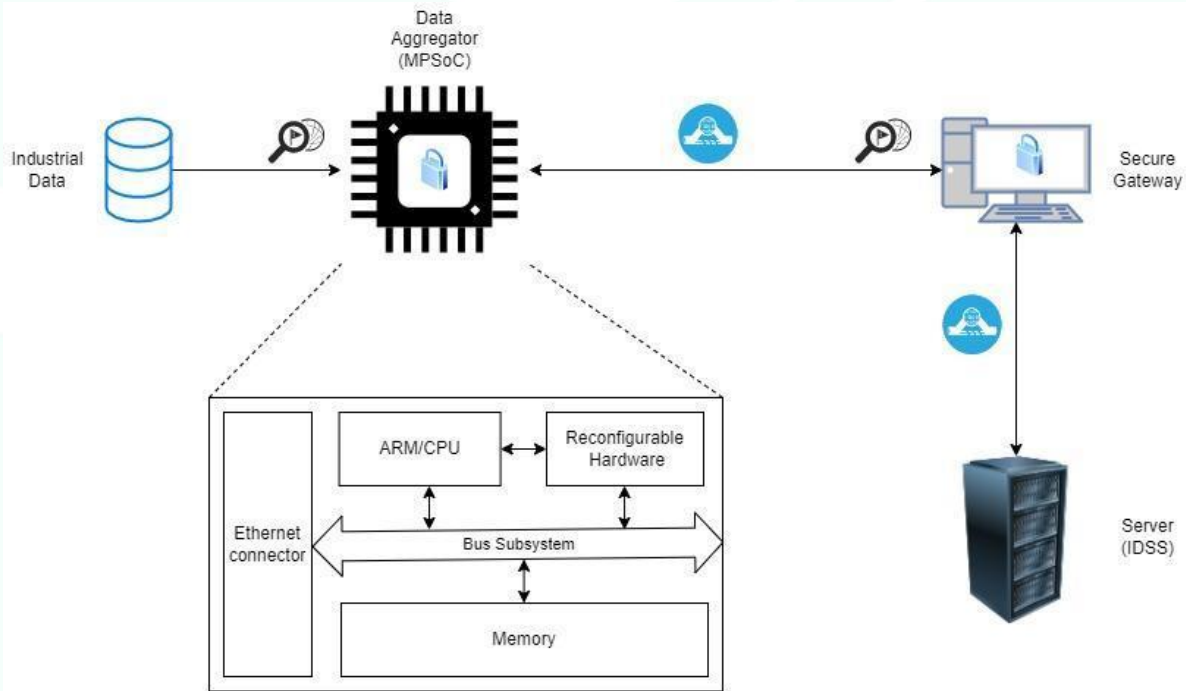


Figure 17. The EnerMan Security Architecture

4.2. Security Mechanisms

4.2.1. Cybersecurity Attack Detection

I2DS: Industrial Intrusion Detection System

I2DS is the EnerMan intrusion detection mechanism that will be deployed across the system and at its very edges, i.e., it is the intrusion detection mechanism that will operate on the data coming in from the architecture's edge devices. I2DS is going to be hosted by the EnerMan edge/end node MPSoC devices, which are devices that employ FPGA technology. Hence, this first layer of intrusion detection capability is going to be implemented directly on (reconfigurable) hardware using the EnerMan edge/end node execution environment described in Section 2.

Specifically, I2DS consists of optimized modules that implement machine learning models for intrusion detection. The modules use rules for string searching that are appropriate for the industrial environments' data. The implementation of the ML model's architecture is to be developed using suitable frameworks so that the final design not only fulfills the functional criteria of the ML model, but it also offers satisfactory performance, such as a high data throughput at the cost of reduced power consumption requirements.

IDS: Intrusion Detection/Prevention Systems

Intrusion detection systems (IDS) are systems that monitor network traffic for suspicious activity and alerts when such activity is discovered whereas Intrusion Prevention Systems (IPS) have a more extended operation by including prevention function when an intrusion is identified. Both IDS and IPS system exploit mechanisms for identify anomalies in the network traffic varying from pattern matching to advanced machine learning and deep learning techniques aiming to increase their detection accuracy. Machine learning methods and Deep Learning methods can automatically discover the essential differences between normal data and abnormal data with high accuracy. In addition, machine learning methods have strong generalizability, so they are also able to detect unknown attacks. Intrusion detection systems that can be trained using a baseline (i.e., normal system behavior), to identify anomalous events (e.g., behaviors differing from the baseline). This type of

anomaly IDS can also monitor network traffic and be trained to recognize malicious streams of packets based on known attack streams.

For the communication between EnerMan's edge/end node (acting as data aggregator) and the rest of the framework we consider deploying a lightweight IPS solution to provide a detection mechanism augmenting the total security posture of the offered solution.

SNORT² is a network-based intrusion detection system that is an open-source software than can be used as an IDS/IPS, as well as a real time network sniffer. Its basic usage can be as follows:

- **Sniffer Mode:** to collect and printout TCP/IP network header information.
- **Packet Logging:** to store network traffic packets that can be further analyzed later on (e.g., for forensic investigation).
- **Active Network intrusion detection mode:** to create alerts, when possible, intrusions are detected.

The intrusion detection is based upon specific rules that have to be setup in snort deployment to match the used case needs. Each of those rules consists of two logical parts:

- **Rule header:** contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information.
- **Rule options:** contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

SNORT has the ability to detect probes or attacks, including, but not limited to, operating system fingerprinting attempts, semantic URL attacks, buffer overflows, server message block probes, and stealth port scans. SNORT rules use signatures to define attacks. These signatures are specifically designed to detect known exploits as they contain distinctive marks, such as ego strings, fixed offsets, debugging information, or any other unique marking that may or may not be related to actually exploiting a vulnerability. Snort will receive packets and process them through preprocessor and compare these packets against the set of rules. The output will log or trigger alerts based on what action the rules will take. An abstracted architecture of SNORT operation is given in Figure 18.

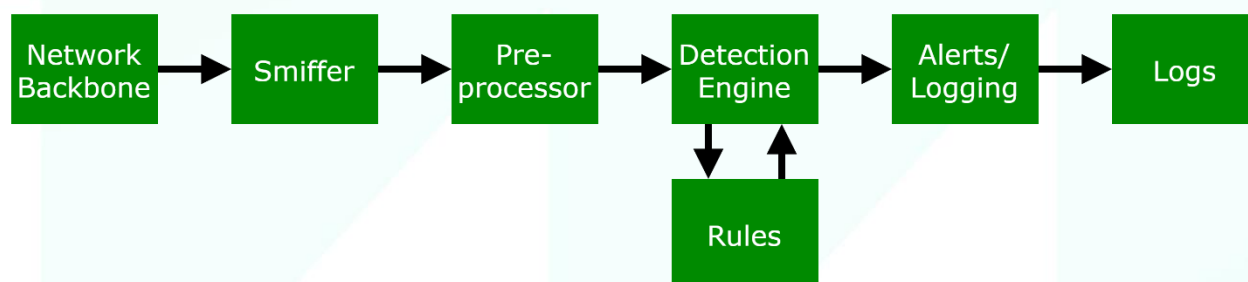


Figure 18. Logical representation of SNORT architecture flow

4.2.2. Cybersecurity Attack Prevention

Secure Gateway

The purpose of the secure gateway is to provide authorization and authentication mechanism to the data aggregator in order to be able to send the aggregated data to the EnerMan framework. A way to

² <https://www.snort.org/>

do this is to add to REST services and request to REST APIs an authentication and authorization layer. The API authentication can be performed in three ways:

1. Based on user credentials
2. Based on tokens that are created by OAuth common flows.
3. Based in certificates

The first method is less secure where the third one required the creation and maintenance on certificate by a trusted certificate authority. Thus, for the EnerMan data aggregator need the second one seems to be more appropriate.

OAuth 2.0

OAuth 2.0³ is a widely adopted and the industry standard protocol for authorization. Its specification and its extensions are being developed and maintained by the IETF OAuth Working Group. OAuth 2.0 aims to provide a simple solution for client development while providing specific authorization flows for client applications such as web applications, desktop applications, mobile phones, wearables etc. and living room devices. It can enable a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. It should be noted that OAuth is able to provide authorization both in human to machine and machine to machine communications.

OAuth2.0 specification describes four different roles:

1. Resource owner: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
2. Resource server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
3. Client: An application making protected resource requests on behalf of the resource owner and with its authorization.
4. Authorization server: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

The OAuth specification work defines a variety of grant types for different use cases such as Authorization Code, PKCE, Client Credentials, Device Code, Refresh Token

³ <https://oauth.net/2/>

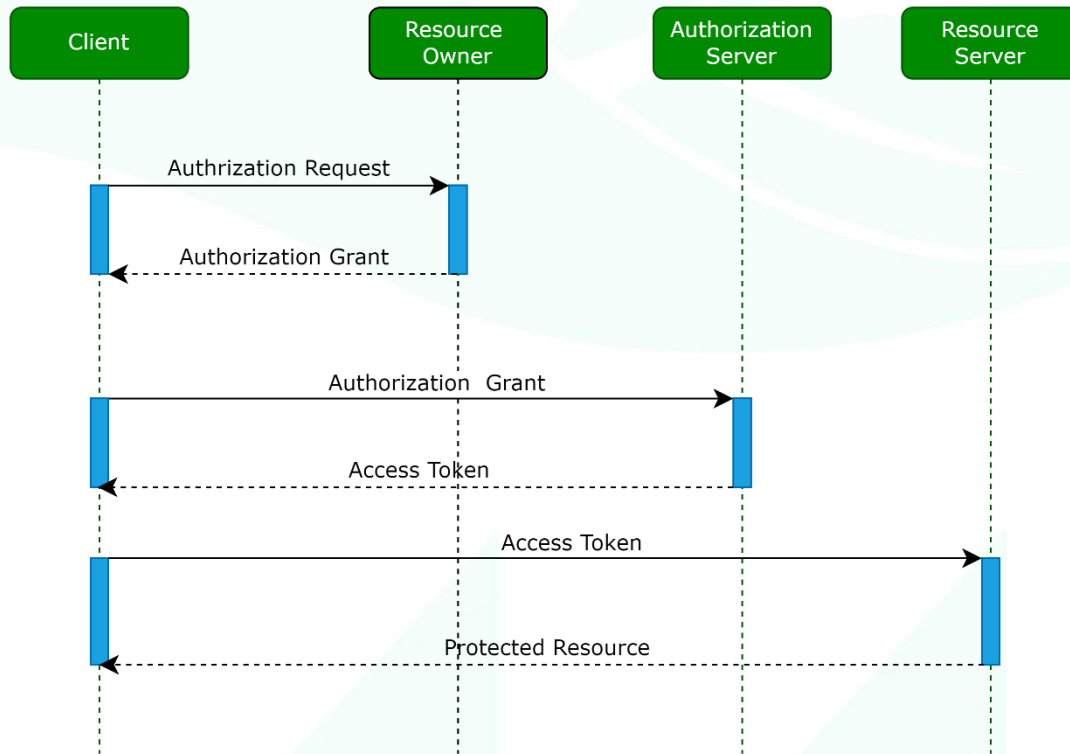


Figure 19. OAuth authentication and authorization flow

An OAuth2 flow is drafted in Figure 19. This flow describes the interaction between the four different defined roles in the specification:

- The client requests authorization from the resource owner.
- The client receives an authorization grant, which is a credential representing the resource owner's authorization, expressed using one of the four authorization grant types.
- The client requests an access token by authenticating with the authorization server and presenting the authorization grant.
- The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.
- The client requests the protected resource from the resource server and authenticates by presenting the access token.
- The resource server validates the access token, and if valid, serves the request

OpenID Connect

OpenID Connect⁴ is an interoperable authentication protocol based on the OAuth 2.0 family of specifications. This protocol provides to clients a tool to verify their or the End-User's identity based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner. It uses straightforward REST/JSON message flows. It allows developers to authenticate their users across websites and apps without having to own and manage password files. Also, it allows for clients of all types, including browser-based JavaScript and native mobile apps, to launch sign-in flows and receive verifiable

⁴ <https://openid.net/connect/>

assertions about the identity of signed-in users. OpenID Connect uses the ID token data structure that enable end-users or applications to be authenticated. The ID token has a JSON Web Token (JWT) format, which is a standard way to generate authentication tokens. The JWT contains various user information which are called claims. Also, it contains information about the validity of the token, such as issue datetime, expiry period etc. The token is normally signed by the token issuer with the issuer's public key to be easily verified using Public Key Infrastructure (PKI).

Encryption

The backbone of any security mechanism that prevents cybersecurity attacks as those are described in the beginning of section 4 is to enforce cryptography operations on the data that are stored or are in transit. Given that several cryptographic operations are computationally intensive and resource hungry (especially the Public Key cryptography operations), mechanisms to efficiently implement such operations are needed. For this reason, the full functionality of the EnerMan edge/end node execution environment is used that involves hardware reconfigurability. This is manifested by the implementation of specific cryptography/security operations as hardware IP cores in the FPGA fabric of the node MPSoC.

The cyberattack prevention security mechanisms that are deployed on the EnerMan edge/end node (acting as data aggregator) are realized mostly using a hardware security token (HST) that is implemented on top of the EnerMan edge/end node execution environment. The overall structure of the HST can be seen in Figure 20. The HST is able to provide a series of security primitive services that include cryptography key generation for symmetric key and asymmetric key cryptography algorithms (like AES, CHACHA in various modes and Elliptic Curve cryptography) but also for quantum safe cryptography algorithms that will guarantee a high level of security even against quantum computer based cyberattacks. For the above algorithms the HST provides encryption and decryption capabilities as well as generation, signing and verification of digital signatures. Furthermore, the HST is able to securely store in special, protected storage structures sensitive security keys and also offer secure storage of pilot data. Apart from that the functionality of the HST can be linked with the detection mechanisms described in the previous subsections where the I2DS tool can act as a security sensor/agent for possible attacks on the HST itself or the pilot data collected by the EnerMan edge/end node. The HST also has an event logging mechanism that is capable of reporting to other security modules (e.g., the security gateway) security events that have been detected by the security sensors/agents.

Finally, given the cryptography capabilities of the HST, the token can offer highly secure quantum safe TLS 1.3 functionality that can secure the transmission of data leaving the EnerMan edge/end node. This can be applicable in all the applications (the EnerMan software agents) that are currently being realized in the edge/end node including the federated learning realization (securely transmitting client local models to the federated learning server).

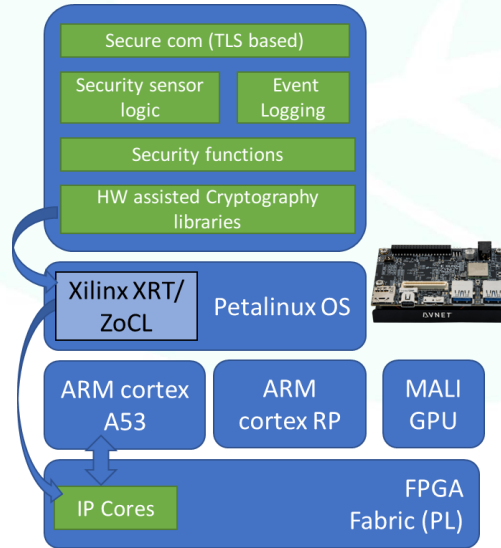


Figure 20 Hardware Security Token edge-node cryptography based cyberattack prevention setup

5. DEMONSTRATION REPORT

5.1. Execution Environment Hardware accelerated application Demonstration

To create and operate the hardware assisted EnerMan edge node execution environment on the Xilinx MPSoC edge board with an integrated FPGA fabric, we are using the Xilinx toolbox for designing and implementing functionality for the Processor System (PS) and the FPGA Programmable Logic (PL) of the Xilinx MPSoC. More specifically we are using the Xilinx Vitis toolbox (version 2021.1 and 2021.2).

The developed EnerMan execution environment on the two used embedded system board (Xilinx ZCU104 and ULTRA96 boards) supports the PetaLinux OS and the ZoCL/Xilinx RunTime (XRT) drivers for deploying hardware accelerators in the PL but also the PYNQ python library (developed by Xilinx) that enables to deploy on the PL through python scripts (using the concept of PYNQ overlays).

Initially, in this demonstration report, we show in detail the overall workflow that we used to create the relevant EnerMan execution environment to support the above functionality. After that, we show how the EnerMan execution environment can be used to deploy a custom hardware accelerated simple application (a matrix multiplication) through python using the PYNQ overlay capabilities of the platform Xilinx ZCU104.

5.1.1. Platform Creation with Linux system

The current flow consists of creating a platform project using Xilinx's Vivado tool⁵, a tool that creates customized hardware to be deployed in the FPGA fabric of the Xilinx MPSoC chip, that contains all the hardware information required by the Vitis HLS tool. The next step is to setup the OS that will run on the specific embedded system based on the hardware information, in our case the PetaLinux OS. Finally, we import both inputs from Vivado and the OS setup into VITIS and develop our application and build the final solution.

The output from VITIS contains the boot components, the .xclbin file containing the IP core and application executable that runs on the device's CPU. We can then write these on an SD-card, boot up the embedded device and run the application, or if the SD-card is already setup, simply deploy the .xclbin file and the application executable on the device and run the application.

What follows is a general flow to create a VITIS application as discussed previously. Appendix I provides a more detailed tutorial of the flow.

Vivado project creation and. XSA export

As discussed, our first step to build our platform for our application is to create a project in the Xilinx Vivado tool with our underlying hardware, connecting our peripherals and other basic components (e.g., AXI buses, interrupt controllers, platform interface etc.).

We are using a Xilinx ZCU104 design from Xilinx for our hardware description⁶ and export the corresponding .XSA file which contains all needed information for our board, hardware, and

⁵ <https://www.xilinx.com/products/design-tools/vivado.html>

⁶ https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104

its internal connections. Xilinx provides specific automated TCL scripts that creates⁷ the design and export⁸ the generated XSA file.

The output of Vivado that will be used in the following steps is the generated XSA file.

PetaLinux project creation

The next step is to setup the appropriate PetaLinux OS for the hardware design that we have generated. A PetaLinux-project can be created using the Xilinx tools.

For this project we aim to create a Linux system which will contain also the PYNQ-overlay and all other components that are needed for a hardware accelerated application to run.

At this stage, we must create a PetaLinux project and import the .XSA file generated by Vivado and designate the target device. After that we have to modify the root file system of the PetaLinux source files with all the necessary packages for XRT, PYNQ and any other required libraries. The last actions are the modifications of the kernel, updating the device tree based on the .XSA file and adding support for the EXT4 filesystem required for VITIS acceleration designs.

Once everything is finished, we build the PetaLinux OS image and create a BIF file that describes all the boot components, which will be required by VITIS.

Vitis platform creation

We need to import now our platform into the Xilinx Vitis tool, to create an application based on our device with the Linux system we created.

First we need to create a platform project using the .XSA and provide the necessary files from the PetaLinux OS we have created. After we can create a Vitis application on the Vitis platform

In our demo we are using a Vitis-application example vector addition from the Xilinx template library, based on the platform we created. We built the application and got the boot components and .xclbin file and application executable.

5.1.2. Python script to run

In order to run our PYNQ-overlay in python we need to import Overlay and XInk from PYNQ library. Also we will need the setitem from Operator library and the numpy library, in order to import/export our inputs/outputs to and from the kernel.

⁷ https://github.com/Xilinx/Vitis-Tutorials/blob/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/ref_files/step1_vivado/system_step1.tcl

⁸ https://github.com/Xilinx/Vitis-Tutorials/blob/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/ref_files/step1_vivado/export_xsa.tcl



```

from pynq import Overlay
from pynq import Xlnk
from operator import setitem
import numpy as np

N = 4096 #Number of elements

#define kernel
ol = Overlay('/media/sd-mmcbk0p1/vadd_2/krn1_vadd.xclbin')
#check available ips in the overlay
ip_dict = ol.ip_dict
ip_name = next(iter(ip_dict))
print('-----IP:', ip_name, ('-----'))
#choosing the ip
kernel = eval("ol."+ip_name)
#print register map
regs = kernel.register_map
print(regs)
#Setting size of elements to N
kernel.register_map.size = N
#Creating interfaces for I/O using numpy
in1 = allocate(N, np.int32)
in2 = allocate(N, np.int32)
out_r = allocate(N, np.int32)
#Address mapping
regs.in1 = in1.device_address
regs.in2 = in2.device_address
regs.out_r = out_r.device_address
#initialize input registers with values
[setitem(in1,i,i+1) for i in range(N)];
[setitem(in1,i,1) for i in range(N)];
#run the hardware
regs.CTRL.AP_START = 1
#print outputs
out_r
    
```

Figure 21. Python code for executing the vadd application and IP core

Initially we load in an Overlay object our .xclbin, then we can search for our IP in this object. After finding our IP, we need to load the specific IP (in our case *krnl_vadd_1* IP). We can see in our terminal the output of our kernel's register map:

- **CTRL** signal which contains AXI-interface control signal
 - **AP_START**: setting to 1 initiates the kernel-execution, otherwise not
 - **AP_DONE**: if 1 indicates that the application is executed successfully (this signal is output for 1 clock period)
 - **AP_IDLE**: if 1 indicates that the kernel is not running, otherwise if 0 indicates that the kernel is still running
 - **AP_READY**: if 1 indicates that the kernel is ready to accept input, otherwise not
 - **AUTO_RESTART**: if 1 indicates that the kernel will re-run after the end of each execution, otherwise if 0 it will stall and wait for new **AP_START** =1 signal.
- **in1, in2**: our stream-inputs to the kernel

- **out_r**: our stream-output from the kernel
- **size**: the length of our inputs/outputs

We can set the values of *register_map* object to our desired values. In our case *size* variable is set to *N*, which is the number of elements that we will insert into the AXI-stream inputs *in1*, *in2* and our output *out_r*. To do this we need to reserve continuous memory for our I/O. We can achieve this by using the *allocate* function, setting the number of elements *N* and the bit-width of each elements (in our case 32-bit integers), which outputs an address that we will assign to our register-map elements *in1*, *in2* and *out_r*. Then we can set the values on each allocation by using the *setiitem* function.

After completing all the above we can start our kernel execution by setting the value of *AP_START* to 1. When *AP_DONE* signal gets to 1, this means our kernel execution has finished, and we can see the outputs on our terminal captures bellow. We can see that the vector addition has been completed successfully. First two array prints are our inputs *in1* and *in2* respectively. The third one is our output array before execution. The last output array is our *out_r* array after execution, correctly adding *in1* and *in2* inputs.

```

GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@petalinux:/media/sd-mmcblk0p1# export XILINX_XRT=/usr
root@petalinux:/media/sd-mmcblk0p1# python3 test.py
[ 65.405106] [drm] Pid 787 opened device
[ 65.411068] [drm] Pid 787 closed device
[ 65.418004] [drm] Pid 786 opened device
[ 65.421938] [drm] Pid 786 closed device
[ 65.435361] [drm] Pid 786 opened device
/usr/lib/python3.7/site-packages/pynq/pmbus.py:230: UserWarning: Could not initialise libsensors library
warnings.warn("Could not initialise libsensors library")
[ 68.116608] [drm] Finding IP_LAYOUT section header
[ 68.116611] [drm] Section IP_LAYOUT details:
[ 68.121443] [drm]   offset = 0x126ae98
[ 68.125710] [drm]   size = 0x58
[ 68.129455] [drm] Finding DEBUG_IP_LAYOUT section header
[ 68.132595] [drm] AXLF section DEBUG_IP_LAYOUT header not found
[ 68.137903] [drm] Finding CONNECTIVITY section header
[ 68.143813] [drm] Section CONNECTIVITY details:
[ 68.148853] [drm]   offset = 0x126aef0
[ 68.153377] [drm]   size = 0x28
[ 68.157120] [drm] Finding MEM_TOPOLOGY section header
[ 68.160252] [drm] Section MEM_TOPOLOGY details:
[ 68.165293] [drm]   offset = 0x126ada0
[ 68.169817] [drm]   size = 0xf8
[ 68.173584] [drm] Download new XCLBIN D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42 done.
[ 68.176731] [drm] zocl_xclbin_read_axlf d39c65d4-09c2-4cf4-8fb6-3489caf6ae42 ret: 0.
[ 68.192171] [drm] -> Hold xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, from ref=0
[ 68.199908] [drm] <- Hold xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, to ref=1
[ 68.207413] [drm] No ERT scheduler on MPSoC, using KDS
[ 68.219963] [drm] scheduler config ert(0)
[ 68.219964] [drm]   cus(1)
[ 68.223965] [drm]   slots(16)
[ 68.226659] [drm]   num_cu_masks(1)
[ 68.229615] [drm]   cu_shift(16)
[ 68.233094] [drm]   cu_base(0x80010000)
[ 68.236315] [drm]   polling(0)
[ 68.240167] [drm] -> Release xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, from ref=1
[ 68.243217] [drm] now xclbin can be changed
[ 68.250951] [drm] <- Release xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, to ref=0
[ 68.258378] [drm] -> Hold xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, from ref=0
----IP: krnl_vadd_1 ----
RegisterMap {
  CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, AUTO_RESTART=0),
  in1 = Register(value=0),
  in2 = Register(value=0),
  out_r = Register(value=0),
  size = Register(value=0)
}
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54

```

Figure 22. Terminal output of the register map

```

GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Control signals View Help
[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
253 254 255 256]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217
218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235
236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253
254 255 256 257]
[ 68.265968] [drm] <- Hold xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, to ref=1
[ 68.639770] [drm] -> Release xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, from ref=1
[ 68.647082] [drm] now xclbin can be changed
[ 68.654817] [drm] <- Release xclbin D39C65D4-09C2-4CF4-8FB6-3489CAF6AE42, to ref=0
[ 68.658998] [drm] Pid 786 closed device
root@petalinux:/media/sd-mmcblk0p1#
/dev/ttyUSB1 115200-8-N-1 DTR RTS CTS CD DSR RI
    
```

Figure 23. Terminal output of the final output of vadd

5.2. Intelligent Data Processing Demonstration using software reconfiguration

Edge devices can become the entry and disaggregated point of initial computation for data processing and harmonization. One probably issue that may arise is the specialized Operating System (OS) that these devices support such as PetaLinux, which does not support any package manager and is hard to support updated modules and libraries.

One solution to alleviate this problem is to use Docker. Docker can create containers based on the device’s processor architecture, since Docker shares the kernel of the host machine and therefore not all images can be run and use any kind of OS such as Ubuntu that can be hosted on that processor. In such OSes package managers can be made use of to install all relevant libraries. Containers are also



lightweight can run on such devices and require only the necessary disk space, memory and CPU resources.

To this end, this demo showcases the use of Docker containers on an Ultra96v2 device that acts as the EnerMan intelligent node. The Ultra96v2 board (similarly to the Xilinx ZCU104 board that is used in the previous demo) hosts a Xilinx Zynq UltraScale+ MPSoC ZU3EG chip that has an A484 Arm processor with 2 GB (512M x32) LPDDR4 Memory and a 16 GB microSD card. We have performed this demo on the Ultra96v2 board that has deployed the full EnerMan edge node capabilities (as those are described in section 2 of this deliverable) and aim to demonstrate the usage of the EnerMan execution environment for edge intelligence processing on top of the PetaLinux OS. Using the Docker container capabilities for software flexibility we can elevate the various constraints of the PetaLinux OS and offer support for widely used python-based ML libraries (eg. PyTorch)



Figure 24. The ULTRA96 Board that is used in this Demo

The application to run on the Docker container is a small neural network (for more details see section 3.2.8) comprised of 4 layers, two convolution layers and two fully connected layers written in Pytorch. The main concept is that we will have multiple Ultra96v2 (local clients) running, each training on a set of data (local data) that will be provided by the EnerMan pilots and updating its local model. After that, the results (updated models) of these training will be sent to a central server which will collect all the distributed neural network configurations and aggregate them into a global model. The above scenario forms the basis of the federated learning scheme that is described in section 3 of this deliverable (performing deep anomaly detection) where the server and the multiple Ultra96v2 work jointly without sharing local data.

As our solutions are not currently part of a public Docker repository, we initially have to write a Dockerfile that will perform the building of the docker by downloading the relevant docker base OS and then perform a number of actions that will set up the container to be ready and compatible to run our application. The following code is the used Dockerfile:

```
#define base
FROM ubuntu:20.04
#Update and upgrade
RUN apt-get update
RUN apt-get dist-upgrade -y
RUN apt-get install -y apt-utils git vim
#Install python
RUN apt-get install -y python3 python3-pip
RUN pip3 install --upgrade pip
RUN pip3 install pandas
RUN pip3 install sklearn
RUN pip3 install torch torchvision torchaudio torchtext torchcsprng -f
https://torch.kmtea.eu/whl/stable.html

#Create directory
RUN mkdir /script
#Set temp workdir
WORKDIR /script/
#Get the files in the script folder
COPY ./anomaly_detection_script.py /script/
COPY ./data.csv /script
```

When building this container, the build process will pull the ubuntu 20.04 distribution, update and upgrade it and install all relevant libraries, such as python3, pip3 and using pip3 install all python modules required for the training of the neural network. Finally, this Dockerfile copies the python script and the relevant data for training.

We start the build process by executing the following command in the same directory we have all the necessary files, the Dockerfile, the python script and the data:

```
docker build -t nnedgetrain .
```

The process begins by performing all the steps in the Dockerfile, such as the following images:

```
root@ultra96v2-2020-1:~/Workspace/AIDocker# docker build -t nnedgetrain .
Sending build context to Docker daemon 177.9MB
Step 1/13 : FROM ubuntu:20.04
--> d5ca7a445605
Step 2/13 : RUN apt-get update
--> Running in f04df27c83b8
Get:1 http://ports.ubuntu.com/ubuntu-ports focal InRelease [265 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports focal-updates InRelease [114 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports focal-backports InRelease [101 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports focal-security InRelease [114 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports focal/universe arm64 Packages [11.1 MB]
Get:6 http://ports.ubuntu.com/ubuntu-ports focal/main arm64 Packages [1234 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports focal/multiverse arm64 Packages [139 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports focal/restricted arm64 Packages [1317 B]
Get:9 http://ports.ubuntu.com/ubuntu-ports focal-updates/restricted arm64 Packages [3530 B]
Get:10 http://ports.ubuntu.com/ubuntu-ports focal-updates/multiverse arm64 Packages [8777 B]
Get:11 http://ports.ubuntu.com/ubuntu-ports focal-updates/universe arm64 Packages [1035 kB]
Get:12 http://ports.ubuntu.com/ubuntu-ports focal-updates/main arm64 Packages [1217 kB]
Get:13 http://ports.ubuntu.com/ubuntu-ports focal-backports/universe arm64 Packages [7186 B]
Get:14 http://ports.ubuntu.com/ubuntu-ports focal-backports/main arm64 Packages [2680 B]
Get:15 http://ports.ubuntu.com/ubuntu-ports focal-security/universe arm64 Packages [758 kB]
Get:16 http://ports.ubuntu.com/ubuntu-ports focal-security/multiverse arm64 Packages [3242 B]
Get:17 http://ports.ubuntu.com/ubuntu-ports focal-security/restricted arm64 Packages [3286 B]
Get:18 http://ports.ubuntu.com/ubuntu-ports focal-security/main arm64 Packages [846 kB]
Fetched 17.0 MB in 11s (1498 kB/s)
Reading package lists...
Removing intermediate container f04df27c83b8
--> 4d398de5da3e
Step 3/13 : RUN apt-get dist-upgrade -y
--> Running in 1eaadf9cb818
Reading package lists...
Building dependency tree...
Reading state information...
Calculating upgrade...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Removing intermediate container 1eaadf9cb818
--> 976a3ff66e01
Step 4/13 : RUN apt-get install -y apt-utils git vim
--> Running in 5fc876e96120
Reading package lists...
Building dependency tree...
```

```
Downloading idna-3.3-py3-none-any.whl (61 kB)
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.7-py2.py3-none-any.whl (138 kB)
Collecting certifi>=2017.4.17
  Downloading certifi-2021.10.8-py2.py3-none-any.whl (149 kB)
Collecting charset-normalizer~=2.0.0
  Downloading charset-normalizer-2.0.7-py3-none-any.whl (38 kB)
Installing collected packages: urllib3, typing-extensions, idna, charset-normalizer, sprng, torchaudio
Successfully installed certifi-2021.10.8 charset-normalizer-2.0.7 idna-3.3 pytorch-1.10.0 torchtext-0.11.0 torchvision-0.11.1 tqdm-4.62.3 typing-extensions-4.0.0
WARNING: Running pip as the 'root' user can result in broken permissions and is discouraged. Consider
  creating a virtual environment instead: https://pip.pypa.io/warnings/venv
Removing intermediate container 766a0e6f9b84
--> 72134f36a37f
Step 10/13 : RUN mkdir /script
--> Running in c610dfb2784e
Removing intermediate container c610dfb2784e
--> 3ca7df7b18dc
Step 11/13 : WORKDIR /script/
--> Running in 6f7aa1c2135b
Removing intermediate container 6f7aa1c2135b
--> d0e1ef631f0c
Step 12/13 : COPY ./anomaly_detection_script.py /script/
--> 912fdd0588fc
Step 13/13 : COPY ./data.csv /script
--> 781b1be16249
Successfully built 781b1be16249
Successfully tagged nnedgetrain:latest
root@ultra96v2-2020-1:~/Workspace/AIDocker#
```

After the build has been completed the container can be started with the following command:

```
docker run -it nnedgetrain /bin/bash
```

This command will start up the container at the working directory defined in the Dockerfile and provide bash script.


```

root@ultra96v2-2020-1:~/Workspace/AIDocker# docker run -it nnedgetrain /bin/bash
root@9257c2601e4f:/script# ll
total 173736
drwxr-xr-x 1 root root    4096 Nov 24 14:03 ./
drwxr-xr-x 1 root root    4096 Nov 24 14:38 ../
-rw-r--r-- 1 root root    4239 Nov 24 13:42 anomaly_detection_script.py
-rw-r--r-- 1 root root 177885734 Nov 22 14:49 data.csv
root@9257c2601e4f:/script#
    
```

We can now start up the training script and wait for it to finish. Since the Ultra96v2 does not have too much processing power, it takes some time to complete.

```

root@9257c2601e4f:/script# date
Wed Nov 24 14:40:38 UTC 2021
root@9257c2601e4f:/script# python3 anomaly_detection_script.py
Training model
Epoch 1
Epoch 1 finished
Loss: tensor(5.0360, grad_fn=<NllLossBackward0>)
Epoch 2
Epoch 2 finished
Loss: tensor(2.9833, grad_fn=<NllLossBackward0>)
Epoch 3
Epoch 3 finished
Loss: tensor(2.8340, grad_fn=<NllLossBackward0>)
Epoch 4
Epoch 4 finished
Loss: tensor(6.1263, grad_fn=<NllLossBackward0>)
Epoch 5
Epoch 5 finished
Loss: tensor(2.3570, grad_fn=<NllLossBackward0>)
Testing the model
| test loss -0.731 | test acc: 0.952
root@ultra96v2-2020-1:~# date
Wed Nov 24 15:10:48 UTC 2021
root@ultra96v2-2020-1:~#
    
```

5.3. AI Industrial Intrusion Detection Security Demo

This section describes the main steps involved in the development of the MPSoC-hosted EnerMan intrusion detection mechanism, namely the Industrial Intrusion Detection System (I2DS). The process is based upon the FINN⁹ framework provided by Xilinx. The goal of the I2DS is to identify, in near real-time, possible attacks in an industrial environment by using Machine-Learning (ML) based techniques. The overall development process described here has been an expansion on the original presented in [8].

5.3.1. Board set up

For this demo we are using PYNQ, which is an open-source initiative that facilitates the use of Xilinx hardware devices, such as the EnerMan MPSoC. PYNQ offers and supports python productivity bootable images that can be used on a variety of Xilinx development boards, which host the Zynq MPSoC, e.g., PYNQ-Z1, PYNQ-Z2 and ZCU104. In addition, using the PYNQ python library we can deploy on the reconfigurable hardware of the MPSoC using python scripts. In this demo we use the ZCU104 evaluation board.

5.3.2. Set up the host

The training phase is conducted on a host PC using the Xilinx FINN docker container with all the tools and libraries for training the target AI model. Xilinx provides jupyter notebooks within the docker for easier development. The host also needs Xilinx Vivado HLS installed, for the generation of the bitfile

⁹ <https://xilinx.github.io/finn/>

that will be downloaded into the reconfigurable part (Programmable Logic (PL)) of the MPSoC, that is linked with the docker container. To run the docker we just have to run on the host the command `./run-docker.sh` notebook.

5.3.3. Train a quantized MLP with Brevitas

Quantize the dataset

The first step for a Quantized Neural Network (QNN) training is to binarize the dataset. In this demo we are using the TON_IoT modbus dataset created by the UNSW Sydney. This can be achieved with a python script called `dataloader_quantized.py`. This script drops irrelevant columns of the dataset such as date, time and the type of the attack and binarizes all the useful data for the training and keeps the label that shows if we have an attack or not for this data. With this dataset, for every input of four integers and one label, the dataloader creates 111 bits. The final quantized dataset is saved in a NumPy compressed format (`.npz`). This script also divides the dataset into training dataset (~80%) and test dataset (~20%).

Define and train the QNN with Brevitas

For the training phase we use the quantization-aware training (QAT) capabilities offered by Brevitas. Brevitas is a PyTorch research library for quantization-aware training. Our MLP has four fully connected (FC) layers in total: three hidden layers with 64 neurons, and a final output layer with a single output, all using 2-bit weights. We also use 2-bit quantized Rectified Linear Unit (ReLU) activation functions and apply batch normalization between each FC layer and its activation. The number of epochs is 15 and the learning rate 0.01. The notebook gives us post-training information about the training loss and test accuracy. The final test accuracy of the model is 0.912248.

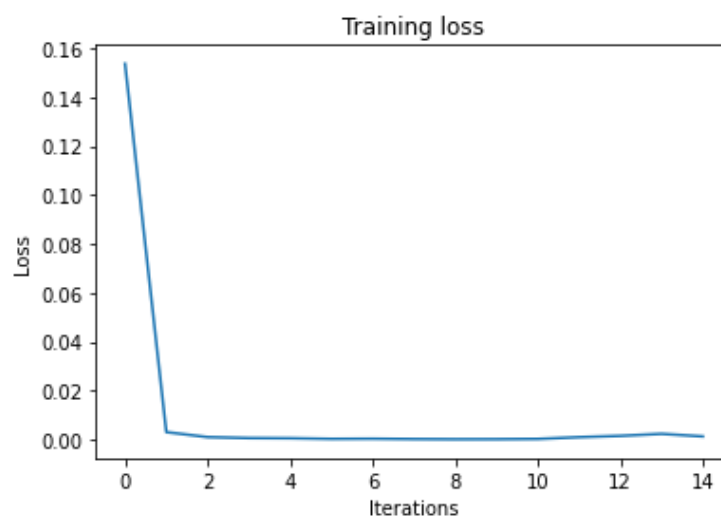


Figure 25. Training loss per iteration

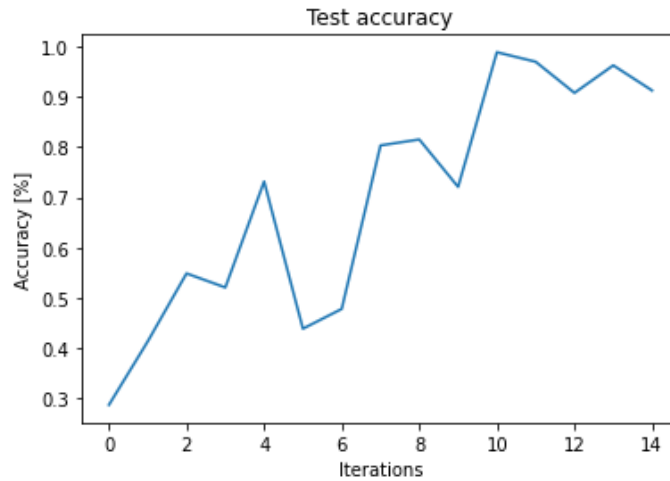


Figure 26. Test accuracy per iteration

Export ONNX model

Before exporting, we can make some changes to our trained network (network surgery). In this case we are padding the input. Our input vectors are 111-bit. For easier parallelization of the first layer, we add a 0-valued column to work with an input size of 112 instead. The FINN compiler expects an ONNX model as input. ONNX is an open format built to represent machine learning models and the output of our model is presented below.

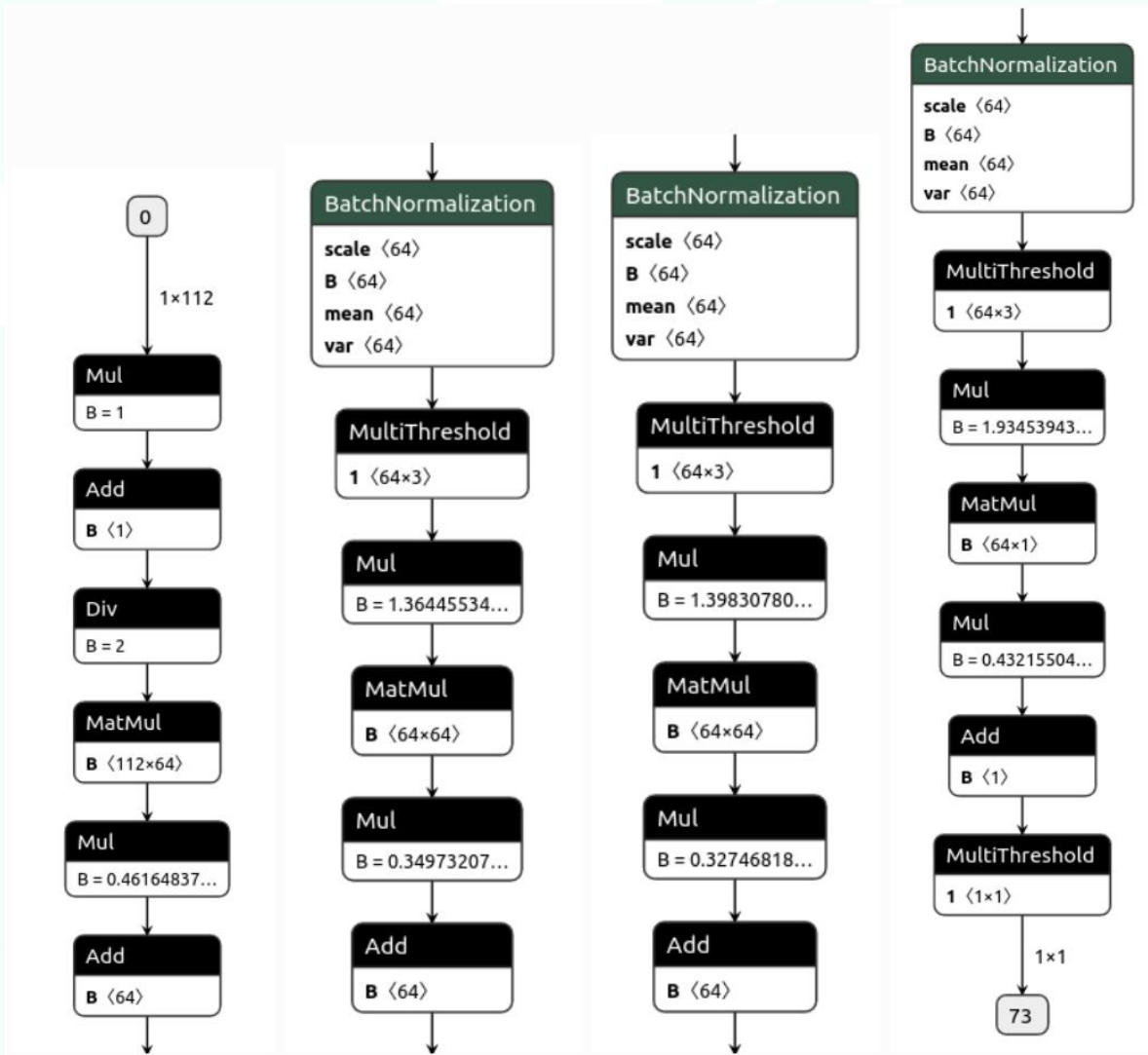


Figure 27. The exported ONNX model in Netron

5.3.4. Import model into FINN and compare it with Brevitas execution

The wrapper around the ONNX model provides several helper functions, so we can extract information about the structure and properties of the model. Before the comparison with the Brevitas execution, we need to prepare our FINN-ONNX model. With the Graph transformations in FINN we transform the model into a synthesizable hardware description. Finally, we can compare the two models by calling our inference helper functions for each input and comparing the outputs.

5.3.5. Synthesis of the accelerator and generation of the bitfile

In this step we use the FINN compiler to generate an FPGA accelerator with a streaming dataflow architecture from our QNN. With the use of the Vivado HLS we map all the layers of the model into hardware description. Hence, we create a hardware architecture with parallel layers that are connected with FIFOs to a full accelerator. Because the synthesis phase is time consuming, we always test our architecture rtl simulation.

```

Final outputs will be generated in output_final
Build log is at output_final/build_dataflow.log
Running step: step_tidy_up [1/16]
Running step: step_streamline [2/16]
Running step: step_convert_to_hls [3/16]
Running step: step_create_dataflow_partition [4/16]
Running step: step_target_fps_parallelization [5/16]
Running step: step_apply_folding_config [6/16]
Running step: step_generate_estimate_reports [7/16]
Running step: step_hls_codegen [8/16]
Running step: step_hls_ipgen [9/16]
Running step: step_set_fifo_depths [10/16]
Running step: step_create_stitched_ip [11/16]
Running step: step_measure_rtlsim_performance [12/16]
Running step: step_out_of_context_synthesis [13/16]
Running step: step_synthesize_bitfile [14/16]
Running step: step_make_pynq_driver [15/16]
Running step: step_deployment_package [16/16]
Completed successfully
CPU times: user 4.38 s, sys: 422 ms, total: 4.8 s
Wall time: 49min 14s
    
```

Figure 28. The steps towards the bitfile generation

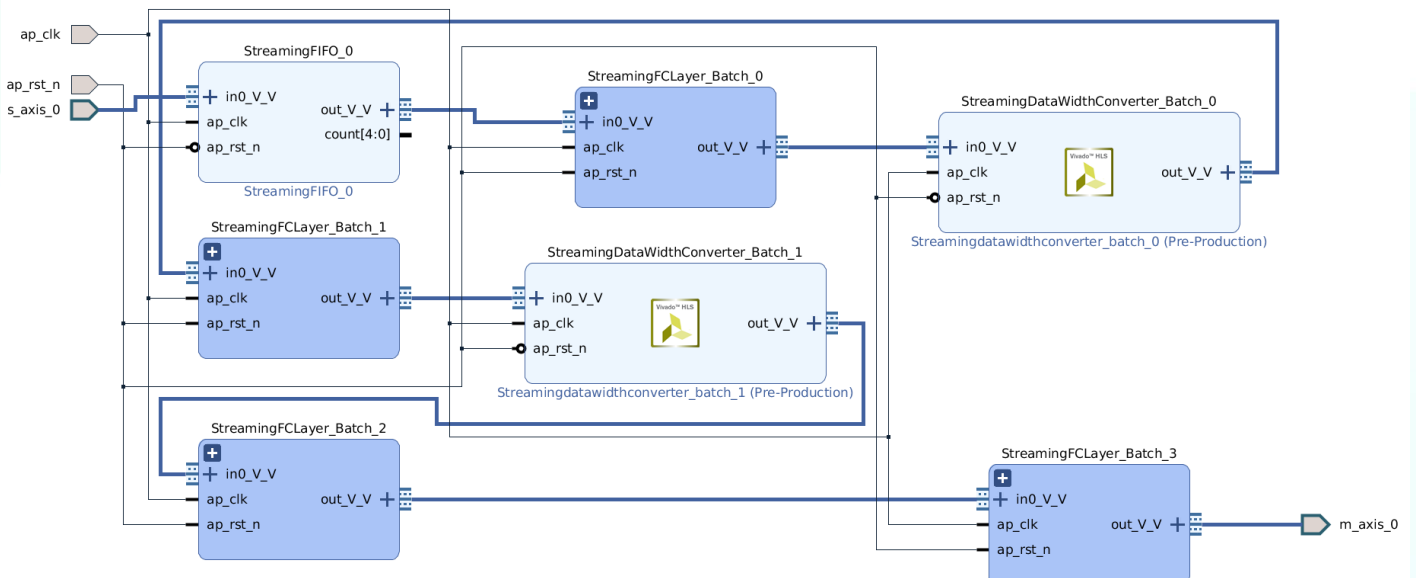


Figure 29: Block design architecture in Vivado

The final output of this process is the bitfile (and the accompanying .hwh file) that will be downloaded to the board. To test the accelerator on the board, we put a copy of the dataset and a premade python script that validates the accuracy into the driver folder, then make a zip archive of the whole

deployment folder. Finally, we send the zip folder to the board and run the commands below for testing our accelerator.

```
unzip deploy-on-pynq.zip -d finn-l2DS-demo
```

```
cd finn-l2DS-demo/driver
```

```
sudo python3.6 -m pip install bitstring
```

```
sudo python3.6 validate_TONIoT.py --batchsize 1000
```

```
Loading dataset...
Initializing driver, flashing bitfile...
Starting...
batch 1 / 10 : total OK 1000 NOK 0
batch 2 / 10 : total OK 1994 NOK 6
batch 3 / 10 : total OK 2918 NOK 82
batch 4 / 10 : total OK 3904 NOK 96
batch 5 / 10 : total OK 4633 NOK 367
batch 6 / 10 : total OK 5633 NOK 367
batch 7 / 10 : total OK 6633 NOK 367
batch 8 / 10 : total OK 7633 NOK 367
batch 9 / 10 : total OK 8483 NOK 517
batch 10 / 10 : total OK 9392 NOK 608
Final accuracy: 93.920000
```

Figure 30. Terminal Output

6. CONCLUSION

In this deliverable, the preliminary research, design, and implementation activities of T2.1 and T2.4 as well as some indicative applications of T2.2 have been presented. The execution environment of the EnerMan edge/end node is described and the overall architecture to support it is presented. Also, a series of demonstration scenarios have been provided showing how the execution environment can be used. Also, some indicative application activities that are been implemented on top of the EnerMan edge/end node execution environment are demonstrated. Currently, the work performed in WP2 is in progress and the reported activities in this deliverable are going to be enhanced and refined further till the close of the WP2 work in M18.

7. REFERENCES

- [1] Zhu, H., Xu, J., Liu, S., & Jin, Y. (2021). Federated learning on non-IID data: A survey. *Neurocomputing*, 465, 371–390. <https://doi.org/10.1016/J.NEUCOM.2021.07.098>
- [2] McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A., 2017a. Communication-efficient learning of deep networks from decentralized data, in: *Artificial Intelligence and Statistics*, pp. 1273–1282.
- [3] Arivazhagan, M.G., Aggarwal, V., Singh, A.K., Choudhary, S., 2019. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*.
- [4] Ghosh, A., Chung, J., Yin, D., Ramchandran, K., 2020. An efficient framework for clustered federated learning. *arXiv preprint arXiv:2006.04088*.
- [5] Brkovic, A., Gajic, D., Gligorijevic, J., Savic-Gajic, I., Georgieva, O., & Di Gennaro, S. (2017). Early fault detection and diagnosis in bearings for more efficient operation of rotating machinery. *Energy*, 136, 63–71.
- [6] Zhang, W., Li, X., Ma, H., Luo, Z., & Li, X. (2021). Federated learning for machinery fault diagnosis with dynamic validation and self-supervision. *Knowledge-Based Systems*, 213, 106679. <https://doi.org/10.1016/j.knosys.2020.106679>.
- [7] Y. Xu, Y. Yang, T. Li, J. Ju and Q. Wang, "Review on cyber vulnerabilities of communication protocols in industrial control systems," 2017 IEEE Conference on Energy Internet and Energy System Integration (EI2), 2017, pp. 1-6, doi: 10.1109/EI2.2017.8245509.
- [8] The National Institute of Standards and Technology, *Guide to Industrial Control Systems (ICS) Security*, Special Publication 800-82
- [9] Le Jeune, Laurens, Toon Goedemé, and Nele Mentens. "Towards real-time deep learning-based network intrusion detection on FPGA." *International Conference on Applied Cryptography and Network Security*. Springer, Cham, 2021.

APPENDIX 1. ENERMAN EXECUTION ENVIRONMENT CREATION WORKFLOW

This appendix provides a detailed tutorial on how to create a VITIS application.

Vivado project creation and .XSA export

As discussed, our first step to build our platform for our application is to create a project in the Xilinx Vivado tool with our underlying hardware, connecting our peripherals and other basic components (e.g. AXI buses, interrupt controllers, platform interface etc.).

We are using a Xilinx ZCU104 design from Xilinx for our hardware description¹⁰ and export the corresponding .XSA file which contains all needed information for our board, hardware and its internal connections. Xilinx provides specific automated TCL scripts that creates¹¹ the design and export¹² the generated XSA file.

The output of Vivado that will be used in the following steps is the generated XSA file.

PetaLinux project creation

Next, we will need to setup the appropriate PetaLinux OS for the hardware design that we have generated. A PetaLinux-project can be created using the Xilinx tools.

In this project we aim to create a Linux system which will contain PYNQ-overlay and all other components that are needed for a hardware accelerated application to run. At this stage, we do the following steps:

1. Create the PetaLinux project based on zynqMP template.
petalinux-create -type project -template zynqMP -name zcu104_custom_plnx
2. Import .XSA file created via Vivado-project.
petalinux-config -get-hw-description=<xsa_directory>
3. Configure device tree with a template of zcu104 by selecting the DTG settings and modifying it to the device, in this case zcu104-revc.
4. Modify our Root File System that originally includes the PetaLinux source files:
 - a. Append the CONFIG_x lines below to the
<your_petalinux_project_dir>/project-spec/meta-user/conf/user-rootfsconfig file.
 - i. Packages for base XRT support by appending
`CONFIG_packagegroup-petalinux-xrt``:
 - i. packagegroup-petalinux-xrt is required for Vitis acceleration flow. It includes XRT and ZOCL.
 - ii. xrt-dev is required in 2020.1 even when we're not creating a development environment due to a known issue that a soft link required by the deployment environment is packaged into it. XRT 2020.2 fixes this issue.

¹⁰ https://github.com/Xilinx/Vitis-Tutorials/tree/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104

¹¹ https://github.com/Xilinx/Vitis-Tutorials/blob/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/ref_files/step1_vivado/system_step1.tcl

¹² https://github.com/Xilinx/Vitis-Tutorials/blob/2020.2/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/ref_files/step1_vivado/export_xsa.tcl



- II. Packages for easy system management by appending
 - ``CONFIG_dnf``
 - `CONFIG_e2fsprogs-resize2fs`
 - `CONFIG_parted`
 - `CONFIG_resize-part``
 - i. dnf is for package package management
 - ii. parted, e2fsprogs-resize2fs and resize-part can be used for ext4 partition resize.
 - III. Packages for PYNQ-overlay
 - ``CONFIG_python3-pynq``
 - `CONFIG_python3-audio`
 - `CONFIG_python3-pillow`
 - `CONFIG_pynq-overlay``
 - `CONFIG_libstdc++``
 - i. Python3pynq is need in order to import PYNQ libraries
 - ii. Audio and pillow are dependencies for PYNQ library
 - iii. PYNQ-overlay imports the device drivers for PYNQ-overlay
 - iv. Libstdc++ is needed in order to OpenCL-applications
 - b. Enable selected rootfs packages
 - I. Run ``petalinux-config -c rootfs``
 - II. Select User Packages
 - III. Select name of rootfs all the libraries listed above
5. Modify kernel
- a. CPU IDLE would cause processors get into IDLE state (WFI) when the processor is not in use. When JTAG is connected, the hardware server on host machine talks to the processor regularly. If it talks to a processing IDLE status, the system will hang because of incomplete AXI transactions. So, it is recommended to disable the CPU IDLE feature during project development phase. It can be re-enabled after the design has completed to save power in final products.
 - I. Launch kernel config: ``petalinux-config -c kernel``
 - II. Ensure the following items are TURNED OFF by entering 'n' in the [] menu selection:
 - III. CPU Power Management > CPU Idle > CPU idle PM support
 - IV. CPU Power Management > CPU Frequency scaling > CPU Frequency scaling
 - V. Exit and save.
6. Update the Device tree: The device tree describes the hardware components of the system. Xilinx device tree generator (DTG) can generate the device tree according to hardware configurations from XSA file. User needs to add customization settings in system-user.dtsi for PetaLinux to consume if there are any settings not available in XSA, for example, any driver nodes that don't have a corresponding hardware, or if user need to override any DTG auto-generated configurations. ZOCL driver module is such a module that has no associated hardware, but it's required by Vitis acceleration flow. It's a part of Xilinx Runtime (XRT). We will add it to the system-user.dtsi.
- We will also override `axi_intc_0`'s parameter interrupt inputs numbers from 0 to 32

because there was nothing connected to the interrupt controller in the XSA, but there will be after v++ links the kernel.

- a. Append the following contents to the ***project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi*** file.

```

`&amba {
    zyxclmm_drm {
        compatible = "xlnx,zocl";
        status = "okay";
        interrupt-parent = <&axi_intc_0>;
        interrupts = <0 4>, <1 4>, <2 4>, <3 4>,
            <4 4>, <5 4>, <6 4>, <7 4>,
            <8 4>, <9 4>, <10 4>, <11 4>,
            <12 4>, <13 4>, <14 4>, <15 4>,
            <16 4>, <17 4>, <18 4>, <19 4>,
            <20 4>, <21 4>, <22 4>, <23 4>,
            <24 4>, <25 4>, <26 4>, <27 4>,
            <28 4>, <29 4>, <30 4>, <31 4>;
    };
};

&axi_intc_0 {
    xlnx,kind-of-intr = <0x0>;
    xlnx,num-intr-inputs = <0x20>;
};

&sdhci1 {
    no-1-8-v;
    disable-wp;
};`
    
```

- zyxclmm_drm node is required by Zocl driver.
- axi_intc_0 node overrides interrupt inputs numbers from 0 to 32, set interrupt kind to level high.
- sdhci1 node decreases SD Card speed for better card compatibility on ZCU104 board. This only relates to ZCU104. It's not a part of Vitis acceleration platform requirements.

7. Add EXT4 rootfs support

It's recommended to use EXT4 for Vitis acceleration designs. Petalinux uses initramfs format for rootfs by default, it can't retain the rootfs changes in run time. Initramfs keeps rootfs contents in DDR, which makes user useable DDR memory reduced. To make the root file system retain changes and to enable maximum usage of available DDR memory, we'll use EXT4 format for rootfs in second partition while keep the first partition FAT32 to store the boot files.

Vitis-AI applications will install additional software packages. If user would like to run Vitis-AI applications, please use EXT4 rootfs. If in any case initramfs would be used, please add all Vitis-AI dependencies to initramfs.

1. Let PetaLinux generate EXT4 rootfs

- Run ``petalinux-config``
- Go to Image Packaging Configuration
- Select Root File System Type as EXT4
- Append `ext4` to Root File System Formats
- Exit and save.

2. Let Linux use EXT4 rootfs during boot

The setting of which rootfs to use during boot is controlled by bootargs. We would change bootargs settings to allow Linux to boot from EXT4 partition. There are various ways to update bootargs. Please take either way below.

- Run ``petalinux-config``
- Change DTG settings -> Kernel Bootargs -> generate boot args automatically to NO and update User Set Kernel Bootargs to

```
`earlycon console=ttyPS0,115200 clk_ignore_unused  
root=/dev/mmcblk0p2 rw rootwait cma=512M.`
```

Click OK, exit thrice and save.

3. Note:

- `root=/dev/mmcblk0p2` means to use second partition of SD card, which is the EXT4 partition.
- Please note that we also set these options in bootargs:
 1. `clk_ignore_unused`: it tells Linux kernel don't turn off clocks if this clock is not used. It's useful clocks that only drives PL kernels because PL kernels are not represented in device tree.
 2. `cma=512M`: CMA is used to exchange data between PS and PL kernel. The size for CMA is determined by PL kernel requirements.

8. Build PetaLinux Image

- a. From any directory within the PetaLinux project, build the PetaLinux project
``petalinux-build``

The PetaLinux image files will be generated in `/images/linux` directory

- b. Create a `sysroot` self-installer for the target Linux system
``petalinux-build --sdk``

The generated `sysroot` package `sdk.sh` will be located in `images/Linux` directory. We'll extract it by running ``sdk.sh`` in a terminal and have a `sysroot` folder in `images/Linux` directory.

9. Create BIF (linux.bif) to describe boot components

- a. Add a BIF file (linux.bif) to the `<full_pathname_to_zcu104_custom_pkg>/pfm/boot` directory with the contents shown below.
- b. The file names should match the contents of the boot directory. The Vitis tool expands these pathnames relative to the sw directory of the platform at v++ link time or when generating an SD card. However, if the bootgen command is used directly to create a BOOT.BIN file from a BIF file, full pathnames in the BIF are necessary. Bootgen does not expand the names between the `<>` symbols.

```
/* linux */

the_ROM_image:
{
  [fsbl_config] a53_x64
  [bootloader] <fsbl.elf>
  [pmufw_image] <pmufw.elf>
  [destination_device=pl] <bitstream>
  [destination_cpu=a53-0, exception_level=e1-3, trustzone]
  <b131.elf>
  [destination_cpu=a53-0, exception_level=e1-2] <u-boot.elf>
}
```

Xilinx Vitis platform creation

We need to import now our platform into Xilinx Vitis tool, in order to create an application based on our device with the Linux system we created.

1. Create Platform Project on the Vitis tool and insert a name for our platform
2. Create a new platform from hardware (XSA) and select the .XSA from Vivado project
 - a. Select Operating system Linux
 - b. Select Processor psu_cortexa53_0
 - c. Select Architecture 64-bit
3. In platform.spr select psu_cortexa53 linux on psu_cortexa53
 - a. Bif File: BIF file we created earlier
 - b. Boot Components Directory: select /images/linux directory
 - c. Linux Image Directory: select /images/linux directory
 - d. Linux Rootfs: select rootfs.tar.gz in /images/linux directory
 - e. Bootmode: SD
 - d. Sysroot Directory: select the export of sdk.sh we run in previous steps
4. Build platform, by right-click on the explorer on the platform we created and selecting Build Project

Now if you create a Vitis application in the same workspace as this platform, you can find this platform available in the platform selection page in platform creation wizard. If you'd like to reuse this platform in another workspace, add its path to PLATFORM_REPO_PATHS environment variable before launching Vitis GUI, or use "Add" button in platform selection page of Vitis GUI to add its path.

In our case we are using a Vitis-application example vector addition from the Xilinx template library, based on the platform we created. We build the application and get the boot components and .xclbin file and application executable.

To create our bootable SD-card, we need two partitions

1. BOOT partition (approximate 4MB), copy the files
 - a. BOOT.BIN

- b. boot.scr
 - c. image.ub
 - d. .xclbin and application executable
2. Rootfs partition (rest space of SD card). Write the rootfs files into the SD-card rootfs-partition:
- ```
sudo tar -zxvf rootfs.tar.gz -C /media/rootfs/
```

# Energy Efficient Manufacturing System Management

enerman-H2020.eu



enermanh2020



enermanh2020



enermanh2020



**HORIZON 2020**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958478

