

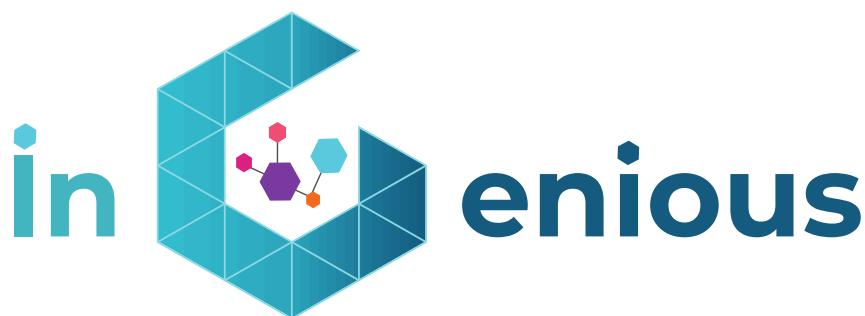


Grant Agreement No.: 957216

Call: H2020-ICT-2018-2020

Topic: ICT-56-2020

Type of action: RIA



D3.3 Secure, private and more efficient HW solutions for IoT devices

Revision: v1.0

Work package	WP 3
Task	Task 3.2
Due date	31/12/2022
Submission date	21/12/2022
Deliverable lead	NCG
Version	1.0
Authors	Clemens Saur (NCG), Carsten Weinhold (BI), Sebastian Haas (BI), Ivo Bizon (TUD)
Reviewers	Nils Asmussen (BI), Tadeusz Puźniakowski (PJATK), Efstathios Katranaras (SEQ), Nuria Molner (UPV), Joe Cahill (iDR)

Abstract	This document covers energy-efficient smart sensor solutions, a highly-secure embedded computer architecture and the co-designed operating system M ³ that turn these sensors into IoT devices.
Keywords	Edge computing, sensors, tile-based computer architecture, operating systems

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.5	31/03/2022	Intermediate version for mid-term review	See author list
V1.0	21/12/2022	Final deliverable, EC-version	See author list

Disclaimer

This iNGENIOUS D3.3 deliverable is not yet approved nor rejected, neither financially nor content-wise by the European Commission. The approval/rejection decision of work and resources will take place at the Final Review Meeting planned in July 2023, after the monitoring process involving experts has come to an end.

The information, documentation and figures available in this deliverable are written by the "Next-Generation IoT solutions for the universal supply chain" (iNGENIOUS) project's consortium under EC grant agreement 957216 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

© 2020 - 2023 iNGENIOUS Consortium

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:		R
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to iNGENIOUS project and Commission Services	

* *R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Executive Summary

This document describes the iNGENIOUS activities towards *secure, private and more efficient hardware solutions for IoT devices*. The deliverable covers the activities within Task 3.2, which is about energy-efficient smart sensor solutions and highly-secure embedded computers that turn these sensors into IoT devices. The requirements from both areas are discussed and suitable and innovative system architectures are derived from them. First, a health-monitoring application for railcar axles using vibration sensors is discussed in detail. This application of IoT sensors focuses on the iNGENIOUS Transportation Platform Health Monitoring use case (Transport UC) and centres around data sampling and analysis at both the edge and in centralized cloud resources. Second, the motivation, design, and current implementation status of a secure-by-default computer architecture and operating system is described. This hardware/software co-design enables system designers to build more secure embedded computers for IoT devices. Isolation of on-device processing and strong protection of data transfers between devices and cloud services are the main benefits for iNGENIOUS supply-chain scenarios and other IoT use cases.



Table of Contents

1	Introduction	8
1.1	Objectives of the Deliverable	8
1.2	Role of T3.2 in iNGENIOUS.....	9
1.3	Structure of the Deliverable	10
2	Edge Sensing and Computation	12
2.1	Edge/Cloud Classification.....	12
2.2	Novelty Identification.....	15
2.3	Power Optimization and Generation	19
2.4	Exploitation Potential and Use Cases	21
3	Ultra-safe IoT Compute Platform	24
3.1	The Case for Hardware/Software Co-Design	24
3.2	M ³ Hardware/Software Co-Design	27
3.3	Enhancements to the Base Platform.....	33
3.4	Security Building Blocks.....	35
3.5	Flexible PHY Implementation on M ³	41
3.6	Exploitation Potential and Use Cases	42
4	Conclusions.....	44



List of Figures

Figure 1.1: iNGENIOUS next-generation supply chain use cases	9
Figure 1.2: Components developed in Task T3.2 within the iNGENIOUS cross-layer architecture	10
Figure 2.1: iNGENIOUS Transport UC – Freight-car rail-health monitoring	13
Figure 2.2: iNGENIOUS Transport UC – Alert location and time reporting	13
Figure 2.3: iNGENIOUS Transport UC – Edge-sensor analytics validation	14
Figure 2.4: iNGENIOUS Transport UC – Context-based diversity & novelty detection	17
Figure 2.5: iNGENIOUS Transport UC – Novelty-data logger (under development)	18
Figure 2.6: iNGENIOUS Transport UC – Energy conservation measures	19
Figure 2.7: iNGENIOUS Transport UC – Hybrid regenerative energy resources	20
Figure 2.8: iNGENIOUS Transport UC – Energy efficiency by pitch size reduction	21
Figure 3.1: Debugging console output of M ³ running a "Hello World" application in the gem5 hardware simulator	26
Figure 3.2: Xilinx Virtex UltraScale+ FPGA (VCU118 board) used to prototype the tile-based computer architecture for the M ³ OS at the circuit level ..	27
Figure 3.3: M ³ base platform with RISC-V tiles, memory tiles, and a debug tile (UDP/IP stack) connected by a network-on-chip with four routers (R) and isolated by TCUs	28
Figure 3.4: Latency of TCU commands	30
Figure 3.5: M ³ x showing image-processing accelerators that are usable from the shell	32
Figure 3.6: M ³ base platform extended with an on-chip network interface card (NIC), virtualized TCUs (vTCU), and a UART interface	34
Figure 3.7: M ³ software components required for secure launch of applications, remote attestation, and secure cross-device communication	39
Figure 3.8: Overview and implementation status of M ³ hardware and components	40
Figure 3.9: Transmitter PHY modules developed for M ³	41

List of Tables

Table 1: FPGA resource consumption: Logic and LUT-RAM (LUTs), registers (Flip-flops, FFs), block RAM (BRAM) with 36 kbit per block..... 29

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
FPGA	Field Programmable Gate Array
HTTP	Hypertext Transfer Protocol
ID	Identification
IoT	Internet of Things
IP	Internet Protocol
IQ	Inverse Quantization
IT	Information Technology
MAC	Medium Access Control
MEC	Multi-access Edge Computing
ML	Machine Learning
NIC	Network Interface Card
NoC	Network on Chip
OS	Operating System
PHY	Physical Layer
RBF	Radial Basis Function
RoT	Root of Trust
RTOS	Real-time Operating System
SDR	Software-Defined Radio
SoC	System on Chip
SVM	Support Vector Machine
TCU	Trusted Communication Unit
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol



1 Introduction

This deliverable aims to present the activities developed in Task T3.2, covering the requirements of security, privacy and efficiency for hardware solutions enabling power-efficient and secure-by-default IoT devices.

In the Internet of Things (IoT), a promising use case for connectivity via high-speed wired or 5G wireless links is to enhance simple and cheap IoT sensors with Artificial Intelligence (AI), Machine Learning (ML), or traditional rule-based algorithms that process sensor data in centralized, remote servers. As enticing as the prospect of more IoT intelligence promises to be, it also opens questions about privacy and security. In consumer-oriented smart home devices, for example, privacy-sensitive information can be processed using AI methods and automations at edge, on the user's device, or it can be offloaded to cloud-based services. Apple's and Amazon's smart-home and voice assistants are representatives of these two models, respectively. The former approach chosen by Apple relies on high-performance AI processing capabilities at the edge. In contrast, Amazon chose the latter architecture which works with less capable and cheaper edge devices by offloading more processing to cloud servers, which are outside the control of the user. The two approaches also have differences in terms of latency and energy requirements depending on how much processing can happen locally and how much data needs to be transmitted over a network.

The different mindsets are not limited to consumer-oriented applications, but also exist among designers of industrial-oriented applications. In industrial applications, privacy concerns may be more about espionage, nevertheless, industrial data shall remain confidential, too. In this setting, it is maybe unimportant whether data is analyzed on a company's edge device or in its secure corporate server environment. A complementary question regards AI/ML models that need training data and/or access to proprietary, confidential information.

To keep both consumer and industrial IoT data confidential, it must be secure from attackers that may try to access it via networks, including the Internet. Hence, securing communication links is critically important to ensure protection of data transfers.

In this document, challenges and solutions for sensor data analysis at the edge verses the cloud, IoT device security, and communication security are discussed from an industrial perspective and related to the INGENIOUS use cases.

1.1 Objectives of the Deliverable

The objective of this deliverable is to describe an exemplary implementation of *secure, private and more efficient hardware solutions for IoT devices*. This implementation consists of an energy-efficient smart sensor solutions and a highly-secure embedded computer platform.

The first part of this deliverable is about a smart sensor solution that considers bandwidth limited connectivity in combination with finite energy resources. It differentiates between a micro-edge sensor which classifies faults and transmits metadata and a novelty detection engine which identifies and logs novelty events to enable data science improvements.

The second part discusses how iNGENIOUS partners advance the state of the art in communication security at the hardware and operating-system level. A computing architecture and operating system are described, along with a detailed discussion of enhancements for device and communication security suitable for both iNGENIOUS use cases and IoT in general.

Although discussed independently in this deliverable, the smart sensor and the highly-secure embedded computing platform can be combined. Within the project's Transport use case, both components are combined in order to enable safe and secure data transmission and sensor access. The combination of these technology concepts is important as it smooths the path to implementation of edge-based swarm intelligence. Micro-Edge sensors can be re-flashed safely and securely based on the insights from a limited number of long term novelty-data loggers over lifetime.

1.2 Role of T3.2 in iNGENIOUS

In WP3, iNGENIOUS aims to evolve both the hardware and software architectures of IoT devices as well as their communication abilities. The goal is to address the current limitations and ease the adoption of devices in next-generation IoT scenarios to support several use cases, which are plotted in Figure 1.1 and further detailed in deliverable **D2.1 Use cases, KPIs and requirements** [1].

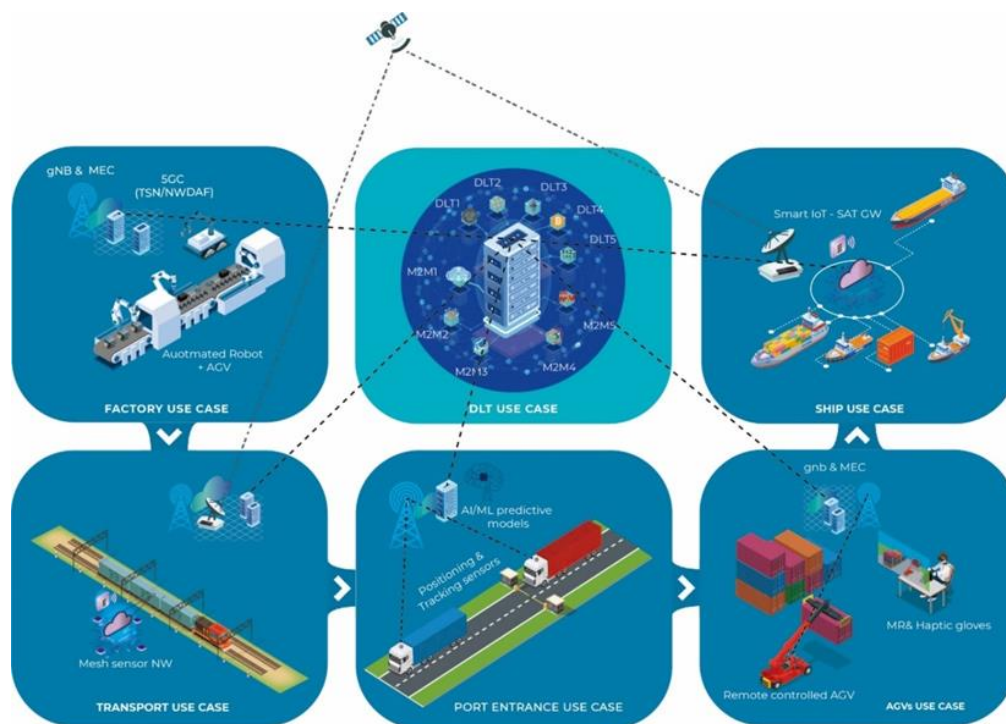


Figure 1.1: iNGENIOUS next-generation supply chain use cases

This deliverable D3.3 will consider the requirements for secure, private and more efficient hardware solutions for IoT devices, and the exemplary implementations described in the objectives above. The concepts described within this document are concentrated in iNGENIOUS Task T3.2 *Ultra-safe low-power dedicated platforms* and are applied in the context of the iNGENIOUS Transport UC, along with an application for the Factory use case. The architectural components discussed in this deliverable are highlighted in Figure 1.2. More information on their roles within the iNGENIOUS cross-layer architecture can be found in iNGENIOUS deliverable D2.4 System and architecture integration (Final) [2].

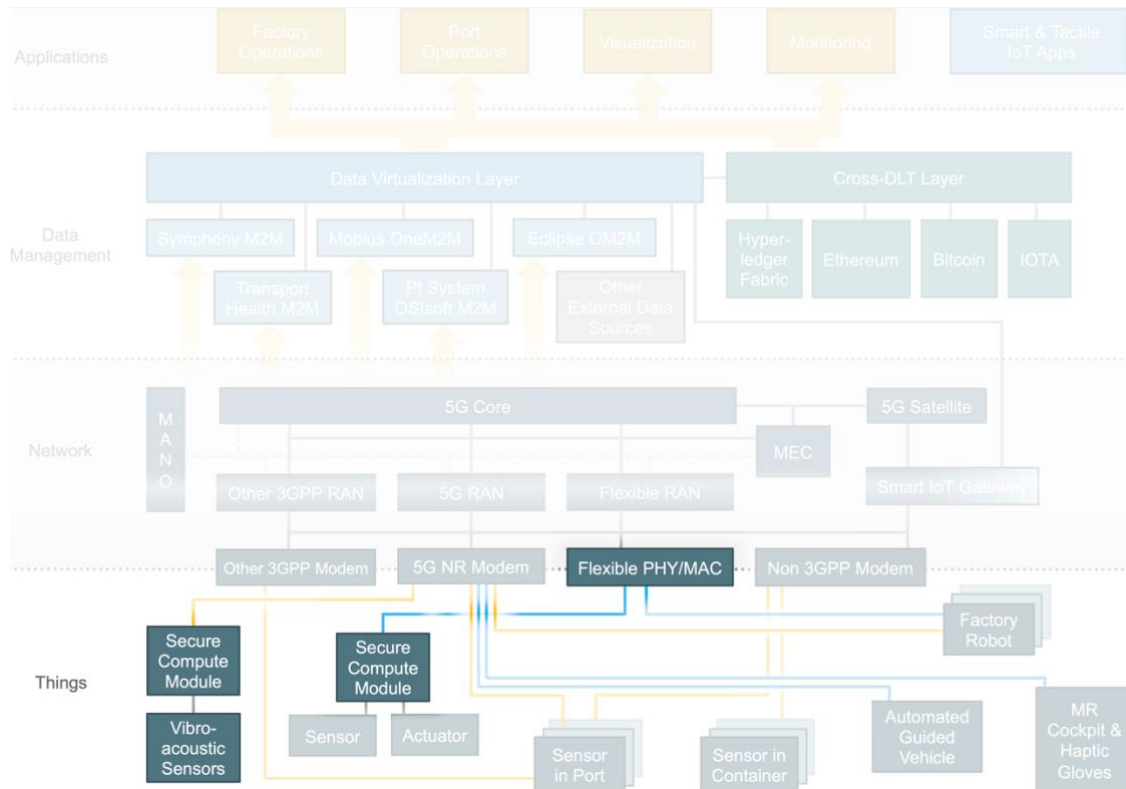


Figure 1.2: Components developed in Task T3.2 within the iNGENIOUS cross-layer architecture

1.3 Structure of the Deliverable

Chapter 2 focuses on the micro-edge sensing implementation solutions for novelty identification and learning, and related economic enablers in energy starved environments, such as power optimization and energy harvesting. It will also describe lessons learned from the Transport UC as well as exploitation opportunities. Chapter 2 is segmented into:

- **Edge/Cloud Classification** (Section 2.1)
- **Novelty Identification** (Section 2.2)
- **Power Optimization and Generation** (Section 2.3)
- **Exploitation Potential and Use Cases** (Section 2.4)

Chapter 3 focuses on a modular approach for building secure-by-default and more efficient computing hardware solutions. This section describes the

open-source development of Barkhausen Institute's tile-based M³ hardware/software co-design platform and innovations on secure communication, remote attestation, and hardware acceleration. Chapter 3 is segmented into:

- **The Case for Hardware/Software Co-Design** (Section 3.1)
- **M³ Hardware/Software Co-Design** (Section 3.2)
- **Enhancements to the Base Platform** (Section 3.3)
- **Security Building Blocks** (Section 3.4)
- **Flexible PHY Implementation on M³** (Section 3.5)
- **Exploitation Potential and Use Cases** (Section 3.6)

The sections on exploitation potential and use case coverage at the end of both main chapters are brief summaries. A detailed discussion will be in iNGENIOUS deliverable **D7.3 Final dissemination, standardisation and exploitation** [3].

Finally, chapter 4 summarizes the achievements and concludes this deliverable.



2 Edge Sensing and Computation

The configuration of Edge Sensing and Computation is driven by latency requirements, power limitations, and data value for example, an autonomous vehicle needs to respond to its environment in real-time without the risk of communication failure or breach. The designer of a battery-powered edge device needs to choose whether to use limited energy resources for local computations on the device or for data transmission. The challenges in meeting such requirements lie not in developing device solutions for systems with unlimited resources, but to devise solutions for systems with severely limited resources. For this reason, rail-health monitoring of freight transports was selected as a representation of a very challenging reference use case.

Logistics is a cut-throat competitive environment. Systems must be tuned for cost performance, without compromising safety and security. While there is a well-established business model for rail-health sensors for passenger transportation, those solutions are far too expensive and service intensive to be applied to freight logistics. The requirements for rail health monitoring of freight carriages are approximately 10 times more ambitious than what exists in the market today:

- Ultra-Low Cost < 25€ per sensor
- Ultra-Long Lifespan > 12 years operation
- Simple, fast, and low-cost installation < 3 minutes
- No external power sources \Rightarrow battery or harvester
- Wake-up on event

The Transport UC aims to show an exemplary implementation for an efficient high-performance low-cost edge IoT application that would meet these challenging requirements.

2.1 Edge/Cloud Classification

Edge computing is as diverse as there are problems and applications. Without restrictions anything is possible. The Transport UC needs to deal with energy and connectivity starved environments with very long operation times. Today most industrial IoT solutions are hardwired and with basically unrestricted power access and connectivity. This works well for fixed pre-installed applications, but it does not work well when installation costs are high, or applications are mobile. In such cases, battery-operated devices are predominantly used. If battery life does not exceed operational life expectancies, then battery maintenance or full sensor replacement becomes necessary. Such state-of-the-art solutions are neither economically efficient nor ecologically friendly.

The iNGENIOUS Transport UC tackles these issues. It is an exemplary implementation of a long-life micro-edge solution with focus on reduced energy consumption and use of alternative energy resources.

One of the most innovative battery-powered edge devices of recent times is the Apple Airtag [4]. Its basic function is to track the location of the item it is attached to and to make beep sounds. Its most interesting feature is that it can operate on a small CR2032 battery for more than one year. It does so without relying on technological breakthroughs, but on smart system architectural design concepts. The sensor relies on random encounters with other mobile Apple devices to broadcast its latest position change via low-power Bluetooth mesh-network communication. An internal motion sensor determines if significant position changes that have occurred since the last broadcast warrant engaging in further random mesh-network communication. In doing so, the Airtag prioritizes its own sensor analytics over energy-consuming network communication, thereby optimizing its power consumption.

The iNGENIOUS Transport UC employs a similar approach. Whenever possible, local sensors analytics are prioritized over communication, even if communication aspects have been optimized for the specific use case, to prolong battery life. The underlying assumption of this strategy is that transmissions consume approximately ten times more energy than sensor-internal processing operations to further compress or meterize data in the edge.

The iNGENIOUS Transport UC is a classical battery-operated condition-monitoring application. Edge-sensors are attached to axles of railcars so they can be used to determine wheel and bearing defects of critical significance (see Figure 2.1).



Figure 2.1: iNGENIOUS Transport UC – Freight-car rail-health monitoring

The edge-sensor shall determine when a fault occurs and if a fault is significant. Fault occurrence is important, because the root cause can be wrong operation by the train operator (e.g., braking induced flat spots) or to problems in the infrastructure (e.g., obstacle or rail defect), which warrant inspection to prevent recurring damage to more railcars. Figure 2.2 visualize how such a defect analysis could work.



Figure 2.2: iNGENIOUS Transport UC – Alert location and time reporting

Cruising speed and payload weight are valuable information for railcar health monitoring. However, this information may or may not be available at system level. If it were available, the edge sensor would have to obtain this information via edge-sensor to edge-gateway communication. Using such information only makes sense, if edge processing can be reduced by a factor of 10 in

comparison to the transmission power needed to receive this information by the edge sensor.

The edge-sensor shall be analytics independent and determine critical event and/or health-condition changes independently. From a power management perspective, a report to the edge gateway should be sent only in case such an event or condition change has occurred.

Analysing the edge-sensors data of a rail axle can be done via crosstalk validation, if one sensor is installed on each wheel of the axle; or via other sensors on the rail carriage if a significant external shock event occurred.

Another pillar of edge-sensor analytics validation is via diverse algorithms. A processing-optimized edge sensor is unlikely to employ complex neural networks for ML-based detection or data-driven Support Vector Machine (SVM) engines. An edge sensor can however perform feature computations to reduce the communication payload by a very significant factor. The edge-sensor analytics validation approach in the iNGENIOUS Transport UC is as shown in Figure 2.3:

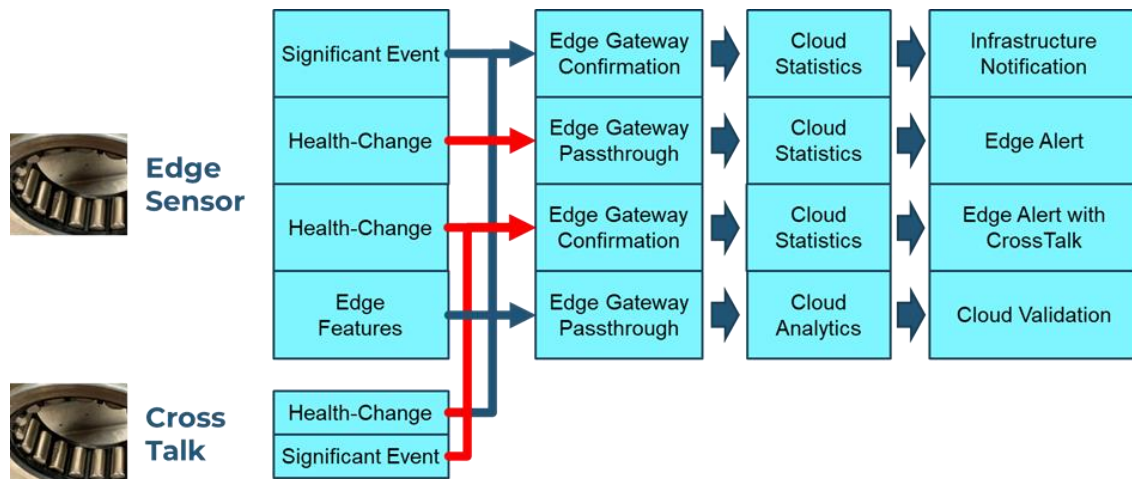


Figure 2.3: iNGENIOUS Transport UC – Edge-sensor analytics validation

The edge-sensor can classify flat-wheel intensity and inner/outer bearing defects with a speed independent algorithm. Since most rail carriages do not have a load sensor for cost reasons, the actual carriage load is typically unknown. Since carriage load has an impact on defect intensity, a worst-case intensity classification may not be known until the next full-load cycle happens.

Flat spots typically occur on both wheels of the same axle, and also among adjacent axles of the same carriage in case of a rapid braking event. This is because the brakes on a carriage are hydraulically actuated, and the braking force is therefore typically equally distributed among the axles. Therefore, flat-wheel health changes, at least significant ones, are reported by multiple axles at the same time. Such simultaneous flat-wheel health change reports implicitly validate edge-sensor classification findings, if all axles on a bogie exhibit similar flat-spot intensities,

Bearing defects are far more difficult to validate. Bearing defects tend to have a much smaller signature, which in some cases, although distinct, cannot be validated via crosstalk, and certainly not via adjacent axle sensors. Bearing defects must be validated via diverse cloud algorithms. For cloud confirmation, an alternate edge feature set is computed which is more suitable for data-based cluster analysis. The cloud analytics consists of both simulated and historic data sets to access the bearing damage and severity. Bearing defects of any degree, once validated, are considered critical and mandate prompt maintenance.

The alternate cloud validation of bearing-defects is done very seldomly, or only when triggered by the edge sensor. This is required to preserve limited battery resources and extend the lifetime of the battery. Unnecessary data communication and its associated energy expenditure must be avoided whenever possible.

This basic edge-sensor \Leftrightarrow edge-gateway \Leftrightarrow cloud classification and communication strategy are the fundamental building blocks to long-lasting battery-powered edge computing.

The strategy however has two weaknesses. One assumption is that the edge-sensor algorithm is robust and mature. The second assumption is that battery power is indeed limited and not fully or partially regenerative. One measure of edge-sensor classification robustness is of course the diverse algorithm validation by powerful data-driven cloud computers. But where is the data coming from? In the iNGENIOUS Transport UC, this issue is tackled via data diversity analysis or novelty Identification.

2.2 Novelty Identification

Data science depends on diverse balanced data covering the full feature space if possible. This is challenging and typically difficult to achieve economically for small science projects. Design of experiments are expensive and resource extensive. A better approach is to analyse data diversity via long duration (product validation testing). Rather than recording all data during such long duration tests, if possible, only diverse data clusters should be logged. This requires recognizing known data clusters from unknown data clusters. A new data cluster describes a hitherto unknown event, which will be referred to as a novelty. In data science terms, a novelty can be considered a data signature or data feature stream which is very different from previous known or experienced data signatures. Recognizing such new data clusters in real-time, in the edge, and without human intervention (i.e., unsupervised novelty detection) is challenging because a novelty refers to something hitherto unknown or different.

Many classification engines segment the complete feature space and classify all data into the defined segments. Such a classification engine is unsuitable for novelty detection as it cannot detect new novelty data clusters. A better approach is to define cluster centroids with limited fields of influence.

iNGENIOUS follows this approach and refines it further. The iNGENIOUS Transport UC uses neuromorphic clusters with limited fields of influence

which search for statistically similar and unknown signatures. A signature is a feature vector derived from the raw data. As the Transport UC is based on cyclic vibro-acoustic signal analysis, this use case specific feature vector focuses on cyclic frequency peaks, harmonics, and peak neighbour relationships. A novelty is a feature vector which is not within a defined distance of a known cluster centroid. This is a completely unsupervised approach and makes the data science project much easier.

Data-driven data analytics need balanced, diverse, and labelled data. Collecting diverse labelled data is typically done by design of experiments [5]. But design of experiments is extremely expensive, time-consuming, and typically incomplete because of limited resources, or unforced omission due to limited ecosystem knowledge.

Limited designs of experiments (physical and simulation) can jump-start application developments by allowing initial or basic algorithm calibrations. The maturity of the algorithms shall grow with more data availability. Especially if diverse edge-cloud algorithms are to be devised to validate findings, the cloud validation will probably be based on big-data analytics.

Diverse data collection can pursue two basic goals. One goal is to extend the knowledge on defects. Another, equally important goal is to expand the knowledge of normal operations to avoid false positives. Whether the system is designed to be more fault-sensitive or more robust against false positives is an economic aspect which is not addressed here.

In the context of iNGENIOUS Transport UC, important questions arise:

- What is a novelty?
- What is diverse data?

Vibro-Acoustic signals have amplitude and frequency characteristics which depend on the length of the time-domain considered. Determining the proper sample-length, time-resolution, and amplitude-resolution is quite application specific. This is where an initial design-of-experiments or simulation model comes in quite useful to determine proper sample parameters.

The sample-length parameter for Transport use-case target signatures is related to wheel speeds. A proper time-horizon can be computed quite easily. As wheel and bearing defects are typically cyclic (assuming that the pendular motion between rails is slow), a proper time horizon would probably be approximately three to four fault signatures at the lowest speed of interest. A data-sample would therefore be considered a data snippet with the minimum time-horizon to include three to four fault signatures at low speed (20-30 kph). While this approach is well suited to pattern analysis, it may not work with iterative algorithms like phase lock loop algorithms, etc.

The proper amplitude resolution depends on the available spectrum and how this spectrum is analysed. If only primary harmonics are being analysed, typically a relatively low amplitude resolution (e.g., 6-8 bits) is more than adequate. If, however, multiple harmonics, or high frequency envelopes, or

broad spectra need analysis, a far higher amplitude resolution (16-24 bits) may be necessary.

The proper time resolution is driven by the highest frequency response to be analysed. Higher harmonics and envelope analysis is a very effective method to suppress random noise signatures. If those methods are employed, higher time resolution may be necessary.

Once the data sample characteristics (lengths, amplitude, and time resolution) have been set, the focus shifts to data variance and diversity. The simplest form of data variance is the standard deviation of the time or frequency domains from their reference. But this is not good criteria for data diversity, as standard deviation is just feature measuring signal variation. Generally, getting rid of the time dependence can be a good thing for variance-analysis if there is not a distinct time-event trigger. A good approach for variance analysis is the use of diverse statistical signal features.

Having good signal feature vectors may not be enough for a good clustering analysis. If signal context can be determined, then signal clustering can be made context specific. The edge gateway in the Transport UC collects information from multiple sensors in addition to its own sensors. The gateway can merge feature vectors from the edge sensors with load information from the load sensor and GPS and gyroscopic information from its own sensors. When edge sensor feature vectors are clustered in different context families, a very thorough data map emerges.

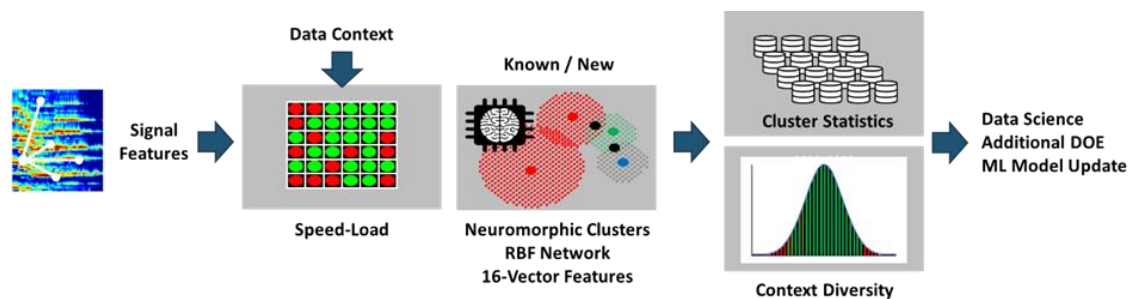


Figure 2.4: iNGENIOUS Transport UC – Context-based diversity & novelty detection

Cluster analysis can be done by various methods. At this stage the focus is neither supervised nor unsupervised learning, it is simply data diversity. While SVMs and K-Means are very popular clustering algorithms they are not suitable for this type of analysis. The focus is to detect new feature-vectors and plot feature-cluster hit rates.

Feature-vector similarity can be determined by standard deviation, or simple Radial Basis Function (RBF) cluster analysis. The goal is to determine what feature-space data samples occur over long periods of time and with what frequency. If at the end of the test period, the returning parts (axles) remain defect free, then all data collected from this series can be labelled as such. If the axle is no-longer defect free, then the initial fault occurrence is analysed via pre-existing fault algorithms, and that data is labelled accordingly.



Figure 2.5: iNGENIOUS Transport UC – Novelty-data logger (under development)

The novelty-data logger, shown in Figure 2.5, has significant energy resources to operate continuously for long periods of time. It continuously samples data points, computes signal feature vectors, and analyses feature-vector diversity via fixed-field-of-influence RBF analysis. If a data sample does not fit previous feature clusters, a new feature cluster is created. The raw data from the first batch of samples (e.g., 20-200 samples) of a given feature cluster are logged for post-processing. In case a data sample fits an existing feature cluster, only its time of occurrence and cluster ID is logged. With this approach of novelty logging, the iNGENIOUS Transport UC aims to build an extensive data library for big data modelling. As the novelty-data logger limits the number of same-cluster signals, it will be avoided that the data library is swamped by similar (biased) data.

The big data can be used to improve the edge algorithms, to improve the cloud validation, or to tune the statistical sensitivity to edge alerts. The novelty-data logger is a development tool to improve field performance. Ideally 50-100 loggers will be employed to continuously expand the big-data library over time.

An interesting use-case example of a novelty-data logger could be when an edge sensor has determined a railcar-health alert. The railcarriage is scheduled for maintenance. A novelty-data-logger gateway sensor is sent to the problematic railcarriage, to log the remaining time of operation before scheduled maintenance. After the maintenance, a detailed fault label is assigned to the logged data prior to maintenance.

2.3 Power Optimization and Generation

The lifetime and performance of a standalone edge sensor is limited by its available energy resources. The typical state-of-the-art energy source in such IoT scenarios is still a non-rechargeable battery. The best way to extend the operational lifetime is to reduce the computation and communication power requirements. Another solution would be to replace the battery with an energy harvester. However, most energy harvesting solutions are not economically attractive, provide too little energy, or are not yet commercially available.

The iNGENIOUS approach is to reduce the micro-edge sensor energy consumption so much, that cost efficient energy harvesting solutions with typically provide limited output power, become a realistic option. The Transport UC reduces computation power requirements by performance optimized algorithms, reducing communication power requirements by minimizing and optimizing communication events and limiting data transmission to optimized meta data. This section will discuss how to keep the sensors alive as long as possible with additional energy conserving measures, and ways to extend sensor lifetime beyond maintenance periods without replacement.

2.3.1 LIFE EXTENDING ENERGY CONSERVATION MEASURES

While all edge sensors and all edge gateways have their own energy resources, the level of these resources varies from sensor to sensor and gateway to gateway.

A railcarriage is a large metallic entity and communication signals are reflected and bounced between metallic surfaces. As there is only one gateway per carriage, the distance between the various edge sensors and the one gateway can vary. The larger the distance, or the larger the metallic interference, the higher the communication power must be to overcome signal loss.

One way to minimize unnecessary communication energy between edge sensor and edge gateway is to optimize the communication gain to the communication distance and signal interference. Another way is to daisy-chain edge sensors to reduce the maximum broadcasting distance between communication participants.

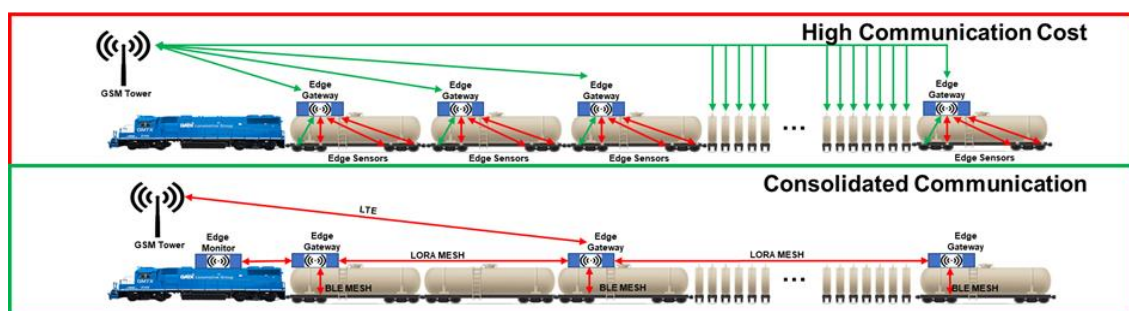


Figure 2.6: iNGENIOUS Transport UC – Energy conservation measures

The same concepts can be applied to gateway-to-gateway and gateway-to-cellular-tower communication. Cellular communication consumes far more energy than LORA communication.¹ Thus, LORA communication can connect edge gateways on the train. Rather than each edge gateway communicating to a cellular tower, data can first be consolidated via LORA communication, and the gateway with the most energy available could facilitate the Cellular tower communication.

2.3.2 HYBRID REGENERATIVE ENERGY RESOURCES

The battery of an edge sensor is not a service part. The environmental and ATEX² explosion requirements will require that the edge device is fully potted which makes it unserviceable. If the lifetime of an edge sensor can be extended to multiple rail service cycles the economic prospects improve significantly. Fortunately, rail carriages exhibit massive vibration sources in the range from 600-1200Hz. This frequency is well-suited for triboelectric energy harvesters. Triboelectric energy harvesters are extremely cost-effective solutions both in terms of direct material costs and associated additional electronics, as they do not require additional step-up circuits.

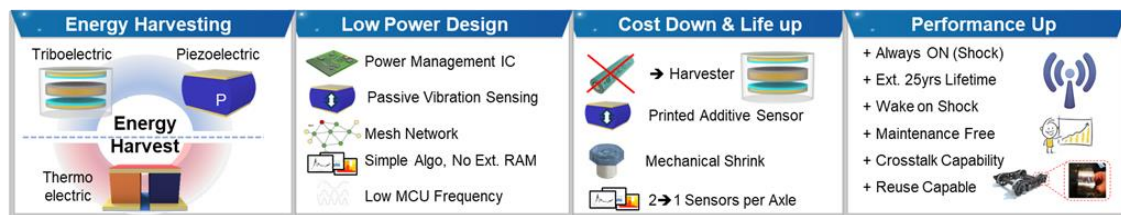


Figure 2.7: iNGENIOUS Transport UC – Hybrid regenerative energy resources

Coupled with adequate capacitor banks, tribo-electric harvesters can replace expensive ATEX prove batteries at near same cost. The impact on the overall system usage would be tremendous. There are still some unknown on the long-term durability of triboelectric materials, however if these are insignificant, then the overall sensor lifetime could be extended to multiple service intervals, or even the lifetime of the rail carriage.

The volumetric requirements for a tribo-electric harvester suitable for the Transport UC are significantly smaller than those of a conventional battery. This allows significant reductions in sensor size and ultimately sensor mechanics and costs, which help to work towards the freight-car rail health extremely tough economic targets.

¹ Transmission peak burst rates from experience can be approximated as follows by technology: BLE Bluetooth 15 mW, WiFi 150-300 mW, LORAWAN 30 mW, LTE 400-600 mW, 3G Backfall 1 W, 2G Backfall 1.5 W.

² ATEX designates design and test requirements for electronic or electrical equipment intended for use in a hazardous area. Electronics used on rail carriages for hazardous material transport (boilers cars) need to be ATEX compliant according to EU directive 94/9/EC.

2.3.3 NEUROMORPHIC COMPUTING

Neuromorphic computing was initially targeted at initial stages of this work in iNGENIOUS as low-power computing because it lacks the processor overhead of conventional CPUs. While the general idea is still valid, the reality is that pitch size reduction for mainstream products is not a match for specialty products like neuromorphic circuits, which are typically manufactured in much larger pitch sizes, therefore have larger power consumption even with reduced transistor count.

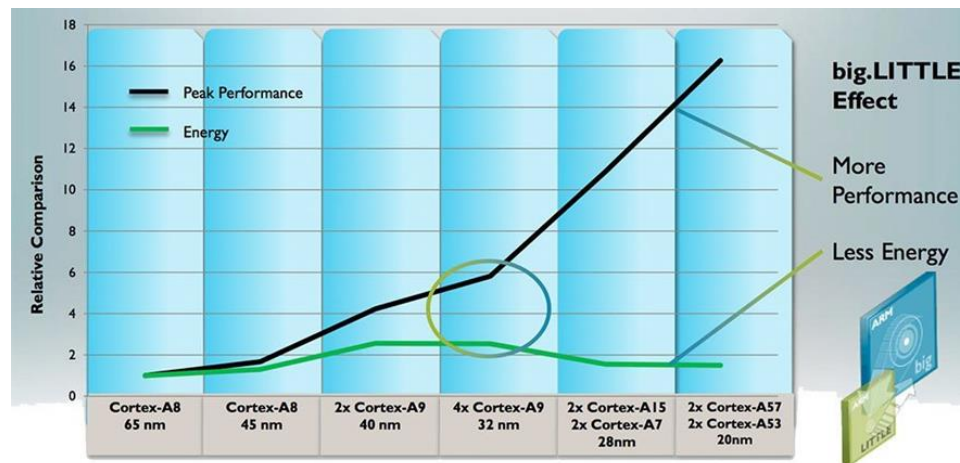


Figure 2.8: iNGENIOUS Transport UC – Energy efficiency by pitch size reduction

2.4 Exploitation Potential and Use Cases

The Transport UC is an complementary roadmap for self-powered micro-edge sensors. Such a solution is ideal for both mobile applications and stationary applications with high hard-wire installation costs.

Replacing “Regular Maintenance” by “Predictive Condition Monitoring” is more effective if regular battery maintenance or sensor replacement is not necessary.

A truly self-powered condition monitoring solution would optimize current implementations and expand to new use-cases that are currently not attractive.

The original plan for directly exploiting the outcome of Transport UC implementation within iNGENIOUS has seen a minor set-back, as self-powered operation was not intended from the outset. But extended operating lifetime requirements and reduced cost-targets allow for no other solution. Therefore, the exploitation plan for rail health monitoring has shifted until self-powered operation via energy harvesting can be proven.

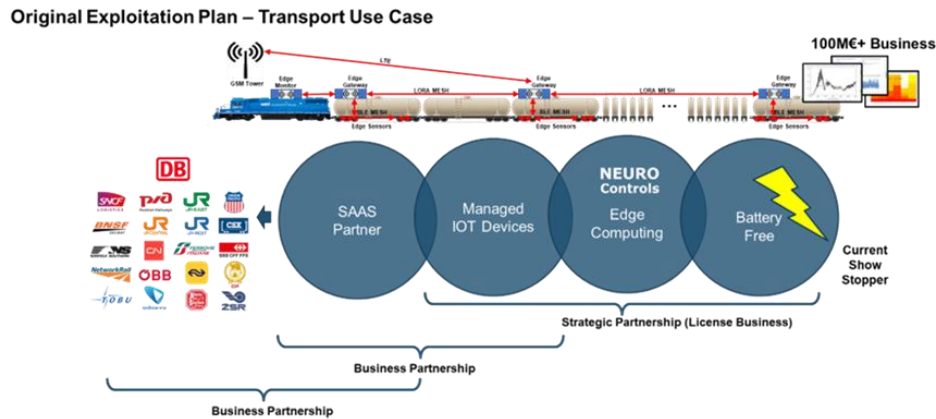


Figure 2.7: iNGENIOUS Transport UC – Rail Health Exploitation Plan

Continued research (contracted, self-financed, grant driven) in this area is ongoing. The immediate exploitation potential is limited to contracted research.

More immediate exploitation potential is provided by the scientific diversity data loggers. Traditional data collection via design-of-experiments is slow and expensive. The scientific diversity data logger collects diverse data clusters during product validation testing and logs cluster samples and corresponding context information (data labels) automatically. This greatly simplifies data collection and is a good starting point for data science. If more data is needed to cover certain context gaps, these tests can be specifically engineered to complete the necessary datasets.

Scientific Diversity Data Logger

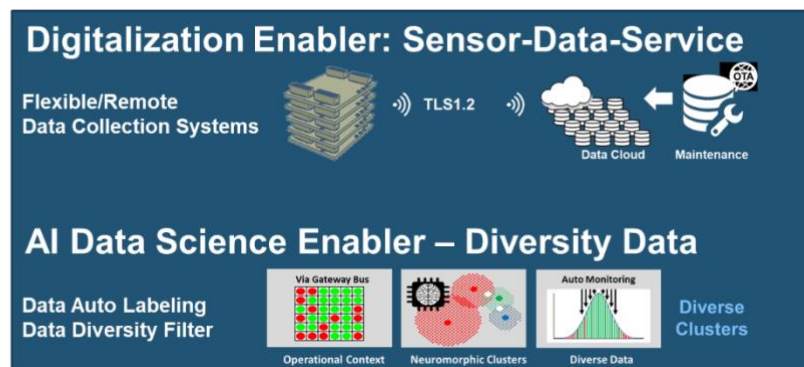


Figure 2.7: iNGENIOUS Transport UC – Scientific Diversity Data Logger

The ultimate micro-edge device will be vibro-acoustically the self-powered condition sensor. It can be used to monitor strain or vibration anomalies to initiate predictive maintenance and event alerts. Similar to the rail-health condition-monitoring, application research on vibro-acoustic energy harvesting must be completed before market deployment is feasible.

Vibro-Acoustic Self-Powered Micro-Edge-Sensor: Condition Monitoring without Battery Maintenance

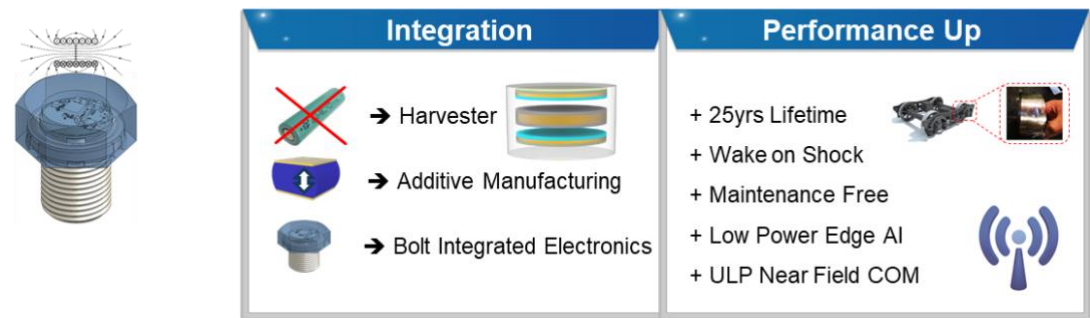


Figure 2.7: iNGENIOUS Transport UC – Self-Powered Micro-Edge Condition Monitoring

The following table states the use case coverage of the technology components described in this chapter according to demonstration and evaluation activities in WP6 of the project. Evaluation results will be reported on in iNGENIOUS deliverable **D6.3 Final iNGENIOUS data management platform** [6]. The summary of exploitation potential shall help the reader assess the value of the technologies. Details are provided in iNGENIOUS deliverable **D7.3 Final dissemination, standardisation and exploitation** [3].

Component: Edge Sensing and Computation
Relevant for use cases: Transport
Exploitation Potential: <ul style="list-style-type: none">• NCG: Freight-Car Condition Monitoring - Test Systems• NCG: Freight-Car Condition Monitoring - Self-Powered Sensors• NCG: Self-Powered Condition Monitoring Bolt - Vibro-Acoustic Applications• NCG: Self-Powered Strain-Gauge Bolt - Vibro-Acoustic Applications• NCG: Scientific Novelty Data Logger - Vibro-Acoustic Applications

3 Ultra-safe IoT Compute Platform

In the IoT, edge devices such as sensors and actuators communicate among each other and with cloud servers. The servers can be physically located in a multi-access edge computing (MEC) cloud relatively close to the devices, or in a remote data center. From an architecture point of view, the IoT is a distributed system, where edge devices and servers represent the nodes that communicate with each other. To send and receive messages, all nodes, including the IoT devices, require local compute capacity that enables them to participate in network-communication protocols. As a result, IoT devices must include computers with networking capabilities.

The functional requirements for enabling network communication force system designers to include complex software that implements the communication protocols. On top of that, they need to include cryptographic algorithms to protect data transfers from eavesdropping and other attacks. Unfortunately, adding complexity to a system makes it harder to achieve security and reliability goals. In fact, it is widely recognized that security and reliability are best ensured by striving for simplicity in design and implementation to avoid mistakes that can potentially lead to failures or security weaknesses.

INGENIOUS seeks to improve the security and reliability of the IoT and therefore tackles the problem of complexity versus functional requirements. To resolve the inherent conflict between complexity that is needed for functional reasons and the need to keep things simple in order to avoid errors, system designers can use isolation between critical and non-critical components. The goal of such separation of functionality is to ensure that failures of or successful attacks on isolated components are restricted to the failed or attacked component, but not all other parts of the system.

This problem needs to be addressed at both edge devices and servers. In this report, we focus on the device side, but the principles can also be applied to secure (and optimize resource usage in) data center environments [7].

The project advances the state of the art through a hardware/software co-design. We justify our approach and describe the basis from which we started in Section 3.1. The pre-INGENIOUS state of our system-on-chip architecture and the operating system M^3 that is specifically tailored to this hardware are described in Section 3.2. After discussing these basics, we describe enhancements to hardware/software platform that benefit INGENIOUS use cases in Section 3.3. Specific solutions for securing communication links between IoT devices and cloud servers will be covered in Sections 3.4. An application scenario is described in Section 3.5. Finally, we discuss how the basic features of M^3 , and the enhancements made in INGENIOUS map to the INGENIOUS use cases in Section 3.6.

3.1 The Case for Hardware/Software Co-Design

A robust and secure-by-default platform for IoT devices must consider both the software and the hardware that is running this software. We describe the

state of the art and iNGENIOUS innovation of our hardware/software co-design in this section.

3.1.1 STATE OF THE ART BEFORE iNGENIOUS

Regarding system software, the concept of a microkernel-based operating system (OS) has been shown to be an effective solution to make computers more secure [8]. A microkernel OS separates all functionality of the OS into isolated components that can cooperate to serve applications, but unintended access to subsystems is limited by hardware-enforced boundaries (e.g., address spaces) and communication control. Isolating a software component prevents access to data or execution of code outside the respective component's address space without explicit permission. It must be noted that microkernel systems are well-researched and have seen some adoption in production environments, but they are not pervasive and in particular, they are rarely used in IoT devices. Instead, most of the time complex OSes such as Linux are used, or real-time executives (RTOS), which are less complex, but lack support for strong isolation between applications and the OS. In deliverable **D3.1 Limitations and improvement axis for the communication of IoT devices** [9], the security-related problems of complex monolithic OSes such as Linux and many RTOS are discussed in detail. The main issue can be summarized as follows: Since there is no isolation between subsystems of a monolithic codebase such as the Linux kernel, a bug in any part of this huge code base can potentially lead to the failure (e.g., a crash) or security attack on all other parts of the code and any data processed by it. For details and some mitigations, see Section 3.2 in iNGENIOUS deliverable D3.1 [9].

3.1.2 iNGENIOUS INNOVATION

To secure IoT edge devices, which can potentially cause physical harm to human life, the environment, or infrastructure, the hardware plays an important role, too. In the project, we continue development of a new, secure-by-design computer architecture that isolates complex and potentially faulty system components. The goal of this architecture is to support the enforcement of isolation in both hardware and software, as needed by the OS and application software. By employing microkernel-like ideas such as isolation of system components in hardware, we further reduce the impact of failures within or attacks on the information technology components.

At the lowest level of the system architecture, especially at the interface between hardware and the OS, it is essential to build both the hardware and the OS in such a way that they can support the overall design goals in the best possible way. Tight integration helps with several design goals:

- Maximizing efficiency (energy consumption, performance, etc.)
- Maximize security (isolation, communication control)
- Customizability (making it possible and easy to add or remove functionality as needed for the specific use case)

- Enable new platform-security capabilities (by building computer systems that are better suited for distributed-computing use cases than existing solutions based on traditional architectures)

To that end, Barkhausen Institute (BI) contributes to iNGENIOUS a hardware/software co-design: a tile-based computer architecture that is hardened against security and reliability failures in hardware, and M³ [10], the microkernel-based OS. M³ is specifically tailored to this hardware and the unique capabilities it provides to the OS.

```

L1i$ =32 KiB (2-way assoc, 4 cycles)
L1d$ =32 KiB (2-way assoc, 4 cycles)
L2$  =256 KiB (8-way assoc, 12 cycles)
Comp =Core -> DTU+AT -> L1$ -> L2$

PE03: build/gem5-x86_64-release/bin/rctmux
Core =TimingSimpleCPU x86_64 @ 1GHz
DTU  =eps:16, bufsz:1024 B, blocksz:64 B, count:4, tlb:128, walker:1
L1i$ =32 KiB (2-way assoc, 4 cycles)
L1d$ =32 KiB (2-way assoc, 4 cycles)
L2$  =256 KiB (8-way assoc, 12 cycles)
Comp =Core -> DTU+AT -> L1$ -> L2$

PE04: build/gem5-x86_64-release/default.img x 1
DTU  =eps:16, bufsz:1024 B, blocksz:1024 B, count:8, tlb:0, walker:0
imem =3145728 KiB
Comp =DTU -> DRAM

Global frequency set at 1000000000000 ticks per second
info: kernel located at: build/gem5-x86_64-release/bin/kernel
info: kernel located at: build/gem5-x86_64-release/bin/rctmux
info: kernel located at: build/gem5-x86_64-release/bin/rctmux
info: kernel located at: build/gem5-x86_64-release/bin/rctmux
warn: DRAM device capacity (49152 Mbytes) does not match the address range assigned (4096 Mbytes)
info: No kernel set for full system simulation. Assuming you know what you're doing
info: No kernel set for full system simulation. Assuming you know what you're doing
platform.com_1.device: Listening for connections on port 3456
0: pe00.remote_gdb: listening for remote gdb on port 7000
0: pe01.remote_gdb: listening for remote gdb on port 7001
0: pe02.remote_gdb: listening for remote gdb on port 7002
0: pe03.remote_gdb: listening for remote gdb on port 7003
warn: CoherentXBar pe04.xbar has no snooping ports attached!
info: Loaded 'root' to 0x8400000000000000 .. 0x8400000000043868
info: Loaded 'hello' to 0x8400000000044000 .. 0x8400000000016eb60
info: Loaded 'hello' to 0x8400000000016f000 .. 0x84000000000299b60
info: Loaded 'rctmux' to 0x840000000029a000 .. 0x84000000002aed08
info: Entering event queue @ 0. Starting simulation...
[kernel @0] Kernel is ready
Hello World
Hello World
[kernel @0] Shutting down
Exiting @ tick 6355915000 because m5 exit instruction encountered

```

Figure 3.1: Debugging console output of M³ running a "Hello World" application in the gem5 hardware simulator

BI has already been developing the M³ hardware/software co-design platform for several years. Within iNGENIOUS, it is being extended to meet the requirements from IoT use cases relevant to iNGENIOUS (but not limited to supply-chain scenarios). As the debugging console output in Figure 3.1 shows, M³ can run in a hardware simulator based on gem5, which “is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture” [11].

In addition to this software-based simulation environment, the hardware of the tile-based computer architecture is available as a system-on-chip prototype in an FPGA circuit simulator. The latter provides the ability to physically connect I/O devices such as the vibro-acoustic sensor hardware

presented in Chapter 2. The FPGA prototype also enables an accurate assessment of performance characteristics and an estimate of hardware metrics such as required chip area for various hardware building blocks.



Figure 3.2: Xilinx Virtex UltraScale+ FPGA (VCU118 board) used to prototype the tile-based computer architecture for the M³ OS at the circuit level

We present the hardware architecture and M³ OS in the next section in more detail, followed by a discussion of enhancements made within iNGENIOUS.

3.2 M³ Hardware/Software Co-Design

This section describes the basics of the M³ hardware/software co-design platform, which also reflects the state of the art before iNGENIOUS. We first explain the hardware and then the software part.

3.2.1 TILE-BASED HARDWARE ARCHITECTURE

The M³ system is based on a tiled hardware architecture which provides high flexibility by default. High-performance general-purpose processors can be integrated together with specialized low-power hardware accelerators. The heterogeneity allows to meet the various performance and energy requirements given by the applications. Furthermore, the isolation-by-default concept allows to integrate untrusted or potentially malicious components such as third-party IP blocks. The approach makes it more difficult for an

attacker to compromise the whole system by exploiting such a single component.

BI already provides the basis for the tiled hardware architecture as depicted in Figure 3.3. The architecture allows to place hardware components such as processing cores, memories, and I/O devices on physically separated tiles. The tiles are connected by a network-on-chip (NoC). Each tile includes a new hardware component called Trusted Communication Unit (TCU) that enables the isolation concept. For general-purpose computing, there are multiple RISC-V tiles each including a Rocket [12] or BOOM [13] core with cache. The cores access main memory via memory tiles, which include interfaces to external DRAM. An Ethernet interface is connected to a debug tile which includes a hardware UDP/IP stack. This tile is only used to access the platform from a host PC to configure the system during start up and for debugging purposes.

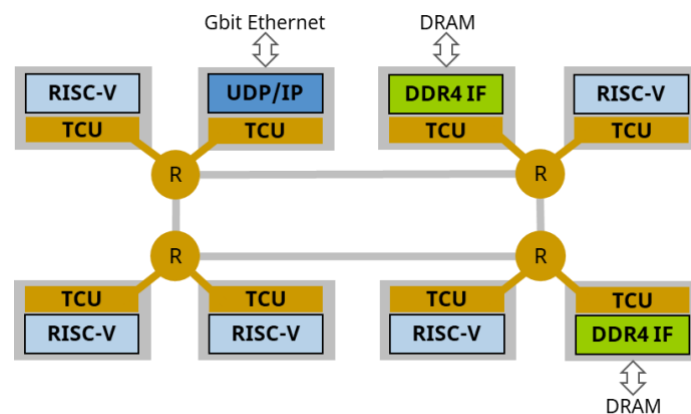


Figure 3.3: M³ base platform with RISC-V tiles, memory tiles, and a debug tile (UDP/IP stack) connected by a network-on-chip with four routers (R) and isolated by TCUs

The system is implemented on a Xilinx Virtex UltraScale+ FPGA (VCU118 board). In contrast to a silicon-chip implementation, the FPGA prototype enables a rapid and flexible realization of the tiled hardware architecture. Hardware designs can be generated within a few hours. Furthermore, it allows the platform to interface with devices and sensors used in the Transport UC of the iNGENIOUS project. For these reasons, the FPGA prototype is an appropriate approach to verify and demonstrate the project results. However, the hardware design on a silicon chip can achieve higher clock frequencies than on the FPGA. Furthermore, measuring the power consumption of individual hardware components on the FPGA is limited. For more accurate power measurements, BI taped-out a part of the M³ platform in a silicon research chip. Since final chip measurements will be available after the project end, we continue to use the FPGA as implementation and measurement platform. Preliminary synthesis results from chip design have been published in [14].

The hardware platform employs RISC-V Rocket cores [12] and BOOM cores [13], which are available as open source. Rocket is a 64-bit RISC-V in-order core with MMU and 16 kB L1 cache, each for instructions and data, as well as a shared 512 kB L2 cache. BOOM is the out-of-order variant of Rocket with the same cache configuration. We set the clock frequencies of the Rocket and BOOM cores to

100 MHz and 80 MHz, respectively, to fully meet the timing requirements during FPGA synthesis and place-and-route. The main memory is located in the external DDR4 DRAM (2x 2GB).

An overview of the consumed FPGA resources is given in Table 1. The TCU is presented in two configurations. The TCU in the RISC-V tile includes all features such as memory transfers, message passing, physical memory protection (PMP), and provides interfaces to the NoC and the core. The TCU in the DDR4 tile is not connected to a core and, hence, does not need to include features to actively send data or messages. Instead, it only processes read and write requests from the NoC and forwards them via the memory interface to the external DRAM. As a result, the TCU in the DDR4 tile can be reduced and thus requires 3.7x fewer Look-up Tables (LUTs) than the TCU in the RISC-V tile. In comparison to the BOOM and Rocket core, the TCU in the RISC-V tile only requires 10.0% and 30.8% of the FPGA LUTs, respectively. Since the TCU contains no memory or caches, the number of required FPGA Block RAMs (BRAMs) is negligible compared to the cores.

Table 1: FPGA resource consumption: Logic and LUT-RAM (LUTs), registers (Flip-flops, FFs), block RAM (BRAM) with 36 kbit per block

	LUTs [k]	FFs [k]	BRAMs
BOOM	143.8	71.8	159
Rocket	46.4	22.0	152
NoC router	3.4	2.2	0
DDR4 interface	15.3	18.3	25.5
TCU (RISC-V tile)	14.3	5.5	0.5
TCU (DDR4 tile)	3.9	1.7	0.5

Besides resource consumption, we measured the latency of the TCU to evaluate its timing overhead when executing commands initiated by the core [14]. Throughput is limited by the bandwidth of the NoC (16 bytes/cycle) since the current TCU implementation supports the full NoC bandwidth. For these experiments, we use two identical processing tiles each including a TCU and a RISC-V Rocket core. The tile which initiates commands contains the so-called local TCU, while the receiving tile contains the remote TCU. The tiles are connected to a single NoC router to minimize the delay induced by the NoC.

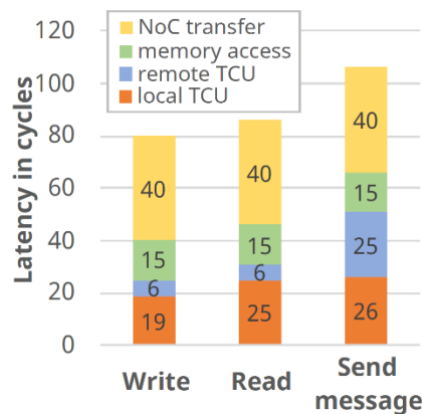


Figure 3.4: Latency of TCU commands

Figure 3.4 shows the measurement results of a DMA-write and read as well as for sending and receiving a message (with 8-byte payload data each). For each transfer, the local TCU requires 11 cycles to initiate the command by reading the corresponding endpoint and validating the access permissions. The remaining cycles of the local TCU are spent to prepare the access to the RISC-V cache and to finish the command as soon the response/acknowledgment packet has been received. The time of a memory access of the RISC-V core is specific to this core and depends on its internal cache access times. For the write and read commands, the remote TCU requires 6 cycles to forward the requests to the memory. For message passing, the remote TCU takes about 25 cycles on average. This includes finding a free slot in the memory-mapped receive buffer and setting up an endpoint for a reply. The NoC transfer delay is about 20 cycles per packet via the single router, which is mainly caused by the synchronization registers in the NoC interface. Asynchronous transitions enable to set different clock frequencies of the NoC and the tile. Since each command consists of a data and a response/acknowledgment packet, we measured 40 cycles for the total transfer.

The software consists of the M³ operating system and applications that run on the RISC-V cores. The binaries are compiled with the RISC-V toolchain and are compatible with both the Rocket and BOOM cores on the FPGA and the RISC-V models used in the gem5 simulator. The hardware platform is configured via the debug tile from a host PC. The debugging interfaces in this tile allow the developer to reset hardware components, enable clock signals, set configuration registers, and upload the compiled binaries to the main memory.

3.2.2 M³ OPERATING SYSTEM

After describing the hardware, this section explains how an OS, called M³, can make use of the unique hardware architecture. In particular, it is explained how the TCU is used by M³ to enforce security policies.

The most critical part of an OS is its kernel, which is the component that must execute at the highest privilege level in order to assign and revoke access permissions to all other software running on the system. M³ is a microkernel-based OS and therefore has a very small and less complex kernel compared

to a monolithic kernel such as Linux. The latter type of kernel can be modular from a software-engineering perspective. However, at run time, all OS-related functionality will be within the same address space without isolation between the individual subsystems. In contrast, a microkernel like the M³ kernel is smaller, because it includes only the functionality that is required to configure and enforce access-control policies and isolation, but nothing else. All other functionality of an OS such as device drivers, file systems, network protocols, and even memory management is moved into separate, isolated OS services that run with reduced privileges.

A microkernel-based OS like M³, consisting of multiple isolated components, fits naturally into the tile-based computer architecture previously described. The M³ kernel runs on one dedicated tile with a RISC-V processor. This tile is therefore called the *Kernel Tile*. All other tiles, except those hosting the DDR4 memory controllers, are called *User Tiles*. They run components of the OS and application software. As the TCU sits between each tile and the NoC, the TCU is the only piece of hardware that can provide access to tile-external resources. Software running on the kernel tile and on processor-enabled user tiles must use their tile-local TCU to perform the following operations:

- Send and receive messages to/from another tile
- Read or write memory that is attached to another tile
- Read or write global memory attached to the DDR4 tiles

All these data transfers can only be done via the TCU and across the NoC, as no other communication links exist in the hardware. The TCU will only allow message-based communication or memory transfers, if an *endpoint* specifying the target tile has been configured within the TCU. There are three types of endpoints that enable the operations listed above:

- Send endpoints for sending messages
- Receive endpoints for receiving messages
- Memory endpoints for reading or writing specific regions of memory

The endpoint configuration of a TCU cannot be altered by a user tile itself, but only by the microkernel running on its dedicated kernel tile. The kernel tile is privileged in the sense that it is the only tile from which any TCU can be configured via the NoC. Thus, the microkernel running on the kernel tile is the only component in the system that can establish communication channels or provide access to remote memory. The microkernel therefore decides whether communication or memory access is allowed and can also request rules regarding them such as the maximum message size for communication or the access mode for memory access (read/write). These so-called *security policies* are represented as endpoints in the TCU and enforced by the TCU hardware based on the existence of endpoints and their properties.

The M³ OS includes several system services that enable basic support for application programs running on the platform. The most important ones are:

- **Root:** A resource manager for launching new applications with the appropriate access permissions
- **Pager:** A service to provide the memory needed to store code and data of running applications (and other OS services)
- **M3fs:** A file system service

These OS services originally required their own dedicated tiles in addition to tiles that run application programs. But enhancements made to the platform now enable them to share processor tiles (see Section 3.3).

Application programs running on M³ must be written for that OS. Software that has originally been written for other systems such as Linux can be ported (e.g., based on the existing POSIX-emulation layer in M³), but the engineering effort varies from case to case. Furthermore, as a research OS, M³ does not support all features of a mature general-purpose OS such as Linux. So not all use cases may be supported at this time. In iNGENIOUS, an application for reading data from a vibration sensor will be demonstrated in the Transport UC. This application accesses the sensor via the UART interface attached to one of the tiles. Sensor status and raw data may then be forwarded to a cloud server for further analysis according to the scenario envisioned in the Transport UC of the project. Another application implementing a signal-processing chain for software-defined radio, relevant to iNGENIOUS Factory UC, is described in Section 3.5.

In addition to software programs running on user tiles, the M³ hardware architecture and OS also support user tiles that include special-purpose, fixed-function accelerators. The simulated version of the hardware platform that is available in gem5 includes accelerator tiles for performing fast-Fourier transforms (FFT) and related operations, as well as an accelerator for certain cryptographic operations (see Section 3.3). At the time of writing, these accelerators are not available in the FPGA version of the hardware platform. Accelerator tiles are integrated into the platform in the same way as general-purpose processor cores: They are connected to the NoC via their own TCU and these TCUs are configured in the same way as processor-based user tiles, including endpoints for cross-tile data exchange.

M³ uses the term *Activity* to designate both a program running on a user tile and data processing on a fixed-function accelerator. In the former case, an activity is mostly equivalent to a process in other OSes. In the latter case, it represents an application-specific context through which an accelerator provides a service to other applications.

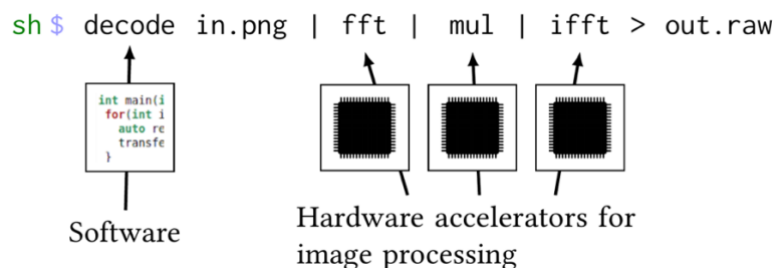


Figure 3.5: M³x showing image-processing accelerators that are usable from the shell

The advantage of the uniform handling of general-purpose cores and accelerators in M³ has been studied in our previous work M³x [15]. In this work, we used edge detection on image files as a use case. The input image is stored in the file system and the output image should be stored as raw pixel data for later post-processing. The actual image processing can be done faster and more energy-efficient on specific hardware accelerators (e.g., using FFT convolution). However, accessing files from accelerators or pipeline accelerators with software is challenging. To achieve this, M³x builds upon the uniform interface that the TCUs provide for all tiles and the unification of processes on general-purposes cores and working logic on accelerators into activities. M³x was therefore able to connect accelerators directly with OS services or other accelerators. This ability enabled the usage of accelerators directly from the shell (see Figure 3.5) and also allowed accelerators to execute “autonomously”, that is, without continuous assistance by the OS. M³x demonstrated that this can lead to significant speedups and reduced CPU utilizations.

Outsourcing access-control enforcement to the TCUs has two advantages. First, access rights of software running on user tiles with general-purpose processors can be policed without involving the M³ microkernel, thereby increasing efficiency of communication control. Second, any other kind of user tiles with a hardware accelerator or I/O device can be connected to the NoC and managed in the same way. Accelerators and I/O devices become “1st-class citizens” that are subject to a unified management and access-control regime supporting both software and hardware components.

3.3 Enhancements to the Base Platform

Within task T3.2, BI extended the base platform shown in Figure 3.3 with:

- 1) Ethernet support via an on-chip network interface card (NIC) for up to four tiles,
- 2) Core-sharing capabilities by virtualizing the TCU (vTCU) and introducing a software component called TileMux responsible for core multiplexing [16], and
- 3) A design of a root of trust (RoT) to enable security features (see Section 3.4.1).

The extended platform is presented in Figure 3.6.

Ethernet is supported by an integrated NIC in a selected RISC-V tile. The NIC provides the necessary hardware basis for our network stack running on the RISC-V core. The on-chip NIC is built out of Xilinx Ethernet IP blocks including an AXI-based Ethernet subsystem with the Ethernet MAC, which is connected to the external Ethernet PHY. In addition, an AXI DMA block is also connected to the core’s cache-coherent system bus to access data of sent and received Ethernet packets. The RISC-V core has interrupt-driven access to the AXI DMA.

Each RISC-V core can be configured with transmit and receive logic for a UART (RS232) interface. The software running on the RISC-V core is used to control the interface (e.g., set baud rate) and access the data. The integrated Ethernet and the UART interface allow to connect external devices and sensors to enable the iNGENIOUS use cases.

Most of the tiles are identical and software-based activities can be placed on arbitrary tiles as needed. However, the TCP/IP network stack must be assigned to a tile with a NIC to enable Ethernet support. Likewise, a service that communicates to an external sensor via the UART interface has to be placed on the UART-enabled tile to use this interface (i.e., the one shown in the lower-left corner of Figure 3.6).

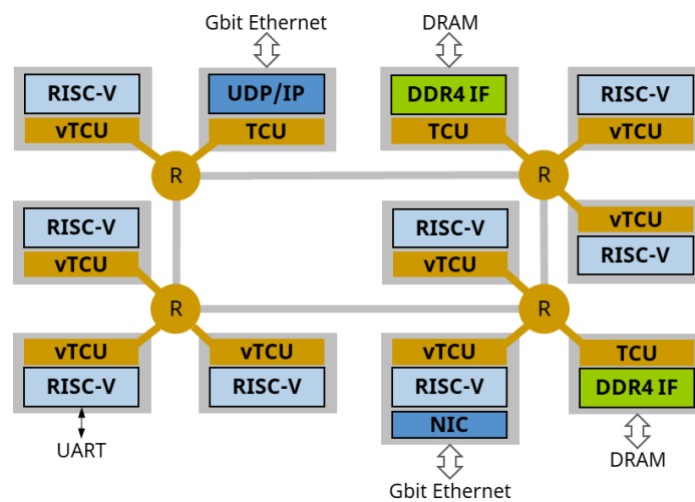


Figure 3.6: M³ base platform extended with an on-chip network interface card (NIC), virtualized TCUs (vTCU), and a UART interface

Besides the mentioned TCU features such as memory transfers and message passing, BI added the ability to multiplex a tile among multiple applications. Tile multiplexing is challenging in the M³ architecture for two reasons. At first, tiles can communicate directly with each other after the OS has set up a communication channel without involving the OS again. This so called “fast-path communication” is in conflict with tile multiplexing, because the multiplexing needs to be performed by the OS, whereas fast-path communication avoids the OS during communication. Second, strong isolation between tiles, as provided by M³, is challenging, because a tile cannot save and restore the TCU state in order to switch between applications on this tile. This is because restoring TCU state also allows to create arbitrary communication channels to other tiles and thereby breaks isolation between tiles.

To overcome these challenges, we virtualized the TCU and introduced a tile-local multiplexer called TileMux on every tile [16]. The virtualized TCU (vTCU) stores the state of all applications on this tile and therefore no save and restore of this state is required. TileMux is a small software component that runs in the privileged CPU mode on this tile and uses additional features in the vTCU to securely multiplex the tile (endpoints and memory) among multiple

applications. For example, TileMux ensures that each application can only access its own endpoints. Therefore, TileMux is similar to a traditional microkernel, but is per tile and has no permissions beyond its own tile. When evaluating the used FPGA resources of the vTCU, an increase of only 6% can be observed compared to the basic TCU implementation [16].

From an application's point of view, there are multiple security goals that the hardware and the OS have to meet to ensure secure program execution and data protection:

- **Confidentiality:** internal state and other data processed by a program should not be visible to unrelated programs running on the same system.
- **Integrity:** Program code, internal state, and other data processed by a program must not be tampered with or rolled back to an older version by unrelated programs running on the same system.
- **Availability:** Program code and data shall be accessible when a program is supposed to run.

When access rights are properly configured, these security goals can be met for activities that are running programs locally inside a system-on-chip (SoC) realization of the M³ platform. This also applies to multi-component applications that consist of separate parts that run as individual programs within the SoC.

However, since the IoT is a distributed system, cooperation between multiple components running on several devices must also be considered. Thus, in addition to the general improvements of the M³ platform, BI worked towards support for end-to-end secure and trustworthy cooperation between devices. These contributions are discussed in the following section.

3.4 Security Building Blocks

In a very common IoT scenario, software components could run on an edge device and a cloud server. The following system requirements are derived from the necessity to support secure cooperation between distributed computers:

- **Identifiability** of the platform (e.g., “tile-based SoC capable of running M³”) using remote attestation: The spectrum ranges from being able to identify a single device (i.e., “exactly this SoC”) to a more anonymous identification that only assures that the device has certain properties (e.g., “one of these SoCs, not important which one exactly”). Some use cases may also require something in between, like for example an assurance that it is “any SoC deployed and operated by X in location Y”.
- **Secure communication** between software components: Parts of a distributed application must be able to cooperate without interference, regardless of their location (e.g., “same SoC” or “SoC in edge server and Linux process on cloud server”). Thus, they require a secure communication channel for exchanging information as part of their cooperation.

The ability to identify both hardware and software of a remote device enables verifiable trust between communication partners. Based on this capability, system designers can build a trustworthy IoT – or any other security-critical distributed system. A *Root-of-Trust (RoT)* integrated into the hardware is an essential building block to support this capability. It will be discussed in Sections 3.4.1 through 3.4.4, including the OS support in M³ that will exploit this RoT and the implementation status.

3.4.1 MINIMAL ROOT OF TRUST

The main functionality of an RoT is that it can 1) create an unforgeable signature over 2) accurate information that describes the state of the device that has the RoT built into its hardware. And 3), this signature and information can be verified using cryptographic means on another, remote device. Thus, in a nutshell, an RoT needs a signature key, which is known to exist only in a specific device. When used for *Remote Attestation*, the information signed using the RoT's *Attestation key* is typically an *Attestation Report* of the software that is currently running on the device. Based on the knowledge of the key (“can only exist in this one device”) and the reported software configuration (e.g., “application X running on M³, which will securely isolate X from any attacks”) one can establish trust in the remote device.

In iNGENIOUS, BI designed a minimal RoT for the tile-based hardware architecture that has been co-designed with M³. In this design, a dedicated tile is added that hides the signature key in hardware. The key itself is not accessible to any software on other tiles, but it can be used to sign data by software running on a RISC-V core that is also part of the tile. In an M³ context, this software implements an *RoT service*. The purpose of that service is to allow other services of the M³ OS and applications to 1) tell the RoT what software is about to be started on the device and 2) request attestation reports about the previously started programs at a later time. Such attestation reports are typically needed when answering to a remote attestation request received from a remote device. The software implementation of the RoT service shall avoid unnecessary complexity (to minimize the risk of security-critical bugs) or interfaces to other components that are not related to the purpose of this service.

By design, the key hidden in the hardware cannot be changed. Thus, it is particularly important to minimize the risk of compromising this *Long-term Platform Key* and to do that, the RoT design minimizes the time window in which it can be used to by the RoT service. It can only be used to create signatures right after powering up until the RoT service deliberately disables access to it by means of a hardware lockout mechanism. This lockout operation cannot be undone until the next power cycle. The RoT service uses the brief time window to sign one or multiple software-managed signature keys, which are held in tile-local, volatile memory that is only accessible to the RoT service itself. By means of this signature, the second type of key, the *Short-term Attestation Key*, is cryptographically linked to the long-term key that is hidden in the hardware. It is the short-term key that is actually used to sign the attestation reports.

The two-level signature scheme ensures maximum protection of the long-term key by minimizing its already limited exposure (only the RoT service can use it) and it ensures that a new short-term attestation key can be created (e.g., in case the device got compromised in some way). The long-term key is "future proof" as it issues hash-based signatures, which are considered secure even against brute-force attacks using quantum computers. The short-term keys are elliptic-curve keys, which are considered state of the art for signing operations.

At the time of writing, the design as briefly described above has been completed. As a first step towards an implementation, a dedicated hardware accelerator for the Secure Hash Algorithm 3 (SHA-3, also known as Keccak [17]) has been designed and implemented in the gem5-simulated version of the M³ hardware. Like the RoT design described in this section, this accelerator tile consists of a hardware implementation of the SHA-3 algorithm, which is controlled by a *Hash Multiplexer* service running on a RISC-V core that is also part of the dedicated SHA-3 accelerator tile. A version of the SHA-3 accelerator will be extended into the RoT as described above.

A complete prototype implementation of the RoT hardware part is expected to be available in the gem5 simulator sometime after the project. The software part, consisting of the RoT service, is expected to be compatible with an FPGA implementation of the RoT that will be realized once the gem5 version has been developed and tested.

In the M³ platform, the RoT will enable several security-related platform capabilities that are beneficial to iNGENIOUS use cases:

- **Secure launch:** The code and selected configuration data of a program started on the M³ platform is reported to the RoT service for later inclusion in an attestation report.
- **Remote attestation:** One or multiple application programs and the M³ OS services they depend on are included in an attestation report generated and signed by the RoT.
- **Secure software updates:** It is ensured that only legitimate software updates will replace an already installed version of a particular program.
- **Secure data storage:** By including an encryption key in the RoT, the RoT service can be enhanced to encrypt data at rest in an M³-based device.

Secure launch and the OS support necessary to enable it in M³ will be discussed in the following Sections 3.4.2 and 3.4.3.

BI integrated remote attestation with the state-of-the-art communication protocol Transport Layer Security (TLS). The current implementation is capable of attesting the software state of a Linux cloud server, provided that server uses a Trusted Platform Module (TPM) [18] to measure its firmware, boot loader, and OS according to specifications released by the Trusted Computing Group [19]. More details of this activity are reported in iNGENIOUS deliverable **D5.3 Final iNGENIOUS data management platform** [20].

Remote attestation can also be used to solve the version compatibility and code-and-data integrity problem during software updates.

3.4.2 TRUSTED EXECUTION ENVIRONMENT IN M³

Section 3.3 of iNGENIOUS deliverable *D3.1 Limitations and improvement axis for the communication of IoT devices* [9] explained in detail why cooperating components of a distributed multi-component application must trust each other. The RoT described in the preceding section is an essential building block for verifying cross-device trustworthiness. Its signatures provide a cryptographic proof that software running on a remote device does indeed meet certain security goals such as integrity of its codebase and confidentiality of data being processed.

To express, enforce, and report these properties, the attesting device must be able to protect running programs and the data they process through isolation. The hardware and software features of the platform that enable this protection are called a *Trusted Execution Environment (TEE)*. In a nutshell, a TEE allows execution of a program and keeps this program's code and its state isolated from whatever is happening outside the TEE. To enable the program in the TEE to interact with the outside when needed, communication channels are configured for the TEE to provide a controlled way for input and output. In an M³-based platform, any tile that includes a processor that can run software can be considered a TEE. The tile's TCU is used to enforce communication control for the activity running in this TEE.

Communication channels are considered critical configuration of a TEE. When a set of cooperating components runs locally as activities within the same M³-based SoC, all communication links are implicitly trusted. The attacker model assumes that inter-tile communication is secure, because all messages are sent through the trusted TCUs and NoC inside the same SoC. However, when components communicate with remote TEEs, additional protection is needed. In the IoT usage scenarios covered by the iNGENIOUS use cases, TLS integrated with remote attestation is used to secure off-chip communication. Note that all communication channels established between a set of cooperating TEEs must be listed in the remote-attestation report; the TEEs of all system services required by a program are also considered "cooperators" and must therefore be listed as well (i.e., not just all parts of a distributed application, but everything on which the application depends to fulfil its purpose and meet its security goals).

3.4.3 M³ SUPPORT FOR SECURE EXECUTION OF APPLICATIONS

Figure 3.7 shows all system-level software components that allow M³ to securely load a program into memory, report its codebase to the RoT for remote attestation, and start it with all communication channels.

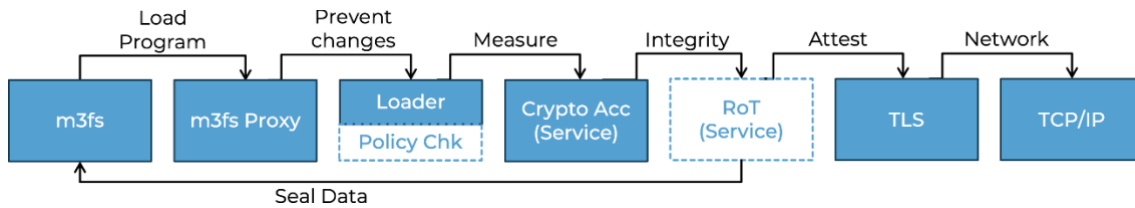


Figure 3.7: M³ software components required for secure launch of applications, remote attestation, and secure cross-device communication

The components interaction with each other as follows:

- **M3fs:** The M³ file system stores the program's binary file and potentially additional configuration files associated with the program. M3fs is trusted only with regard to availability, but not integrity or confidentiality, because its backing store may not be sufficiently protected.
- **M3fs proxy:** The M3fs proxy offers the same interface as M3fs, for which it acts as an interposition layer. Its purpose is to read the content of security-critical files from the untrusted M3fs into secure memory, where the contents of that file can no longer be changed. For program binaries and configuration files associated with program, the secure memory can be regarded part of the TEE.
- **Loader:** The loader allocates a user tile and then sets up memory areas and communication channels such that the program can be started. Once the program is running, it becomes an activity running on the chosen tile, executing the code that has been copied to the secure memory by M3fs proxy.
- **Crypto Accelerator:** The crypto accelerator computes a hash over the program's codebase and configuration files, if any. Its purpose is to efficiently create a measure of the integrity of the program before it is started. The measurement, in the form of a cryptographic hash, is reported to the RoT.
- **Root-of-Trust (RoT):** The RoT will issue a signature over the integrity information that has been reported for a given program. If extended as suggested, the RoT can also be used to sign and encrypt arbitrary data for storage in M3fs.

In the picture shown in Figure 3.7, the **Policy Checker** part of the loader and the RoT service are drawn in dashed lines, because their implementation is incomplete at the time of writing. The purpose of the policy checker sub-component is to report communication channels to the RoT, such that they can be included in the attestation report as described in Section 3.4.2. Furthermore, it is planned to implement attestation-based compatibility checks when multiple cooperating programs shall be started as a multi-component application. Only if the compatibility and integrity check of each involved program succeeds, the cooperating programs are started as M³ activities and communication channels are established among them.

Finally, to send an attestation report to the remote device that requested information about a (group of) program's integrity, two more M³ services are needed:

- **Transport Layer Security (TLS):** Within WP5 of iNGENIOUS, BI developed a wrapper library around TLS that integrates remote attestation into the TLS handshake. The purpose of this component is to provide an end-to-end secure communication channel between an activity running inside a TEE in an M³-based device, and another M³ device or cloud server. Details are described in iNGENIOUS deliverable **D5.3 Final iNGENIOUS data management platform** [20].
- **TCP/IP Network Stack:** The TCP/IP network stack runs on a dedicated tile with an Ethernet NIC. If communication is protected by the TLS service, as envisioned for all IoT scenarios in iNGENIOUS, the network stack needs not to be trusted except for availability.

3.4.4 IMPLEMENTATION STATUS

The overall state of BI's M³ platform is visualized in Figure 3.8. The picture contains all software components described above and the state of the hardware implementation in both the gem5 hardware simulator and the FPGA development boards. Components shown as filled boxes are fully implemented at the time of writing, whereas the implementation of building blocks drawn in dashed lines will be worked on in the remainder of the project and beyond. To provide a complete overview of the current status, the finished platform enhancements described in Section 3.3 are included as well (i.e., core sharing for increased hardware utilization and UART support for connecting sensors).

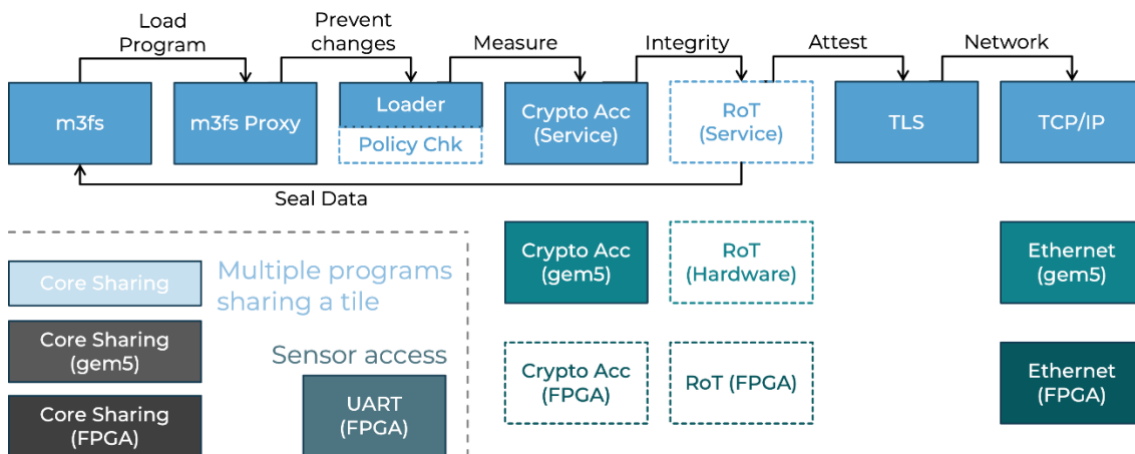


Figure 3.8: Overview and implementation status of M³ hardware and components

The integration of remote attestation with TLS (shown in the top right corner of the figure) is described in detail in iNGENIOUS deliverable **D5.3 Final iNGENIOUS data management platform** [20] and in a publication [21]. As work on some security building blocks related to the root-of-trust and remote attestation is still progressing at the time of writing, additional information will be reported as part of the evaluation activities in WP6 of the project in deliverables **D6.2 PoC development, platform and test-bed integration** [22] and **D6.3 Final iNGENIOUS data management platform** [6].

3.5 Flexible PHY Implementation on M³

As a conclusion of the M³ platform overview, this subsection describes an application example. Within the INGENIOUS Factory use case, TU Dresden (TUD) implemented PHY modules that perform radio signal processing steps such as bit-to-symbol mapper, cyclic redundancy check (CRC), scrambler, interleaver, channel encoder, and waveform modulation. These modules are parts of a multi-component application running as a set of cooperating M³ activities.

Within the scope of heterogeneous IoT networks, a flexible physical layer (PHY) chain that can attain conflicting requirements is an important component for providing wireless connectivity. Moreover, private wireless networks can benefit from optimized configurations for specific scenarios or tasks. However, a flexible PHY implementation on general-purpose processors limits the achievable performance in terms of latency and throughput, thus restricting certain use cases.

Within the INGENIOUS framework, the flexible PHY-MAC development is divided in two distinct platforms named Type-1 and Type-2. The Type-1 implementation has been developed with the National Instruments (NI) Software Defined Radio (SDR) platforms and consists of a host implementation for control and an FPGA implementation for real-time signal processing. The Type-2 implementation has been developed for BI's M³ platform, which is being designed with focus on security. It is important to point out that a Type-1 receiver can operate with a Type-2 transmitter, since the PHY protocol can be configured identically. Therefore, only the PHY modules from the transmitter side are currently under development.

The PHY modules have been written with the Rust programming language. Rust is a general-purpose programming language developed for attaining safer concurrency when compared to the popular C++ language. Moreover, Rust is particularly suited for running on devices with little computational resources, since it is a programming language with very little run-time overhead.

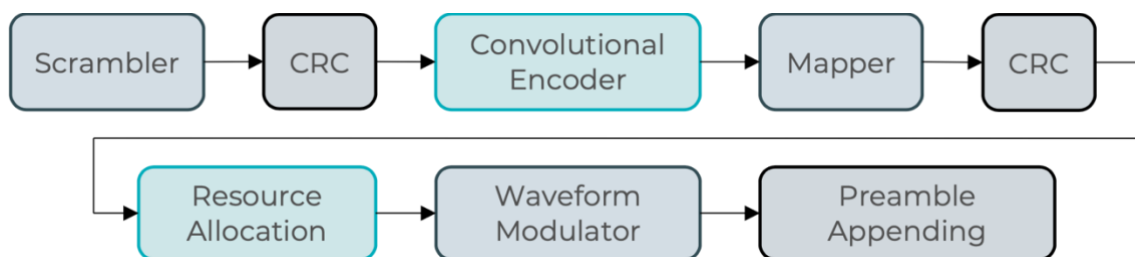


Figure 3.9: Transmitter PHY modules developed for M³

The modules of this software-defined radio application are shown in Figure 3.9. Their roles are as follows:

- The **Scrambler** shuffles the bit stream for avoiding long sequences of zeros or ones, thus allowing the channel coding to be effective.

- **Cyclic Redundancy Check (CRC)** is implemented with 8 or 16 bits in length. This module generates checksums that allow the receiver to check the validity of a received code word.
- The **Convolutional Encoder** generates parity bits from the input bit source by convoluting it with a binary polynomial, referred as generator. This particular implementation uses the generator $[133, 171]_8$, giving a nominal code rate of $3/4$. The constrain length used is 32 bit.
- The **Mapper** translates the coded information bits into complex symbols, i.e., quadrature amplitude modulation (QAM). The current implementation supports 4 and 16 QAM.
- The **Resource Mapper** allocates the subcarriers that will be used for data transmission and pilot transmission. Two cases are currently supported, 64 and 1024 subcarriers assuming orthogonal frequency division multiplexing (OFDM) as waveform modulation scheme. When the available bandwidth is divided into 64 subcarriers, 4 subcarriers are set as pilots, 12 are used as guard band, and 48 are used for transmitting useful data. For 1024 subcarriers, 8 are set as pilots, 128 are used as guard, and 888 carry useful information.
- The **Waveform Modulator** combines linearly the QAM symbols for producing samples that will be transmitted through the wireless channel. The current implementation uses OFDM. Within this block, the cyclic prefix is also added to allow for simple frequency domain equalization on the receiver side.
- **Preamble Appending** will concatenate to the transmit signal a known preamble for synchronization at the receiver side.

The computation of each block in the transmitter chain is executed as one activity running on a user tile with a RISC-V processor in the M³ platform. The modules cooperate in the PHY processing chain by exchanging messages through the TCUs and the NoC. Thus, they distribute the computational load across multiple processing cores while allowing run-time flexibility.

3.6 Exploitation Potential and Use Cases

The hardware/software co-designed platform, comprising the tiled architecture with the TCU and the microkernel-based M³ OS, are highly relevant in all INGENIOUS use cases. The platform is a secure foundation for building IoT devices. However, in the project, the development and evaluation in trials will focus on the Transport UC. When fully implemented, it enhances the state-of-the-art in IoT computer architectures in the following ways:

- M³ provides an isolation-by-design platform for securely running IoT applications in Trusted Execution Environments (TEEs).
- Remote attestation and TLS integration enable end-to-end secure identification, integrity reporting, and cooperation between multiple devices in IoT communication. Sensor data produced by vibro-acoustic

sensors (see Chapter 2) can be securely reported to a central control centre to assess and detect possible defects in rail carriages.

- Secure data storage on device using the Root-of-trust (RoT) enables buffering of sensor data on an M³-based gateway device until communication with the control centre becomes possible.
- Detection of corrupt on-device software enables trustworthy cross-device collaboration in the edge-cloud continuum.
- Secure and robust software updates thanks to trustworthy information about remote software versions.
- Application workloads such as flexible PHY processing can utilize computational resources of the platform in strict isolation from other software activities that may share the M³-based SoC.

Despite the focus on innovation in the Transport and Factory use cases of the project, the presented M³ hardware/software co-design is a suitable basis for building secure IoT devices for many more use cases. In the wider supply-chain domain, signed attestation reports about the security of devices and the data they produce could be recorded in DLT (distributed ledger technology) networks employed in other iNGENIOUS use cases. Beyond that, networked devices of many more distributed-computing scenarios could benefit from the M³ platform.

The following table states the use case coverage of the technology components described in this chapter according to demonstration and evaluation activities in WP6 of the project, and summarizes its exploitation potential. Evaluation results will be reported on in iNGENIOUS deliverable **D6.3 Final iNGENIOUS data management platform** [6]. Details on exploitation plans will be provided on in iNGENIOUS deliverable **D7.3 Final dissemination, standardisation and exploitation** [3].

Component: Ultra-safe IoT Compute Platform

Relevant for use cases: Factory (Flexible PHY/MAC only), Transport (M³ platform)

Exploitation Potential:

- **BI**, as an academic research institute, will exploit iNGENIOUS-funded developments of its platform for teaching and as a basis for future research and collaborations. Parts of the BI compute hardware and operating system platform are already open source and therefore available to academia and industry; more components shall become open source as research and development progresses.
- **TUD:** The PHY modules coded by TUD carry out the radio signal processing tasks that can be executed in the M³ platform. This collaboration between BI and TUD allows for research and experimentation of hardware and communications co-design aiming at secure wireless communications.

4 Conclusions

This deliverable presented the activities of the iNGENIOUS partners towards building secure, private, and more efficient hardware solutions for IoT devices.

In Chapter 2, the document covered the activities towards smart and efficient sensors, which are needed for many types of edge devices. The work focused on requirements and the approach chosen for the iNGENIOUS Transport use case. The discussion covered the trade-off between costs for local computation on the edge device and offloading communication to cloud resources at the expense of energy required for communication. Data sampling and analysis have been discussed with a focus on vibro-acoustic sensors for railcar health monitoring, as required by the Transport UC. The chapter closes with a summary of exploitation potential.

The lessons learned from the Rail-Health Transport UC was that not only technological aspects, but also financial business case aspects are extremely challenging. While preventive maintenance has many indirect benefits such as productivity improvements and reduced safety risks, in the end it is only the reduced maintenance costs that really matter. And if regular maintenance cycles are very long – as is the case for rail health – then the return on investment also becomes very long. For this reason the cost targets were dropped to half and the operation lifetime expectations increased from 12 to 24 or 30 years. This certainly has technology and reliability impacts on the product to be developed. Rather than giving up, the iNGENIOUS approach of focusing on ultra-low power operation opens the door to cost effective low-power energy harvesting solutions which could smooth the path to self-powered Micro-Edge IoT Solutions. And that truly will be Next Generation IoT.

Chapter 3 motivated that the embedded computers of IoT devices are of critical importance for the overall security and trustworthiness of the Next-generation Internet of Things. It also made the point that both hardware and software need to be considered to enforce isolation. The work shows that enforcement of this design principle is one of the most important security measures to ensure security in the presence of system-software and hardware complexity. The M³ microkernel-based operating system and the tile-based computer architecture have been designed with that mindset. The M³ hardware/software co-design is being enhanced within the project with new capabilities, including first steps towards a minimal hardware root-of-trust. This root-of-trust enables the concept of remote attestation, which can be the technical foundation of verifiable trust and secure communication between IoT device and cloud servers, as well as an enabler for secure and robust software updates. Both the basic platform and the still ongoing enhancement have been described in detail. Finally, it was outlined how they fit the needs of supply-chain use cases within iNGENIOUS. However, the application of the M³ approach can benefit many more IoT use cases.

References

- [1] “iNGENIOUS deliverable D2.1: Use cases, KPIs and requirements,” 2021.
- [2] “iNGENIOUS deliverable D2.4: System and architecture integration (Final),” 2022.
- [3] “iNGENIOUS deliverable D7.3: Final dissemination, standardisation and exploitation,” 2022.
- [4] Apple, Inc., “AirTag - Apple,” Apple, Inc., [Online]. Available: <https://www.apple.com/airtag/>. [Accessed 31 03 2022].
- [5] Wikipedia, “Design of experiments,” [Online]. Available: https://en.wikipedia.org/wiki/Design_of_experiments. [Accessed 28 March 2021].
- [6] “iNGENIOUS deliverable D6.3: Final iNGENIOUS data management platform,” 2022.
- [7] L. Vilanova, S. Bergman, T. Miemietz, M. Hille, N. Asmussen, M. Roitzsch, H. Härtig and M. Silberstein, “Slashing the Disaggregation Tax in Heterogeneous Data Centers with FractOS,” in *EuroSys '22: Fifteenth European Conference on Computer Systems*, Rennes, France, 2022.
- [8] S. Biggs, D. Lee and G. Heiser, “The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-Based Designs Improve Security,” *Proceedings of the 9th Asia-Pacific Workshop on Systems (APSys'18)*, 2018.
- [9] “iNGENIOUS deliverable D3.1: Limitations and improvement axis for the communication of IoT devices,” 2021.
- [10] N. Asmussen, M. Völz, B. Nöthen, H. Härtig and G. Fettweis, “M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores,” *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*, pp. 189-203, March 2016.
- [11] “The Gem5 Simulator System,” [Online]. Available: <https://www.gem5.org>. [Accessed 22 March 2022].
- [12] K. Asanovic, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim and J. Koenig, “The rocket chip generator,” in *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.
- [13] J. Zhao, B. Korpan, A. Gonzales and K. Asanovic, “SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine,” in *Fourth Workshop on Computer Architecture Research with RISC-V*, 2020.
- [14] S. Haas and N. Asmussen, “A Trusted Communication Unit for Secure Tiled Hardware Architectures,” *29th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2022.
- [15] N. Asmussen, M. Roitzsch and H. Härtig, “M3x: Autonomous Accelerators via Context-Enabled Fast-Path Communication,” in *2019 USENIX Annual Technical Conference (ATC'19)*, 2019.
- [16] N. Asmussen, S. Haas, C. Weinhold, T. Miemietz and M. Roitzsch, “Efficient and Scalable Core Multiplexing with M³v,” in *27th ACM*



International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22), Lausanne, 2022.

- [17] National Institute of Standards and Technology (NIST), “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” 2015.
- [18] “The Trusted Computing Group - Trusted Platform Module (TPM),” [Online]. Available: <https://trustedcomputinggroup.org/work-groups/trusted-platform-module/>. [Accessed 22 March 2022].
- [19] “The Trusted Computing Group,” [Online]. Available: <https://trustedcomputinggroup.org/>. [Accessed 22 March 2022].
- [20] “iNGENIOUS deliverable D5.3: Final iNGENIOUS data management platform,” 2023.
- [21] R. Walther, C. Weinhold and M. Roitzsch, “RATLS: Integrating Transport Layer Security with Remote Attestation,” *Applied Cryptography and Network Security Workshops, 4th Workshop on Cloud Security and Privacy (Cloud S&P)*, pp. 361-379, June 2022.
- [22] “iNGENIOUS deliverable D6.2: PoC development, platform and test-bed integration,” 2023.
- [23] “iNGENIOUS deliverable D5.1: Key technologies for IoT data management benchmark,” 2021.
- [24] “iNGENIOUS deliverable D4.1: Multi-technologies network for IoT,” 2021.

