D4.1. Specification of the Platform & Specific Security Solutions

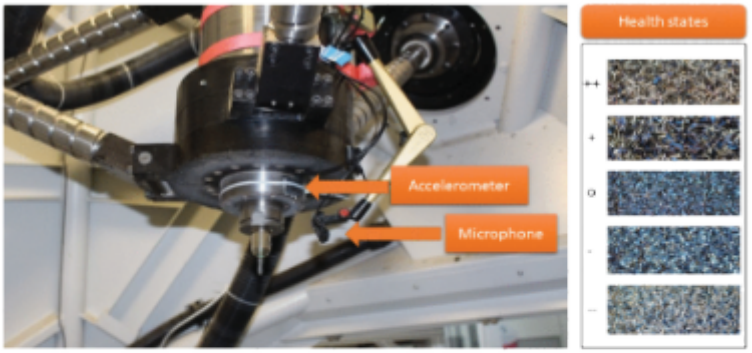
This document serves as deliverable **D4.1** in the CORNET-funded **Trusted IoT project**.

This deliverable includes the overall framework architecture to define the different components, which meet these requirements and define the intercommunication among these components in order to address all the requirements in an integrated way.

UC 1: Coarse-Grained Reconfigurable Architectures (CGRAs) (URO)

Coarse-Grained Reconfigurable Arrays (CGRAs) represent a significant advancement in reconfigurable computing architectures, which consist of coarse computational units and interconnections. These arrays offer an exceptional blend of flexibility and high performance, which is particularly suited for specific applications that require dynamic configuration. Unlike traditional architectures, CGRAs boast a dataflow-oriented design that excels in parallel processing and complex computations, making them highly applicable to the needs of Industry 4.0.

In the realm of Industry 4.0, CGRAs could play a critical role by supporting real-time data analytics and machine learning, integral components of smart manufacturing and the integration of Industrial Internet of Things (Industrial IoT). The adaptability and powerful computing capabilities of CGRAs fulfil the stringent requirements of modern industrial applications. Figure 1.1. provides an example of a use case. It shows a milling machine augmented with sensors. The sensors can be utilised to predict the failure of the tool, which will likely result in discarding of the part machined at the time of failure. Replacing the tool before this happens is favourable. CGRAs are well suited to the algorithms commonly associated for processing the sensor data.

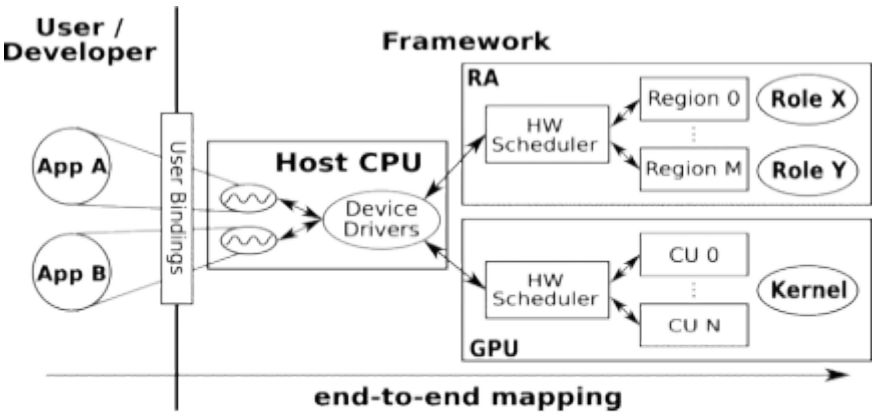


From: P. F. Suawa, A. Halbinger, M. Jongmanns and M. Reichenbach, "Noise-Robust Machine Learning Models for Predictive Maintenance Applications," in IEEE Sensors Journal, vol. 23, no. 13, pp. 15081-15092, 1 July, 2023, doi: 10.1109/JSEN.2023.3273458 (used with permission)

Figure 1.1: Example for a use case in Industry 4.0 (Predictive Maintenance for Milling Machine)

Addressing security, a paramount concern in Industry 4.0, CGRAs face similar challenges to FPGAs, such as ensuring component separation, particularly from different manufacturers. Emphasising a robust "security-by-design" approach is crucial in safeguarding networked systems against vulnerabilities.

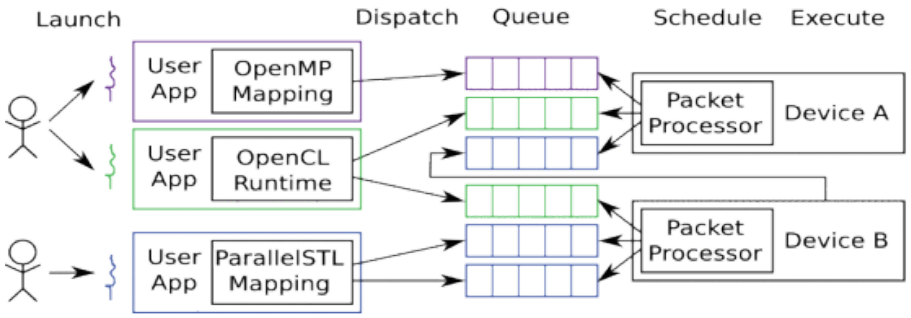
To this end we extend the Heterogeneous Reconfigurable Architectures (HERA) [UC1-1] methodology, which focuses on enhancing separating workloads and improving the security perspective. HERA aims, among other goals, to bolster security in systems that utilise reconfigurable architectures. An overview of the HERA concept can be found in Figure 1.2.



From: P. Holzinger and M. Reichenbach, "The HERA Methodology: Reconfigurable Logic in General-Purpose Computing," in IEEE Access, vol. 9, pp. 147212-147236, 2021, doi: 10.1109/ACCESS.2021.3123874 (Used with permission)

Figure 1.2: Overview of the HERA Methodology

A key component of systems utilising HERA is the packet processor. Its role in CGRAs is crucial to the outlined concept as it handles scheduling, dispatch, resource allocation optimization and load management. This component is essential for managing CGRAs' dynamic configuration affecting different security relevant mechanisms, such as separation of concerns, memory safety and side channels. The role of the packet processor is depicted in Figure 1.3. The specific solution in the TrustedIoT project will extend the HERA methodology to support CGRAs by providing a packet processor specific to CGRAs.



From: P. Holzinger and M. Reichenbach, "The HERA Methodology: Reconfigurable Logic in General-Purpose Computing," in IEEE Access, vol. 9, pp. 147212-147236, 2021, doi: 10.1109/ACCESS.2021.3123874 (Used with permission)

Figure 1.3: Packet Processor

The intended platform to be supported is the VCGRA [UC1-2]. It is an overlay architecture for FPGAs. This eases the implementation and integration of CGRAs as it can be used in existing physical systems.

Bibliography

[UC1-1]	Holzinger, P., & Reichenbach, M. (2021). The HERA Methodology: Reconfigurable Logic in General-Purpose Computing. IEEE Access, 9, 147212-147236. https://dx.doi.org/10.1109/ACCESS.2021.3123874
[UC1-2]	Werner, A., Fricke, F., Shahin, K., Werner, F., & Hübner, M. (2019, March). Automatic toolflow for VCGRA generation to enable CGRA evaluation for arithmetic algorithms. In International Symposium on Applied Reconfigurable Computing (pp. 277-291). Cham: Springer International Publishing.

UC 2: Ultra Low-powered FPGA (TUD)

Mobile robots that operate in untrusted environments must be protected against unauthorised access. This includes FPGAs in particular, which support mobile robots, e.g., with navigation. The use of FPGA-based Systems-on-Chip (SoC) in mobile robots is advantageous as hardware accelerators speed up the execution of computationally intensive processes, e.g. in the field of image processing, and thus reduce the workload on the processor. Further, the low power consumption inherent in FPGAs is essential for battery-operated robots in particular.

This document describes the specification of the robotic platform as well as the security features that were implemented in order to protect hardware accelerators.

Robcar2 - Mobile Robot Testbed

The mobile robot platform testbed for the Trusted-IoT project, the Robcar2 (as seen in Fig. 2.1), was implemented. It utilizes the KR260 development board as the central component. This setup aligns with the objectives of WP4, incorporating ultra-low-powered FPGA technologies alongside hardware and software optimizations for maximum energy efficiency. It is a four-wheeled skid-drive robot with omni wheels that allows movements in any direction.

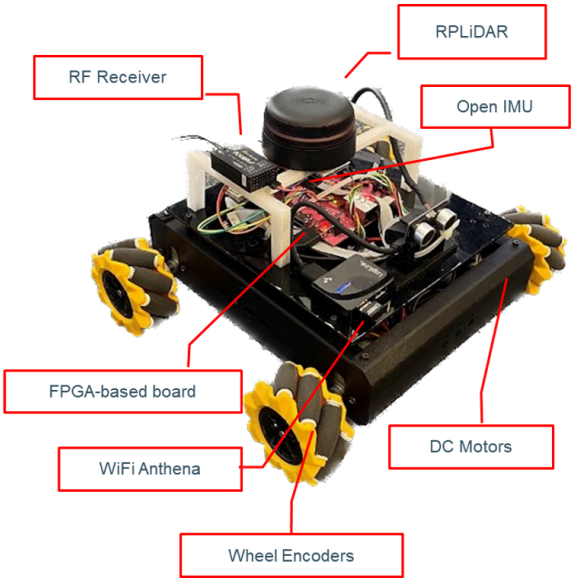


Figure 2.1: Robcar2. The mobile robot testbed for TrustedIoT

The platform consists of the following components:

1. **Ultra low-power FPGA device board (KR260):** Serving as the platform's core, the KR260 Dev board hosts various extensions and interfaces for seamless integration of actuators, sensors, and processing units.

- **Actuators Interface Extension:** This facilitates control over the DC motors. The motors' shafts are linked to omnidirectional wheels that can be driven with full force in any order, allowing the robot to even slide laterally.
- **Sensor Interface Extension:** Enables perception and environment sensing through various sensors, including:
    - Camera/LiDAR: For visual perception and mapping capabilities for spatial awareness.
    - Inertial Measurement Unit (IMU): For motion tracking and orientation estimation.
    - Wheel Encoders: Providing feedback on wheel rotation for odometry and navigation.

2. **Processing Architecture:** For computation and decision-making tasks, the processing architecture comprises:
    - **OS / Middleware:** Provides a robust software framework for managing system operations, communication, and data processing. Fig. 2.1 expands on this and provides a better overview.
    - **Perception Accelerators:** Optimising perception algorithms for efficient execution on the FPGA, enhancing real-time responsiveness and accuracy.
    - **Control Accelerators:** Improving control algorithms for precise actuation and navigation, ensuring smooth and reliable robot operation.
    - **Interconnect:** Facilitating seamless communication and data exchange between components, ensuring coordinated functionality.
3. **Wireless Communication:** The platform supports RF and Wi-Fi communication protocols, enabling remote control, data exchange, and connectivity and ensuring flexibility and adaptability to diverse environments.

Trusted OS and Middleware for Mobile Robots

The developed mobile robot implements micro-ROS [UC2-2] as middleware to perform local localisation and mapping tasks at the edge. In the context of Trusted-IoT, micro-ROS bridges the gap between the resource-constrained embedded systems and the rich functionality offered by the Robot Operating System (ROS). With its comprehensive set of ready-to-use basic functions encapsulated in the ROS robotics stack, ROS is establishing itself as the de facto standard for industrial service robots. This allows the project to utilise the benefits of the ROS ecosystem while taking into account the limitations of a mobile robot operating at the edge. By utilising micro-ROS, the project aligns with industry standards and promotes interoperability and collaboration within the broader robotics field.

Additionally, the mobile robot could benefit from the employment of a hypervisor. Hypervisors isolate software tasks in Virtual Machines (VM) from each other and thus ensure that errors or security breaches in one VM do not affect other VMs. We investigated the employment of L4Re [UC2-1], an operating system framework based on the Fiasco.OC microkernel. The following overview shows the software architecture of a possible micro-ROS-based application for TrustedIoT (Fig. 2.2):

- **RTOS partitioning**: FreeRTOS is used as a real-time task manager for critical operations such as Lidar, MPU and GPS sensor data processing and engine control to ensure timely and deterministic execution.
- **Middleware functionality:** Micro-ROS, which is based on FreeRTOS, can act as middleware to facilitate communication between the various components of the robot system (ROS nodes for sensors, localisation, mapping and planning). It also enables ROS-compatible environments for the development and deployment of robotic applications.
- **Virtualization at system level:** L4Re enables the isolated execution of different components and controls the communication between these components. In this way, L4Re prevents interference between tasks and thus improves the stability of the overall system.

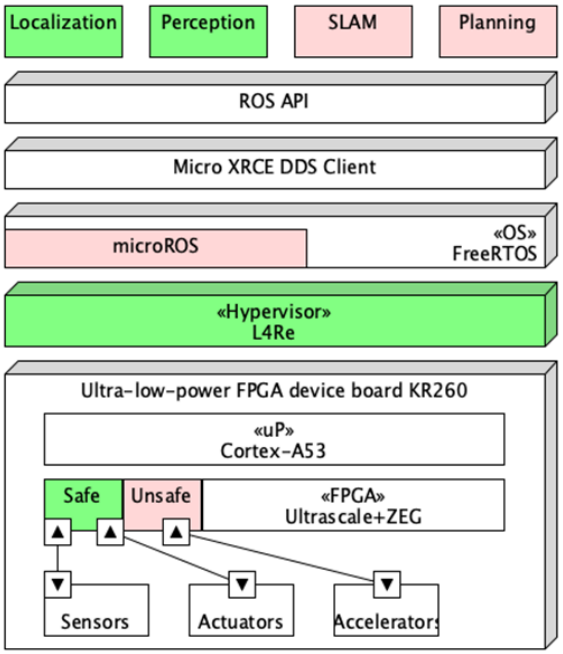


Figure 2.2: Trusted-IoT-Architecture for mobile robots

Security Solution

Hypervisors for FPGAs must ensure that software tasks from different guest operating systems, such as FreeRTOS, cannot access the same hardware accelerators, otherwise they could read or modify the data from tasks in other guest operating systems. Tasks that use Direct Memory Access (DMA) to access hardware accelerators, for example via the AXI stream protocol, can use techniques provided by the operating systems / hypervisors for DMA. For example, L4Re provides the kernel object ‘DMA space’, which represents a virtual address space for DMA. Applications that use hardware accelerators via AXI4-Lite or AXI4-Full bus protocol exchange data in a memory mapped way. L4Re provides the IO server in order to control access rights, e.g., for Memory-Mapped IO (MMIO) regions. However, access rights are static and must be assigned by the IO server at design time. It is

not possible to grant an application exclusive access rights to a hardware accelerator only for a certain period of time and at a later point grant access to another application.

For this reason, hardware-based access and control mechanisms for hardware accelerators have been developed as part of this work package, which enable a flexible memory-mapped access while maintaining the required isolation and task priorities. In the adversary model, we target malicious software tasks that violate the address space of a hardware accelerator that has been assigned to another task. The attacker can falsify data, steal results or interfere with the execution of a hardware accelerator, thus jeopardising the confidentiality, integrity and availability of the corresponding data.

2.2. Comparison with state-of-the-art

Although Xilinx provides a memory protection unit (XMPU) [UC2-3] that partitions memory areas for isolated secure masters, only separate processors can be selected as masters, not tasks that share a processor. Further research work deals with hardware-based control mechanisms for hardware accelerators: Deb Nath et al. [UC2-4] developed a centralised security policy engine in combination with security wrappers for each hardware accelerator. Mbongue et al. [UC2-5] transferred the Mandatory Access Control (MAC) policy, which is implemented in SELinux, to hardware accelerators. Here, a software security server is combined with a hardware controller for each hardware accelerator to check access authorisation. The same authors further discuss the FPGA provisioning model and VM integration for multi-tenant cloud FPGAs [UC2-6]. Elnaggar et al. [UC2-7] target address redirection, where an attacker modifies the memory address to load an incorrect bitstream or redirects the authenticated communication between the applications and the hardware tasks to malicious entities. AmorphOS [UC2-8] encapsulates FPGA logic in morphable tasks and introduces the hull for cross-domain protection. The hull includes memory protection and access mediation. Malik et al. [UC2-9] propose a Safe-and-Secure AXI crossbar that includes mapping tables to support safe resource sharing across multiple master IPs. An orchestrator realised in software dynamically updates the security register configuration.
None of these works deals with the data isolation of hardware accelerators communicating in a memory-mapped way that are to be dynamically assigned to different tasks in a virtualized system.

2.3. Hard- and Software Components

The solution developed as part of the TrustedIoT project uses the security mechanisms of L4Re and supplements them with the following hardware components: a hardware task scheduler for spatial scheduling, a memory management unit for programmable logic (PLMMU) and a temporal scheduler for each hardware accelerator. The solution uses the concept of virtual FPGAs (vFPGA) to control access to hardware accelerators. With a vFPGA, each guest Operating System (OS) receives a virtual view of the FPGA, which only contains the hardware accelerators that the tasks performed within this guest OS require. The software tasks are not aware whether they have to share these accelerators with other tasks or have exclusive access. L4Re provides the IO server, which allows to set up a virtual bus from each guest OS to its vFPGA and to give access rights to MMIO address ranges. As Fig. 2.3 demonstrates, the MMIO address ranges include a virtual MMIO address range for

accessing hardware accelerators and a MMIO address range for the communication with the spatial hardware task scheduler. As the address ranges of all software tasks are disjoint, the sender of an AXI4-Lite message can be clearly identified using the sender's address. A task that attempts to use an MMIO address outside the address range assigned by the IO server receives a page fault. This prevents unauthorised access to the scheduler and to hardware accelerators.

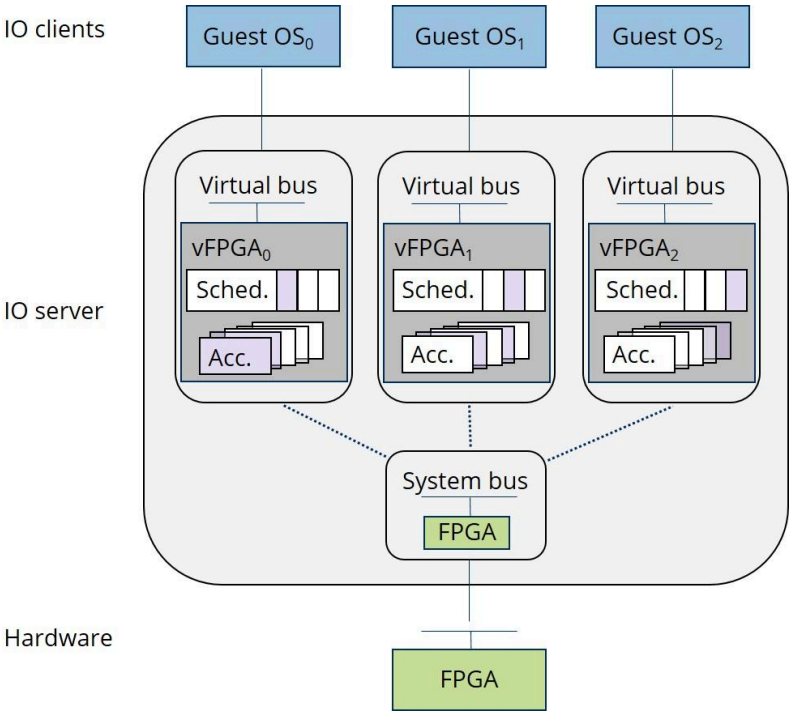


Figure 2.3: Virtual FPGAs

The three hardware components are not only used for access control, but also regulate the temporal and spatial distribution of the hardware tasks. The term 'hardware task' refers to a unit of execution on a hardware accelerator, respectively IP core. The hardware components include

- the spatial hardware task scheduler, which dynamically controls access to hardware accelerators and can therefore react to changing software task requirements,
- the Memory Management Unit for the Programmable Logic (PLMMU), which is used to translate virtual addresses into physical addresses and prevents any access to addresses that have not been entered,
- A temporal scheduler within the wrapper of each hardware accelerator, which ensures that prioritised tasks are given preference and contains disjoint buffers for the tasks' parameters in order to reach data isolation.

Periodic tasks in particular benefit from the proposed solution because once they have been assigned to a hardware accelerator, they can be executed as often as required without further spatial scheduling. Fig. 2.4 gives an overview over the interaction of the components.

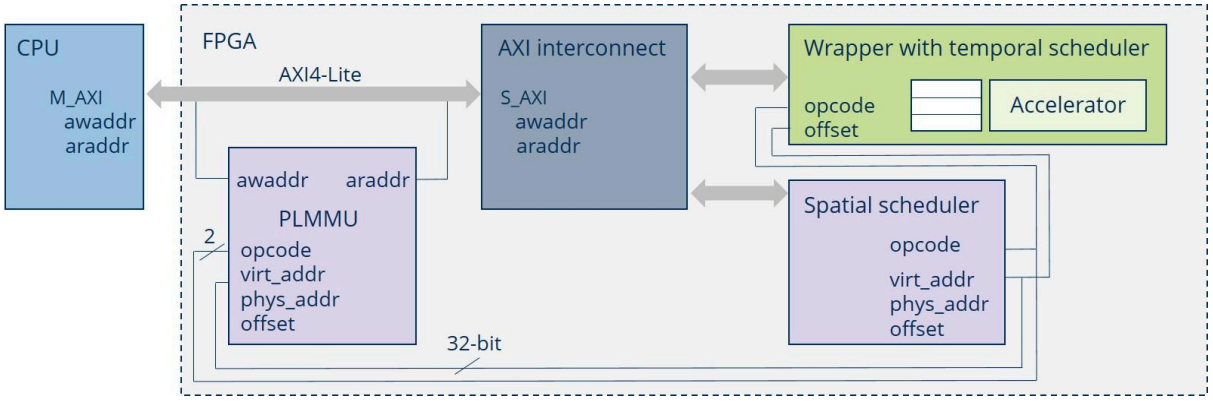


Figure 2.4: Access control with PLMMU and hardware task scheduler

If a software task requires hardware acceleration, it sends a request to the spatial scheduler via the AXI4-Lite bus. The scheduler identifies the software task using its unique sender address. The request contains the following parameters: Opcode, functionality, period, Worst Case Execution Time (WCET), Virtual Offset. The opcode (INSERT or REMOVE) is used to instruct the scheduler to create an entry in the PLMMU for a new task or to remove it for a terminated task. If there are several hardware accelerators with the same functionality, the utilisation of the accelerator determines the selection. The utilisation is calculated from the tasks' WCET and period. The virtual offset makes it possible for several software tasks of the same virtual address space to use hardware accelerators - each with a different virtual offset.

As soon as the hardware task scheduler has made a scheduling decision, it enters the physical address of the selected hardware accelerator and a virtual address from the virtual address range of the software task in the address table of the PLMMU. The PLMMU has been inserted into the read and write address channel of the AXI4-Lite bus in order to translate virtual addresses into physical addresses. As soon as an entry exists in the PLMMU address table, the software task can use the assigned hardware accelerator once or periodically without having to contact the spatial scheduler again.

Figure 2.5 shows the entries in the translation table of the PLMMU for two tasks of the same guest operating system whose virtual addresses must differ by an offset. In this example, all three software tasks require an encryption accelerator, of which only two instances are configured on the FPGA. Task 1 and Task 3 therefore share an accelerator. In order to achieve data isolation, the physical addresses of all tasks that share an accelerator must also differ by an offset. In the translation table of the PLMMU, each entry must be unique and each address range must be disjoint.

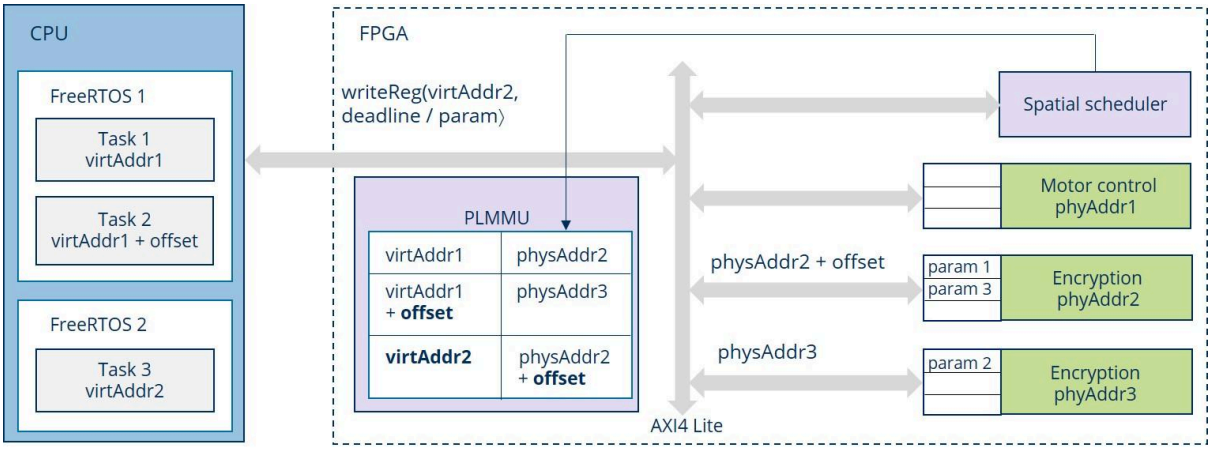


Figure 2.5: Data and control flow

If several hardware tasks share an accelerator, they are scheduled in temporal respect according to the Earliest Deadline First (EDF) policy. In addition to the parameters, each task must also transfer its deadline to the temporal scheduler. Figure 14 shows the finite state machine, which is located in the wrapper of each accelerator. The state “Data transfer” contains tasks that transfer data to the accelerator. As soon as the accelerator becomes free, the task with the shortest deadline is selected for the next execution from all the tasks that are still transferring data or have already finished doing so (“Ready” state). Once the execution has been completed, the done signal indicates to the calling software task that it can retrieve the result of the execution.

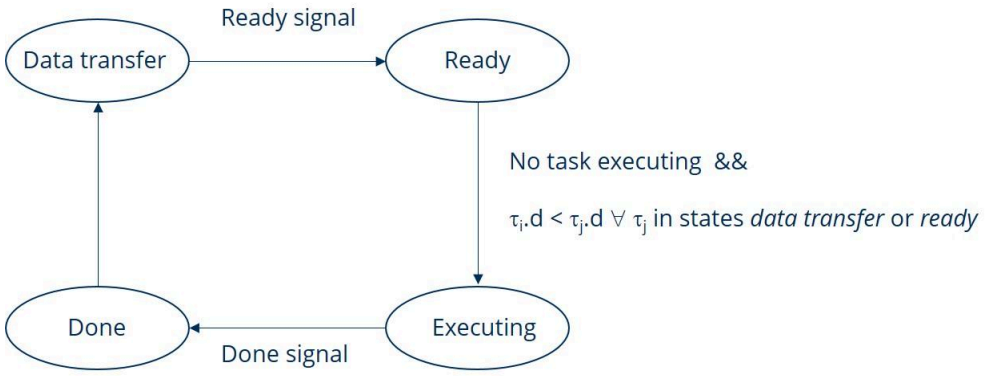


Figure 2.6: Finite state machine for temporal scheduling

2.4. Evaluation

Several AXI4-Lite-based hardware accelerators are available for use in mobile robots, e.g., wheel encoders, IMU or lidar. One possible use case is a robot with two software tasks that process two workpieces simultaneously. One workpiece at a time has exclusive access to an accelerator for motor control for a defined duration. The processing of this workpiece is started or ended with the opcodes INSERT and REMOVE, exclusive access is achieved by selecting a WCET equal to the period. While one workpiece has access rights to the motor control accelerator, the other workpiece cannot access it. An example of a shared

accelerator is an encryption IP core. Both tasks mentioned as well as other tasks that, e.g., monitor execution can use this accelerator to encrypt communication with a gateway or another mobile robot.

For the evaluation, we use a motor control accelerator and 4 encryption accelerators that implement the Prince method [UC2-10]. If a task attempts to access an address range of another task, the IO server generates a page fault. If the task accesses an address range that has been assigned to it by the IO server but for which it has not yet requested accelerator use, access is prevented by the PLMMU. In order to evaluate the advantage of the implementation in hardware compared to an implementation in software, a software version was created in which the software tasks communicate with a hardware task scheduler in software via Inter-Process Communication (IPC) and can only access the hardware accelerators via the scheduler. As Figure 2.7 shows, scheduling in hardware was executed 7.02 times faster than in software.

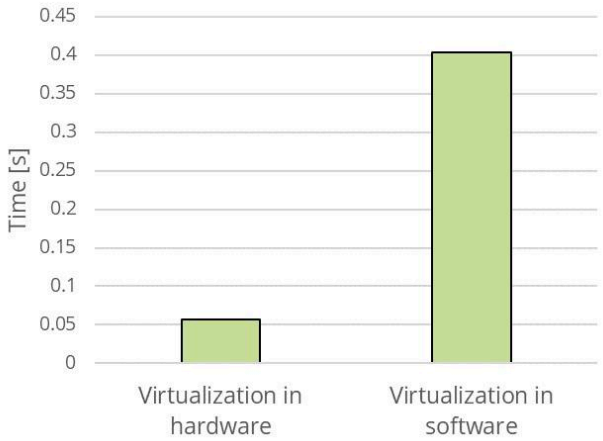


Figure 2.7: Execution time per 100 hardware task instances

However, this advantage comes at the cost of an overhead in temporal and spatial respect. The spatial scheduling takes an average of 105 clock cycles per task. However, the spatial scheduler is only called once per task. Figure 2.8 shows the overhead caused by temporal scheduling. First, an accelerator was called bare-metal, i.e. without memory protection and the option of shared execution. The middle bar shows the call of the same accelerator from L4Re (with memory protection, but without shared execution). Virtualized execution with temporal scheduling increases this latency by a factor of 1.47.

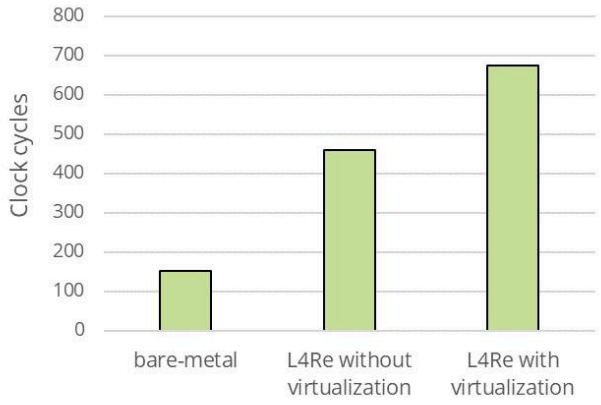


Figure 2.8: Overhead induced by temporal scheduling

Figure 2.9 shows the resource consumption of the spatial scheduler. It increases proportionally to the maximum number of hardware tasks. Additionally, 4 DSPs were used. The PLMMU with a maximum of 16 entries requires 2966 lookup tables (LUT) and 1345 flip-flops (FF). The temporal scheduler increases the resource consumption of the respective wrapper containing up to three tasks per accelerator by 470 LUTs and 628 FFs.

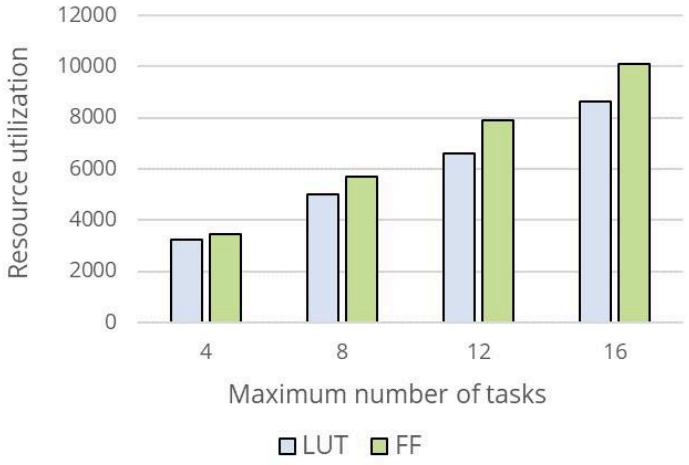


Figure 2.9: Resource consumption of spatial scheduler

2.5. Conclusion

The presented hardware-level control mechanism combined with a scheduler for hardware tasks prevents unauthorised access and allows to share hardware accelerators in compliance with data isolation and the preservation of dynamic task priorities.

Future work includes the evaluation of more complex use cases as well as the virtualization of interrupts. Further, the exchange of accelerators via Dynamic Function eXchange (DFX) is envisaged.

Bibliography

[UC2-1]	https://l4re.org/doc/index.html
[UC2-2]	https://micro.ros.org/
[UC2-3]	https://docs.xilinx.com/r/en-US/ug1137-zynq-ultrascale-mpsoc-swdev/Protecting-Memory-with-XMPU
[UC2-4]	Deb Nath, A.P., Ray, S., Basak, A. et al. “System-on-chip security architecture and cad framework for hardware patch.” *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. pp. 733–738, 2018.
[UC2-5]	Mbongue, J.M., Saha, S.K., Bobda, C. “Domain isolation in FPGA-accelerated cloud and data center applications.” *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. pp. 283–288, 2021.
[UC2-6]	Bobda, C., Mbongue, J. M., Saha, S. K., Ahmed, M. K. “Domain isolation and access control in multi-tenant cloud FPGAs”. Security of FPGA-Accelerated Cloud Computing Environments, Springer, pp. 29–55, 2023.
[UC2-7]	Elnaggar, R., Karri, R., Chakrabarty, K. “Multi-tenant FPGA-based reconfigurable systems: attacks and defences”. Design, Automation and Test in Europe Conference and Exhibition (DATE), Florence, Italy, IEEE, pp. 7–12, 2019.
[UC2-8]	Khawaja, A., Landgraf, J., Prakash, R., Wei, M., Schkufza, E., Rossbach, C. J. “Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS”. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 107-127, 2018.
[UC2-9]	Malik, A. A., Karabulut, E., Awad, A., Aysu, A. “Enabling secure and efficient sharing of accelerators in expeditionary systems”. Journal of Hardware and Systems Security, pp. 1-19, 2024.
[UC2-10]	Borghoff, J., Canteaut, A., Güneysu, T. et al. “PRINCE–a low-latency block cipher for pervasive computing applications.” *Advances in Cryptology–ASIACRYPT: 18th International Conference on the Theory and Application of Cryptology and Information Security,* Springer, *Proceedings 18*, pp. 208-225, 2012.

UC 3: Multi-Core RISC-V platforms (KU Leuven)

In D2.1 (Requirements Specification) we've reported a block diagram of the essential components on a single drone. This diagram was translated to a system-on-chip (SOC).

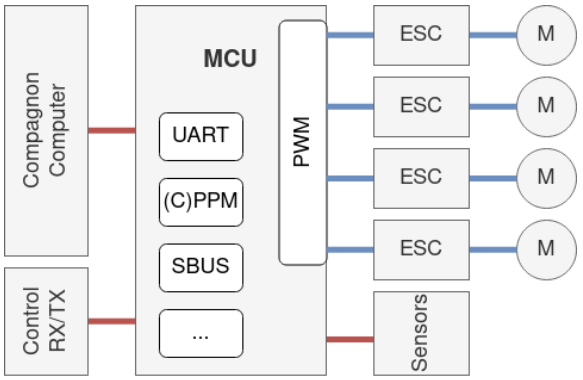


Fig. x: Block diagram of the components on a single drone

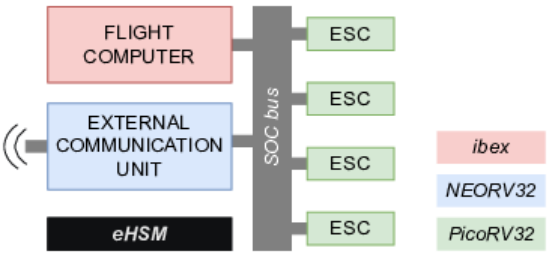


Fig. x: *Block diagram of the components on a single drone*

RISC-V is an open ISA that is freely available [UC3-1]. Many people and/or companies have made a hardware implementation of this processor, according to the specifications. As is typically the case, different implementations serve different problem-settings. This results in different implementations (with other optimisations) of the same specifications.

The block diagram above, shows three different (existing) implementations of the RISC-V ISA:

- Ibex
- NEORV32
- PicoRV32

Comparison of the chosen implementations

	Ibex	**NEORV32**	**PicoRV32**
HDL	SystemVerilog	Verilog	VHDL
Author(s)	lowRISC	Stephan Nolting	Yosys HQ
Licence	Apache License v2.0	3-clause BSD	ISC
Reference	[UC3-2]	[UC3-3]	[UC3-4]

For more information on the implementations, their respective websites can be visited. The HDL code of the implementations can also be obtained through the GitHub pages.

Security Solution

The security solution in this use case is discussed here.

An observation that is made is that one single drone can be seen as a “swarm” of different processors. In the block diagram of the essential components, three different types of processor were identified. Two of these processors are present once on a single drone, while the third one is present four times.

With an industry-grade drone, this number of processors is higher, but the proposed security solution can scale up without any issues.

To ensure that one drone, or “a group of processors collaborating towards a shared mission”, is secure, every single one of these processors should be checked. With the number of processors increasing, scalability will be an issue. The proposed solution is to substitute all the processors with a RISC-V implementation on a single FPGA.

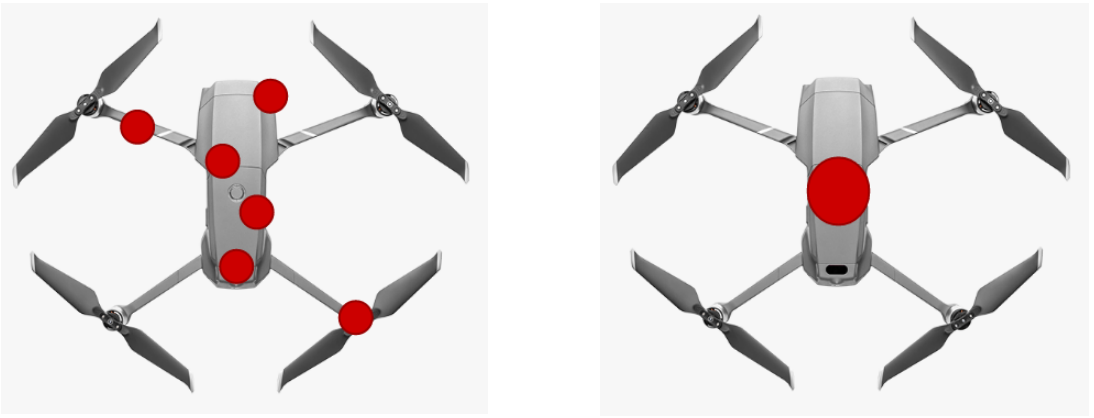


Fig. x: The security solution lies in the substitution of different processors with an implementation on a single FPGA

To verify that a processor is in a known-and-trusted state, the techniques in Attestation are used. Within this setting, there are two entities: a **Verifier** and a **Prover**. The Verifier is the entity that wants to check the health of the entity that is referred to as the Prover. Typically a protocol is played between both entities that look like shown below.

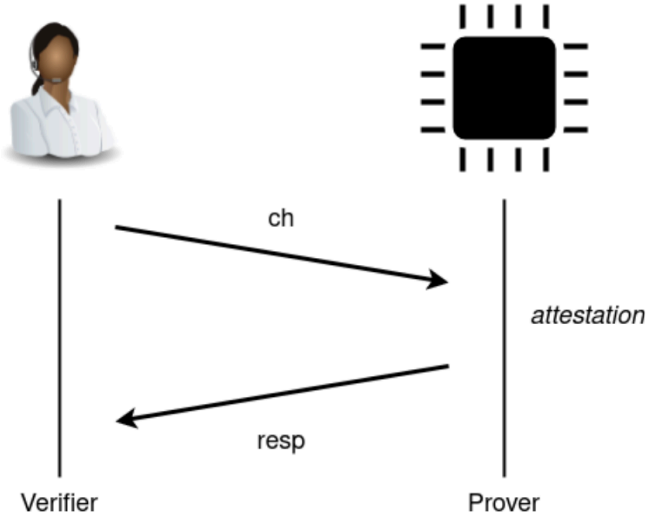


Fig. x: A typical protocol that is played between the Verifier and the Prover to check whether the targeted device is in a known-and-trusted state.

Given the setting that a single drone has many different processors, running an attestation on the entire drone will be a considerable job. With the number of processors increasing, this attestation will be an increasingly work-intensive exercise that does not scale well.

With the proposed security solution, this non-scalable issue will be solved. Instead of having to run multiple attestation protocols, only one single attestation round has to be run on the FPGA.

Technical background

An FPGA, which stands for Field Programmable Gate Array, is a versatile integrated circuit (IC) chip. Unlike most ICs that are manufactured to perform a specific function, FPGAs are designed to be configurable after they're built. They can be seen as blank circuit boards with a ton of tiny switches. By flipping these switches the behaviour of the FPGA can be configured. This allows for creating custom digital circuits for a wide range of applications.

Although there many different building blocks in an FPGA, the two most important ones are:

- **Configurable Logic Blocks**: FPGAs are filled with these blocks, which act like the building blocks of digital circuits. By programming them, they obtain their desired functionality. A CLB mainly consists of flip-flops (FFs) and look-up tables (LUTs).
- **Programmable Interconnects**: All the CLBs (and the other componentes, not mentioned explicitly here, have wires that can interconnect the different building blocks. Which wires are connected and which ones are not, can also be programmed in these programmable interconnects.

The configuration of these reconfigurable resources typically comes from a (volatile) memory. The configuration, which is called a bitstream, is stored in the configuration memory and dictates how the underlying reconfigurable fabric should behave. For example: CLB x holds an “AND-gate” and uses the flip flop; or the output of CLB y is wired to the input of CLB z. The image below illustrates this: the bitstream in the configuration memory determines the functionality of the FF, the programmable interconnect and the LUTs, in the reconfigurable fabric.

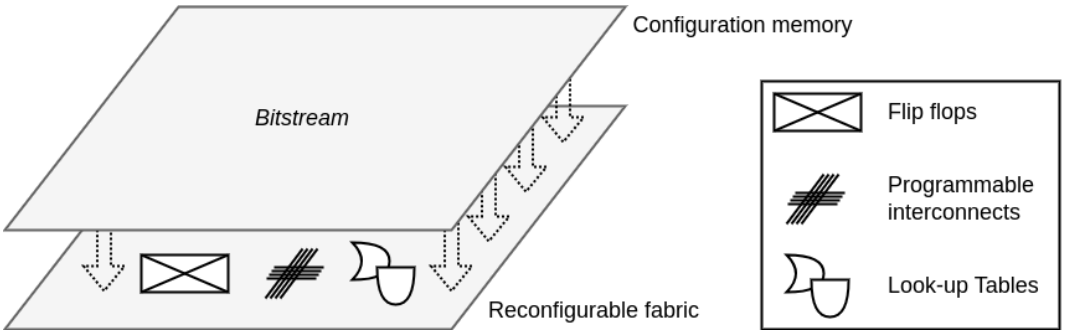


Fig. x: An illustration of the internal workings of an FPGA.

The writing to the configuration memory typically uses some industry standard, like JTAG, to transfer the configuration from a computer to the FPGA. In a typical setting, the reconfigurable fabric has no (need to) access the configuration memory. However, modern FPGAs have a special component that does allow this. The Internal Configuration Access Port (ICAP) provides an interface to the configuration memory from the reconfigurable fabric.

The smallest “unit” in a bitstream is referred to as “a frame”. Although there are some differences with (older) FPGA devices and/or brands, a typical, modern AMD/Xilinx FPGA has frames of 101 32-bit words.

Using the ICAP to obtain the Security Solution

The reconfigurable fabric can be configured to (almost) any digital design. This still holds for implementing the RISC-V processor. When all the required implementations have been decided on, they can be implemented next to each other in a single FPGA, as is shown in the image below.

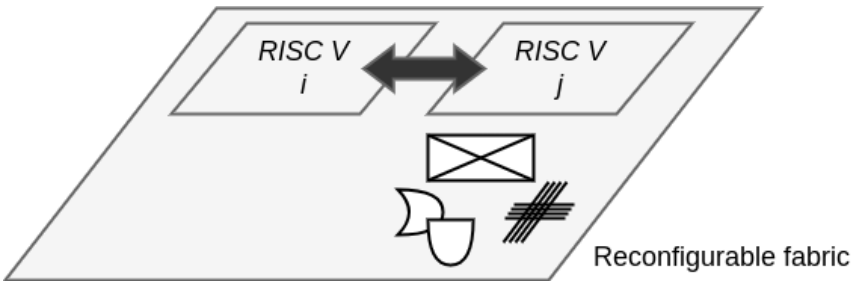


Fig. x: Multiple RISC V implementations on a single FPGA.

To obtain the proposed Security Solution, one more implementation needs to be added. This component is the embedded Hardware Security Module (**eHSM**), which is a new component that is designed, and implemented by the research team.
The eHSM is placed in a specific position, so it has access to the ICAP, as illustrated below.

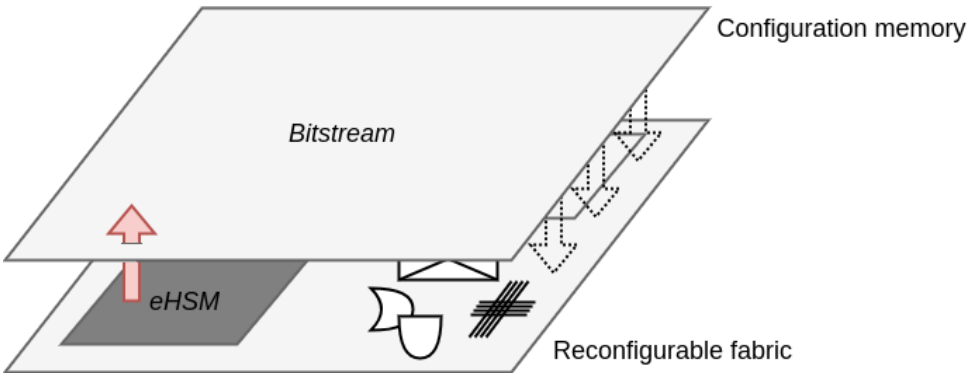


Fig. x: RISC V implementations with the eHSM

By using the ICAP, the eHSM can read back the content of the configuration memory.

It is important to highlight that the bitstream defines **the design and the routing** of the configuration.

The eHSM

The eHSM will have exclusive access to the FPGA's configuration memory. Given the coordinates of a certain frame, the eHSM can read back the configuration of that single frame. Similarly, a whole range of frames can be read back.

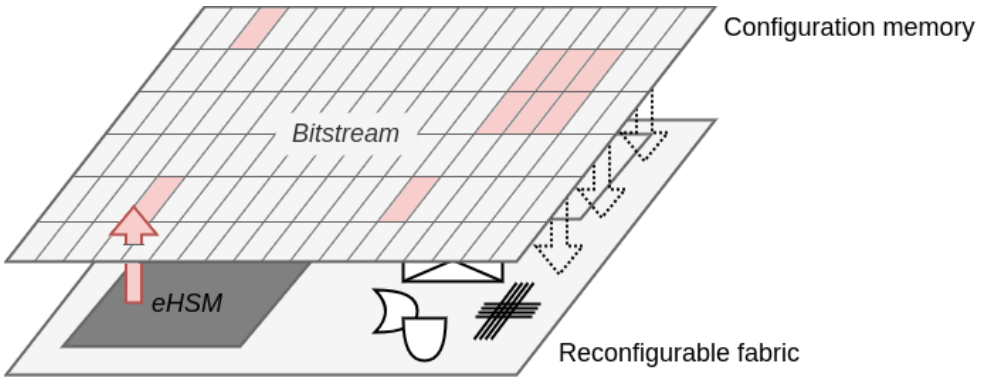


Fig. x: eHSM with frame-precise read back functionality in the eHSM

When a Verifier wants to run an attestation protocol with the Prover, in this case the FPGA, the verifier can choose which parts of the FPGA need to be verified. This allows for multiple attestation strategies.

One single RISC-V

The verifier can instruct the eHSM to do the readback of the configuration memory of a single RISC-V implementation. By doing so, both the firmware and the hardware of the processor can be verified.

“All” single RISC-V’s

The verifier can instruct the eHSM to do the readback of the configuration memory of all RISC-V implementations. By doing so, the entire configuration of the drone can be verified.

Entire FPGA

The verifier can instruct the eHSM to do an entire readback of the configuration memory. This allows a verifier to additionally check eHSM.

Randomly selected frames

The verifier can instruct the eHSM to do the readback of randomly selected frames. Although the coverage of this strategy is not as good as the readback of an entire FPGA, the order in which the frames are selected is another source of entropy.

Adding a MAC

It’s pointed out, again, that with this readback everything can be checked. Hardware configuration of a processor, firmware that is running on the processore, but also the wiring that is configured. The latter implies that, in case an attacker is eavesdropping certain wires, this can also be detected.

Given the Documentation [UC3-5], a CLB on one specific (X,Y) coordinate can have up to 36 frames of configuration. With 101 words per fame, and a word being 32 bits in width (= 4 bytes), this comes down to 36 x 101 x 4 = 14544 bytes.
This illustrates that simply returning all the read back data to the verifier is not a scalable solution.

To have a solution for this large amount of data, it is proposed within the solution to use a Message Authentication Code (MAC) on the data. Using a MAC is a cryptographic technique which provides data authentication. Data authentication is achieved when there is both data-origin authentication and data integrity.

Data-origin authentication implies that the receiver of a message can be certain that a message is sent by the intended sender. Data integrity implies that the message is received identically to how it is sent.

Within this use case it is selected to use an algorithm that can provide authenticated encryption. This is a cryptographic technique that provides both confidentiality and data authentication. It achieves this by simultaneously encrypting the data using a secret key (rendering it unreadable to unauthorised parties), and generating an authentication tag. This tag, calculated using the same key and on the data itself, allows the recipient to verify the data's integrity and origin. Any modification to the data or the tag will be detected, preventing forgeries and ensuring the data comes from a trusted source.

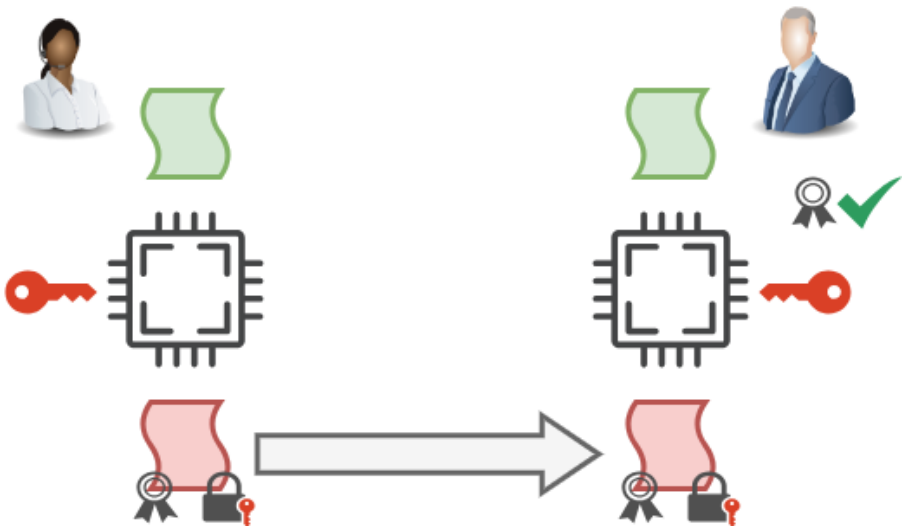


Fig. x: Visualisation of authenticated encryption

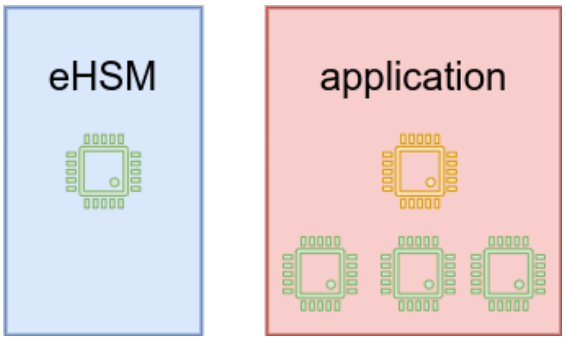
The image above illustrates how authenticated encryption works. The sending entity (on the left) uses a key and transforms a plain text message (in green) into a cipher text (in red) and a tag. This is sent to the receiving entity which then uses the same key to decrypt the message and verifies whether the tag is correct.

Given the rather constrained processing environment that comes with IoT devices, a choice is made to use ASCON [UC3-6]. This algorithm was chosen as the winning candidate in the Lightweight Cryptography Competition which was held by NIST between 2019 and 2023.

Results

This section describes the results of the implementation.

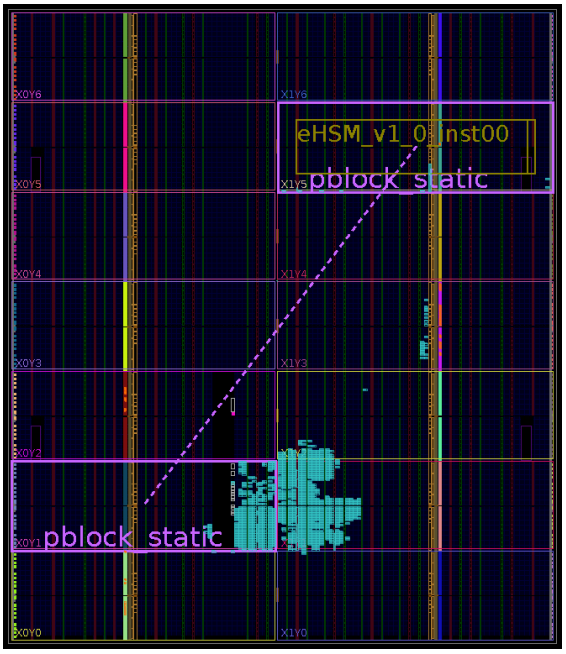
FPGA implementation



The FPGA implementation is split into two main parts: the static part (which contains the eHSM) and the dynamic part (which contains the application, in this case the drone components). The eHSM contains 1 RISC-V implementation (PicoRV32) and the application contains 4 implementations (NeoRV32 + three PicoRV32).

Both ‘partitions’ are synchronised on different clocks to prevent all links.

This division can also clearly be seen in floorplanning. The two purple rectangles in the image on the side represent the eHSM partition. The reason for this consisting of two parts is because of the use of a general purpose development board. Both the incoming clock signal and the access to the ICAP need be positioned in the eHSM partition.

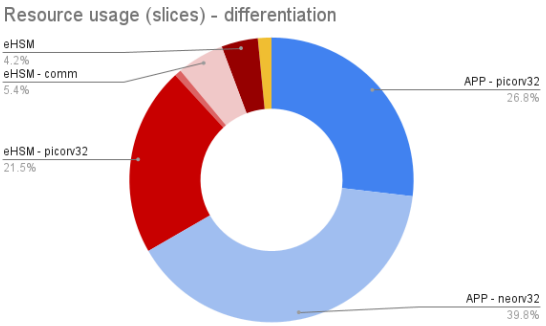


Visually, it can already be seen that there is still a lot of reconfigurable fabric unused. On the one hand this is because a rather high-end FPGA is used, on the other hand, also the choice in design strategy has its impact.

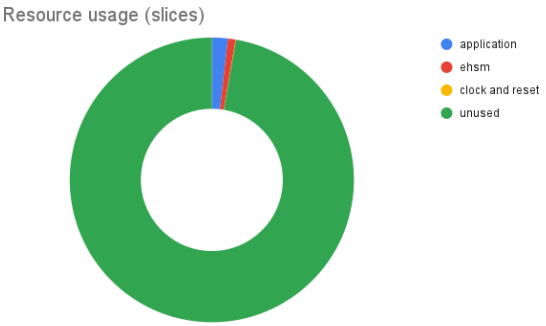
The table below summarises the used resources for the final FPGA implementation.

		Slices	BRAM	ICAP	MMCM
Clock & reset		**32**	**0**	**0**	**1**
eHSM		**657**	**3.5**	**1**	**0**
	SoC with PicoRV32	*444*	*2.0*	*0*	*0*
	LEDs	*16*	*0*	*0*	*0*
	Communicator	*111*	*0*	*0*	*0*
	eHSM core	*86*	*1.5*	*1*	*0*
Application		**1379**	**9**	**0**	**0**
	SoC with PicoRV32	*555*	*2*	*0*	*0*
	SoC with NeoRV32	*824*	*7*	*0*	*0*

Relative resource usage in the entire XC7VX485, and in the “used” part are shown in the pie charts below.

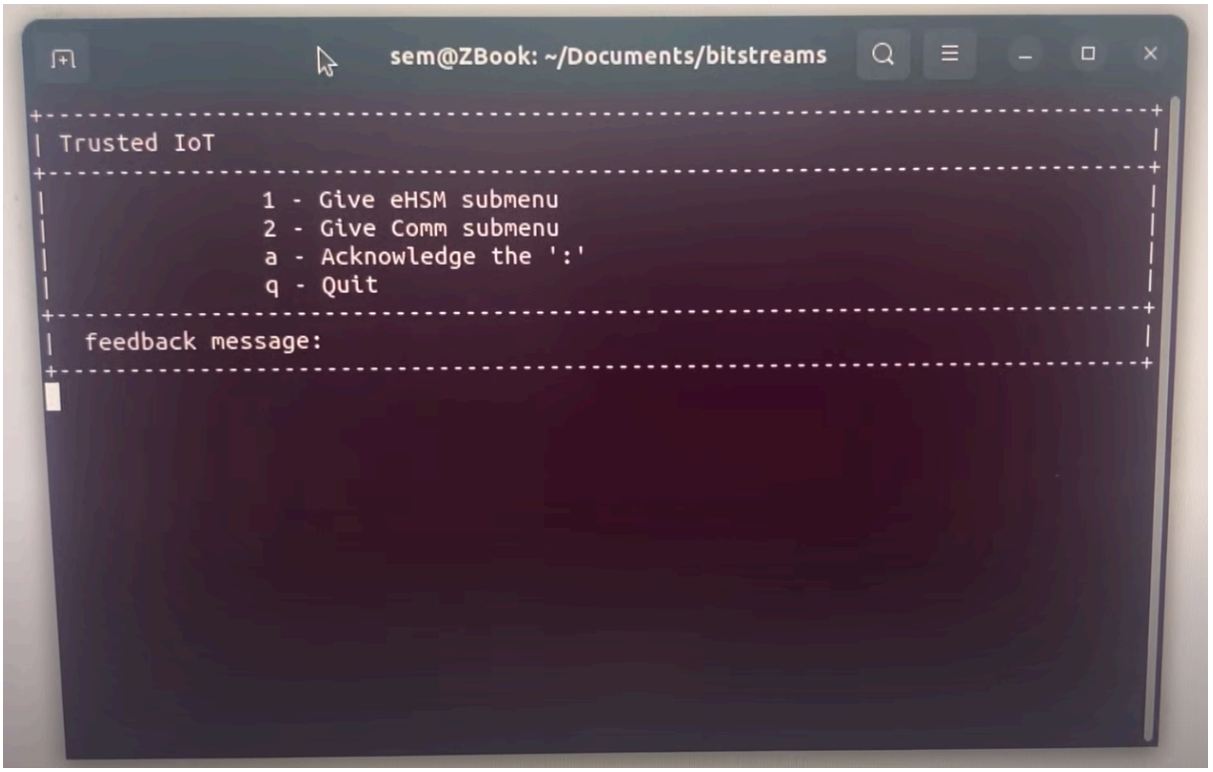


Relative slice usage in used design



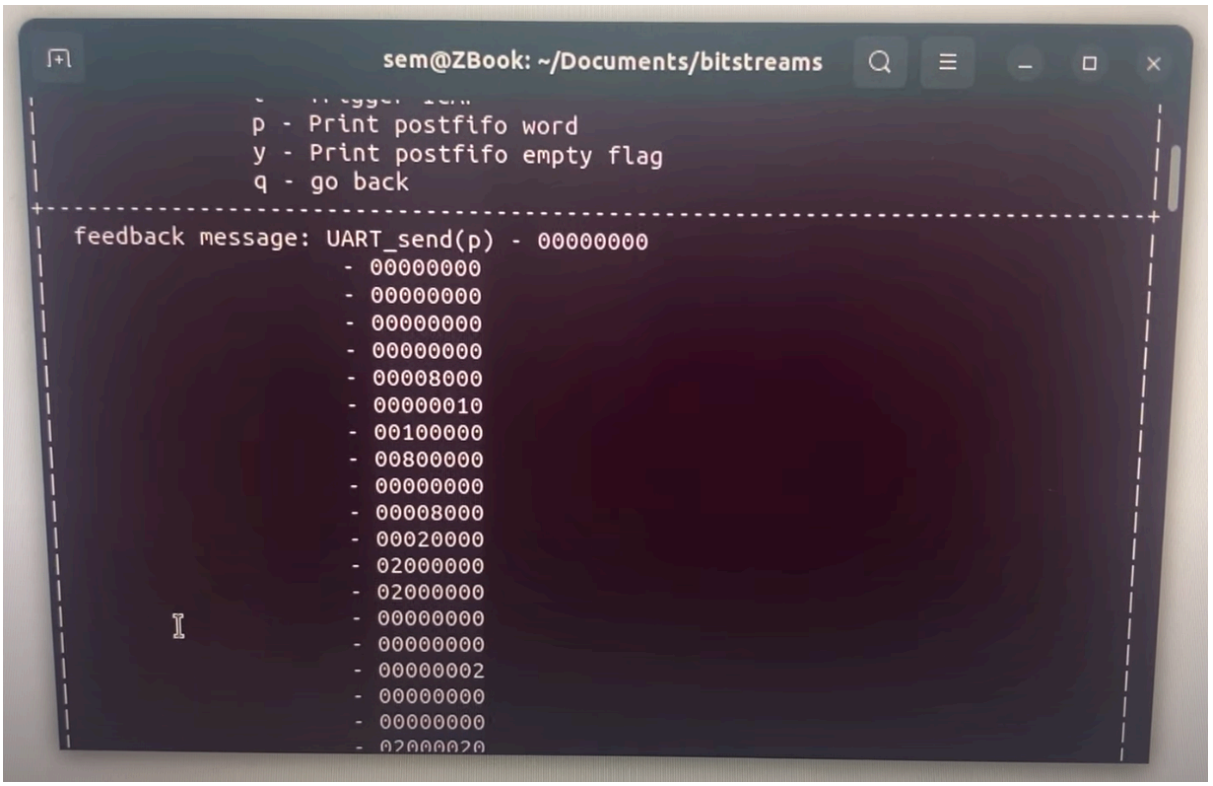
Relative slice usage in FPGA

A small User Interface script is made so communication with the drone (both the eHSM and the application) can happen through a text interface.



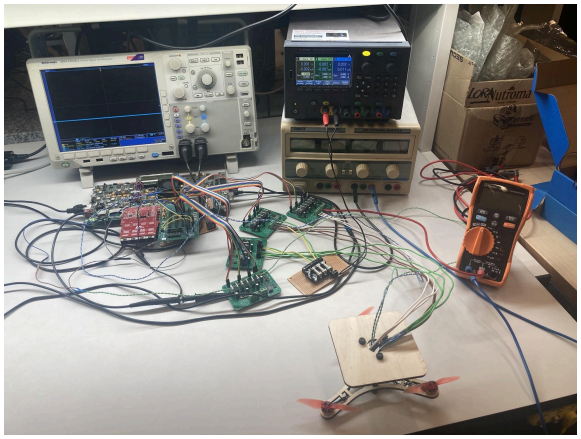
Text-based user interface

Readback of the configuration gives the configured data in a per-frame-basis. A frame is the smallest unit of configuration that can be read back and consists of 101 32-bit words.

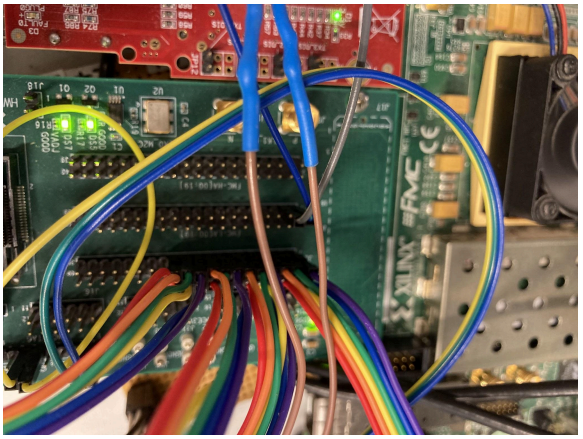


Demonstrator

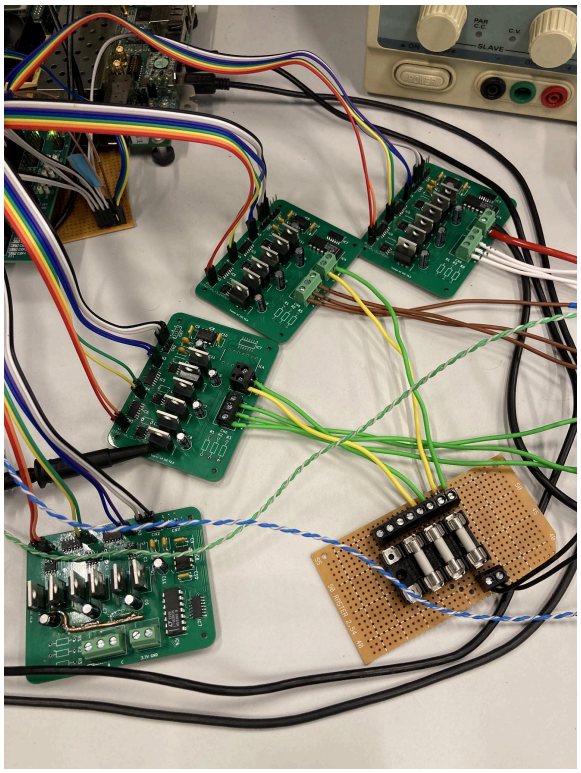
The demonstrator has been implemented. The tricopter consists of a wooden frame with 3 BLDCs. An inertial measurement unit (MPU6050) was added for steering. A guiding system was installed for safety reasons.



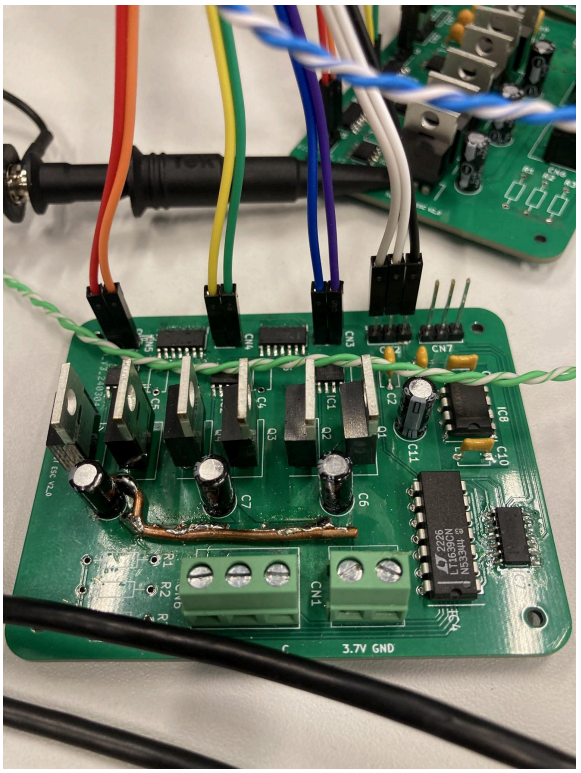
The entire setup with FPGA, power supply, PCBs, battery connector and drone



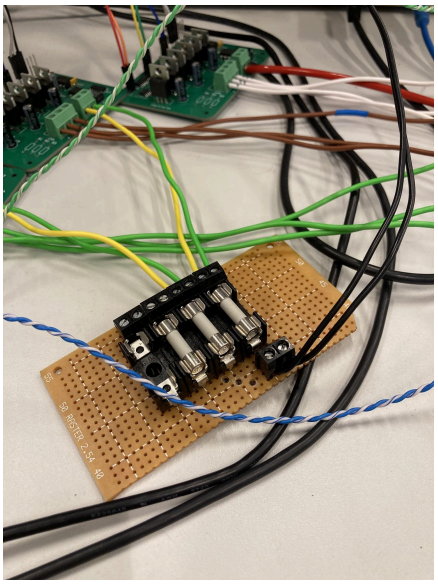
Close-up of PCB driving



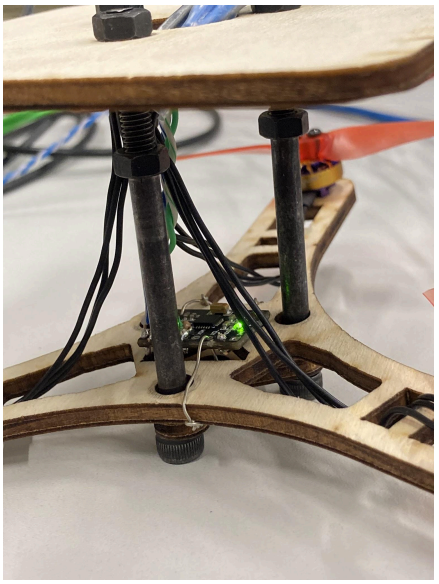
PCBs with power electronics



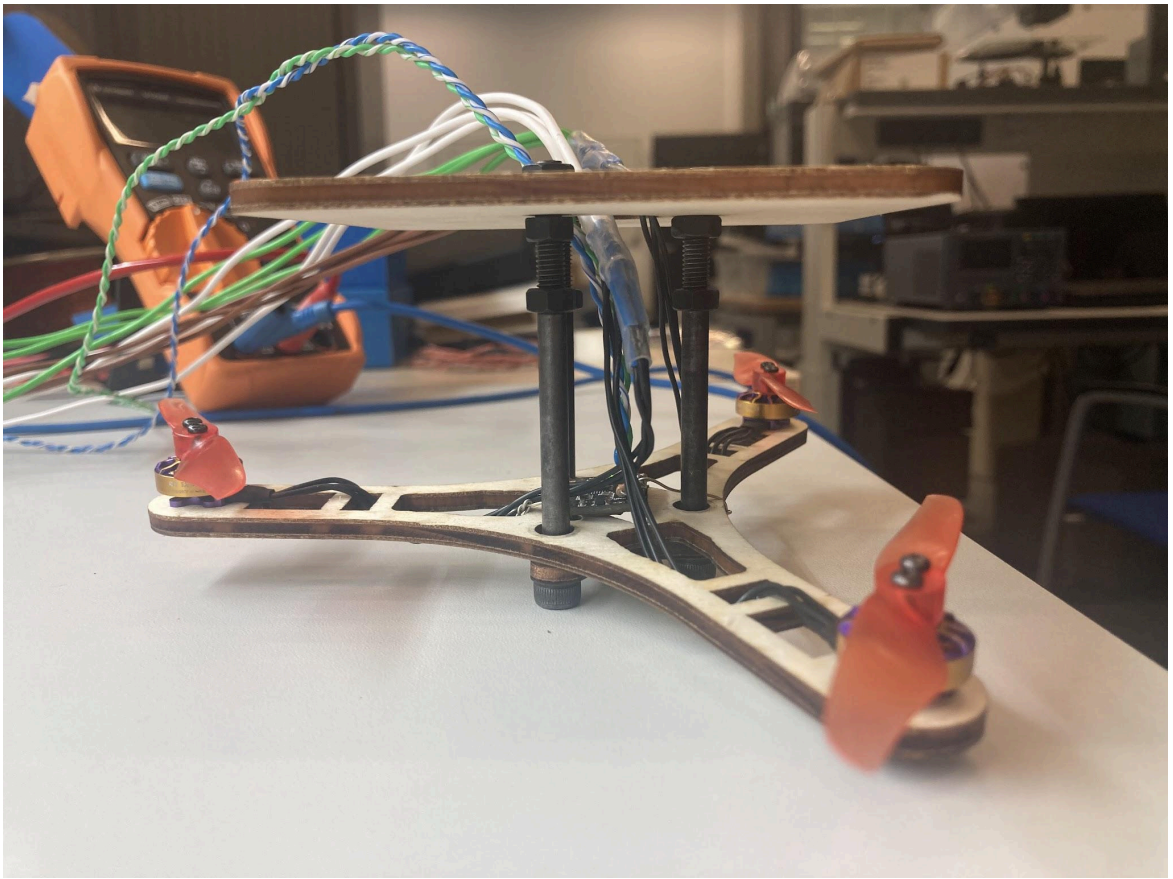
Detailed photo of a single PCB



Battery connector with fuses



Drone with MPU6050



Drone with guiding system

Bibliography

[UC3-1]	https://riscv.org/technical/specifications/
[UC3-2]	https://github.com/lowRISC/ibex
[UC3-3]	https://github.com/stnolting/neorv32
[UC3-4]	https://github.com/YosysHQ/picorv32
[UC3-5]	https://docs.amd.com/v/u/en-US/ug470_7Series_Config
[UC3-6]	https://ascon.iaik.tugraz.at/

UC 4: Heterogeneous Embedded system Architecture (VUB)

Platform requirements

A number of major microcontroller manufacturers offer new technologies related to embedded hardware security in their portfolio. Vendors such as Microchip, NXP, STMicroelectronics, Infineon and Texas Instruments offer a variety of platforms based on the ARMv8 core family. The primary implementation of this core family was intended for the 64-bit high-end platforms running at MHz and beyond and started to emerge in 2011. Security has been an increasing concern in the Internet of Things (IoT) applications and since a few years the upcoming of 32-bit versions of the ARMv8 core family made its entrance in the low-end ultra-low power devices (ARM Cortex-M23) and the mid-range (ARM Cortex-M33) devices optimised for embedded computations. Many security features are now available. Note that the list of available microcontrollers and platforms is rapidly evolving. Indeed, manufacturers are currently broadening their portfolios with new features and product families to suit most security requirements. The presence of the following hardware-accelerated solutions in the microcontrollers are taken into account when selecting appropriate platforms for embedded secure sensing and processing:

- Secure boot (and secure update): this mechanism is achieved via both “integrity verification” (via hashing, SHA or HMAC), and via “authenticity verification” (checking digital signature). Different implementations and features are supported by the different vendors. Within the ARM Cortex-M23 implementations, Microchip offers microcontrollers which have only the secure boot option and some microcontrollers which also support root of trust (RoT) together with secure boot [UC4-1]. In the simple case, the microcontroller checks the integrity of the application to be booted via an internal checking mechanism. For the second option, the microcontroller requires the assistance of a crypto-authenticator module such as the ATECC608B module to be attached to the microcontroller. Aside of the integrity check, the RoT also allows to verify the authenticity of the firmware to be booted. Since April 2024 Microchip also manufactures the PIC32CK series with the ARM Cortex-M33 core inside. Other vendors such as NXP (LPC5500, high-end IMx8 family) , STMicroElectronics (STM32L5 and STM32U5) implement similar features but in a slightly different manner. Texas Instruments also provides some platforms for the high-end processing (AM335 and higher).
- Secure code (hardware) isolation (TrustZone) or Trusted Execution Environment: in the low-end device category, the microcontrollers based on the Cortex-ARM23 and Cortex-ARM33 architectures currently have the ability to isolate secure code from non-secure code. The effective (hardware) partitioning of both the flash memory and RAM memory ensures that non-secure code cannot gain undesired access to the secure code sections. Communication between the two parts takes place exclusively via function calls demarcated with specific hashing techniques which are called veneer functions. If the returned hash does not match, the calling code is not reliable. Special notes need to be drawn here. Although both the secure and non-secure codes can be programmed using the C programming language, it is not possible to develop the secure application with C++. Indeed, vtables are not (yet) supported by

the secure environment as these lead to special memory operations which may lead to stack/heap overflows. Moreover, many libraries are only available to the non-secure code section because of these reasons. The major vendors include the PIC32CM5164Lx (Microchip) [UC4-1], STM32L5 [UC4-4], STM32U5 [UC4-5] and STM32H5 [UC4-6] (STMicroeletronics), and the LPC5500 [UC4-3] (NXP) single core and the MCX-N [UC4-2] dual core series. These series of microcontrollers also have secure boot and secure update mechanisms.

- Cryptographic acceleration: Data encryption and hashing can take place in both hardware and software. However, the hardware variant is preferable due to lower consumption and more efficient processing. Many microcontrollers offer P-256 Elliptic curve cryptography (ECC), AES, public key signing, compression techniques and SHA. A “true random generator” based on subtle hardware (silicon) differences is also included in some microcontrollers. This includes the PIC32CM5164Lx (Microchip), STM32L5, STM32U5 and the STM32H5 (STMicroeletronics), and the LPC5500 and the MCN-X (NXP) series. These series of microcontrollers also have secure boot and secure update mechanisms.
- Secure peripherals: in addition to secure code, peripherals are also divided into a secure and non-secure section. This is inherently related to the presence of TrustZone, where peripherals (UART, SPI, I2C, RTC, etc.) have a double set of hardware registers. Once a peripheral is linked to, for example, the secure section, it can no longer be used in the non-secure section.
- Secure storage: Many microcontrollers targeting security also offer mechanisms to permanently store data in flash in a “secure manner”. Scrambling techniques (based on a user-defined key) are also used on both the data and the addresses of the stored data. Secure key storage is also mandatory.

Proposed implementation

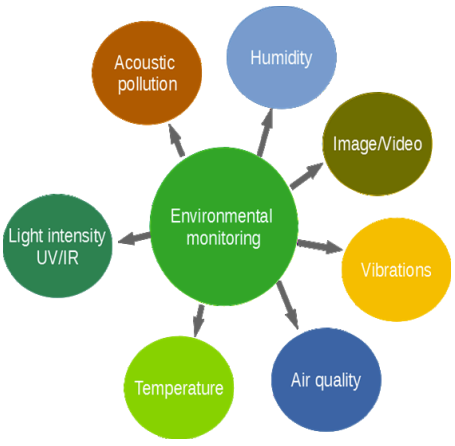


Figure 4.1: Use case of the VUB. Multiple types of sensors can be read out and processed locally before being transmitted to a remote server.

The use case of environmental monitoring (Figure 4.1) involves the capturing of sensory information such as temperature, humidity, vibrations, light intensity, images/camera, etc. In

our scenarios we make a distinction between high-end sensing requiring a high data throughput and low-end sensing which requires minimalist processing capabilities. After some local processing, the measured data is sent to a remote server for further processing.

Low-end sensing

Hardware
In our low-end implementation, we develop a prototype which is capable of measuring sensors that are read-out once per second. These sensors include humidity, temperature, vibrations, light and infrared intensity. From the list of requirements, the PIC32CM5164LS family of microcontrollers fits for a demonstrator prototype. Our prototype is built around the PIC32CM5164LS00064 microcontroller which is based on the ultra-low-power ARM Cortex-M23 architecture. This microcontroller offers the following key security features:

- Secure boot. Via internal registers it is possible to do an integrity check of the non-secure firmware. Root of Trust (RoT) is only possible with the PIC32CM5164LS60064 microcontroller. This microcontroller was during the design stage of the prototype not available for purchase. However, the board layout can be re-used and adapted to fit the ATECC608B crypto authentication module to boot with RoT.
- TrustZone: up to 5 regions in flash, 2 regions in data flash and 2 regions in the SRAM.
- A True Random Number Generator based on a random entropy source.
- Support for several encryption and hashing methods (AES and SHA). Although the programming mentions hardware acceleration most of it is achieved via software libraries.
- Anti-tamper IO + secure sercom (SPI, UART, I2C) multiplexing for secure communication. The secure and non-secure communication channels have dedicated sets of registers so overwriting from the non-secure application shall not be possible.

The sensors we read out are for most part available on breakout boards. Combining the breakout boards with the development board of Microchip would theoretically be the shortest route to the development of the firmware. The connection between the breakout boards and the development board would be achieved via jumper wires. These jumper wires are prone to suboptimal connections which lead to wrong sensor read-out or to “dead” sensors. A more robust method is to design a custom board which integrates the microcontroller and the sensors which are chosen as desired. The knowledge and expertise in custom printed circuit board (PCB) designs in the lab allowed us to efficiently design a custom board to house the microcontroller and the sensors. The design of the microcontroller board is shown in Figure 4.2.

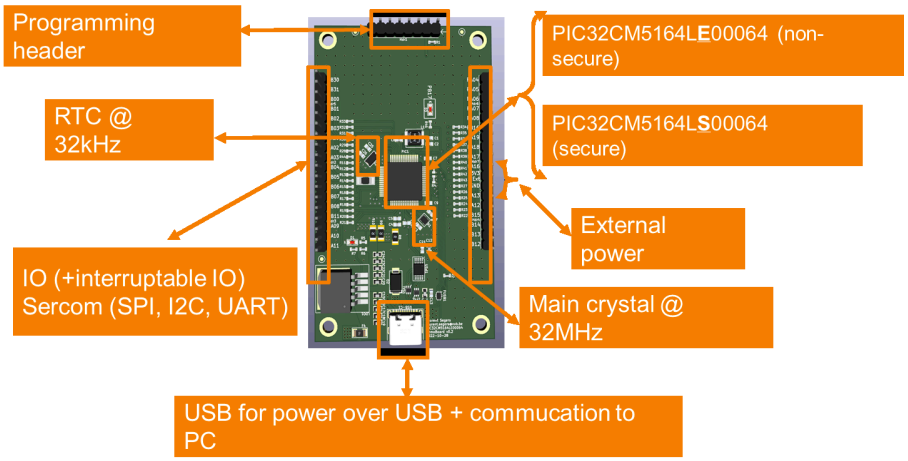


Figure 4.2: Printed circuit board housing the PIC32CM5164 with the vital components and the headers to house additional PCBs for reading our sensors.

The PCB houses a USB type C connector for power. The same connector is also connected to allow USB 2.0 communication to pass through to a connected host. Two rows of headers are provided onto which custom shields can be mounted. In our case such a shield contains all the sensors which can fit the low-end sensing. A programming header allows the programming of the microcontroller from MPLabX IDE with the PICkit 4.

A second PCB housing all the sensors is designed and shown in Figure 4.3. The following sensors are chosen and implemented on the PCB:

- SHT41: temperature and humidity,
- ADXL343: 3 axis accelerometer,
- SPU0410LR5H-QB: analog microphone to detect the amplitude of the environmental sound expressed in dBSPL,
- VEML3328: light and infrared sensor

Aside from the listed sensors, a GPS module (L96-M33) is selected to obtain the current location of the device. A UART-to-USB converter and a USB connector are provided to facilitate debugging via message echoing. An SD-card allows to temporarily store collected data. At last, an ATSHA204A crypto-authentication module is added on the sensor board. This module allows some cryptographic operations which are required during the process of secret key renewal (key agreement protocol).

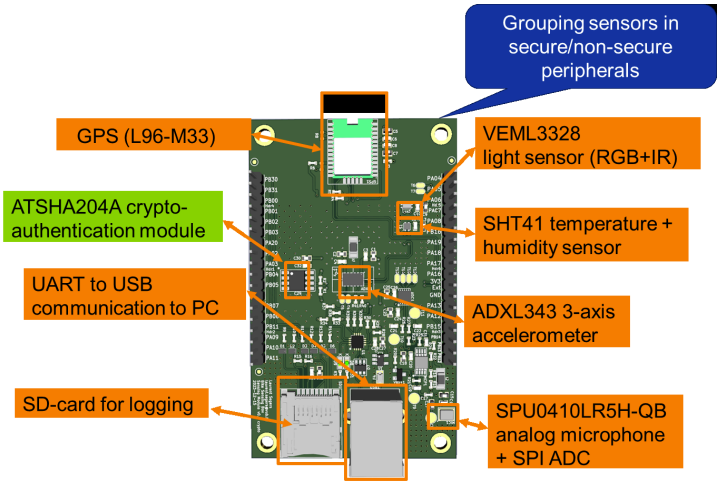


Figure 4.3: Sensor board which houses the sensors such as the SHT41, the ADXL343, a microphone and the VEML3328. It also incorporates an SDcard reader, a crypto-authentication module together with some debugging options and a GPS module.

Software/Firmware

The software to configure the microcontroller and the remote communication consists of 2 major parts. On the one hand, there is the firmware of the microcontroller and on the other hand, a software to interact with the device on a remote computer is to be developed. Both microcontroller and the computer software share an important but opposite functionality. Indeed, the firmware of the microcontroller is intended to collect, transform and transmit the sensory information while the software of the computer is able to decode and represent the transmitted data. Figure 4.4 demonstrates schematically the operation of the simplified data flow at microcontroller level without the functionalities of TrustZone.

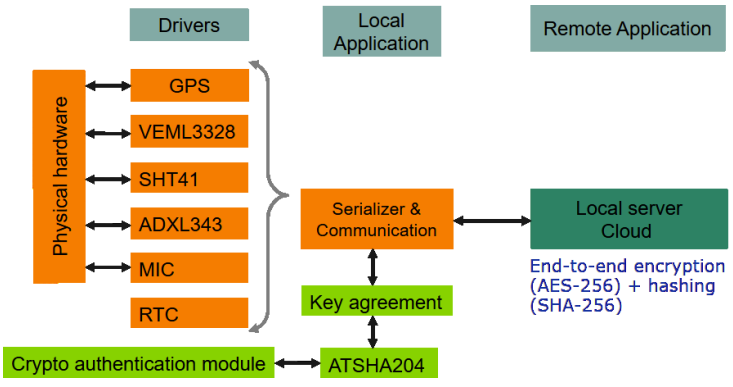


Figure 4.4: Schematic overview on how low level drivers are designed on a platform without TrustZone.

The firmware structure is laid in a multilayered manner. At the lowest level we provide a driver to interact with the sensors. At the next level serializer takes care of proper communication between the device (MCU) and the remote computer, shown as “Local

server / Cloud".
For each of the attached sensors a driver is implemented to separate the functionalities in a logical fashion. Each of the drivers provides the glue logic between the low level hardware transactions and the communication mechanisms at higher level. We have opted for a general approach in which any type of sensor can be added regardless of the selected platform and the selected sensors. The drivers are written in C++ in an object oriented manner and share the 3 same functions: "Decode", "Encode" and "Process". These functions are defined in a common class to be inherited by each of the desired driver subclasses. The goals of these functions are:

- Encode: the locally processed sensor information (e.g. ADXL343, VEML3328, etc.) is encapsulated in a common data structure which is converted into a serial byte stream fashion. In general the "Encode" method is intended for all communication destined to the other side of the communication medium. The driver module packs all required information into a stream of bytes which can be unpacked at the opposite side of the communication medium.
- Decode: is the opposite function of "Encode". All the received bytes are decoded and interpreted. According to these bytes some actions can be performed. These actions can be interpreting a sensory value, obtaining the status of the counterpart, etc.
- Process: this method is periodically issued. All the drivers have the "freedom" to select the appropriate operations handled in the "Process" method. In our case we selected to read out sensor values based on the received input via the "Decode" method.

The "Serializer and Communication" takes all the datastreams from the individual drivers and sensors and generates one single packet with all the available data. The Serializer can also apply encryption and hashing with available methods for the PIC32CM5164 families. These include AES and SHA and allow to guarantee data encryption and integrity. The Serializer also generates a header which contains the information address of the sender, receiver and the total length of the packet to be decoded. The hashing and encryption are performed on the payload only. The above method works well for microcontrollers which do not offer support for TrustZone. A similar strategy can be applied for microcontrollers offering TrustZone. The adapted schematic with TrustZone is shown in Figure 4.5.

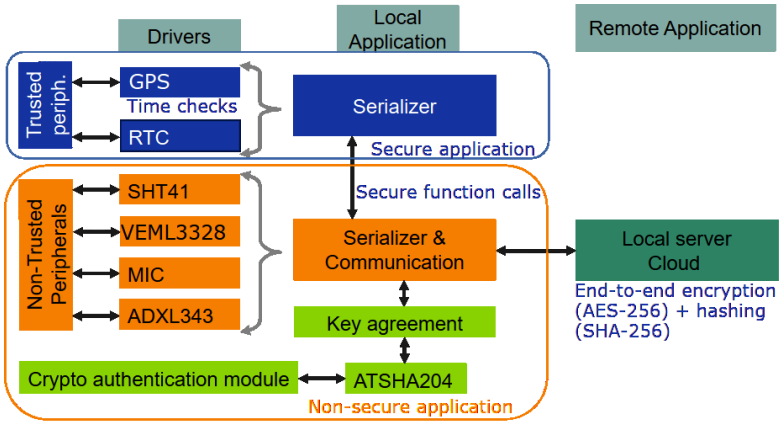


Figure 4.5: Schematic overview on how low level drivers are designed on a platform

including the TrustZone technology.

In the scheme shown in Figure 4.5, the application is subdivided in a secure and a non-secure application. The communication between the secure and the non-secure applications is performed via “veneer”-functions. These functions allow checking the validity and integrity of the transferred arguments before these arguments can be passed on to the secure world. These “veneer”-functions are therefore located in a “demilitarised” area. In our application, we provide the same 3 functions to pass information between the secure and non-secure area: “Decode”, “Encode” and “Process”. Two key concepts are taken into account:

- The non-secure application can call the 3 mentioned functions. These function do, however, not access directly the GPS/RTC. Instead, both the GPS and RTC drivers collect their respective data via interrupt methods and store that information in the local secure SRAM. The veneer functions then request for the data which can then be shared.
- The firmware code for the secure application must be written in C. C++ is not supported for that area of the microcontroller. Moreover, since the secure area has very strict programming rules with regards to stack usage, all warnings such as unused variables are treated as errors prohibiting the compilation of the secure application. Special care must therefore be taken.

With regards to the communication between all these modules, we follow a similar structure as can be found for the OSI model for internet communication. The model can be found in Figure 4.6.

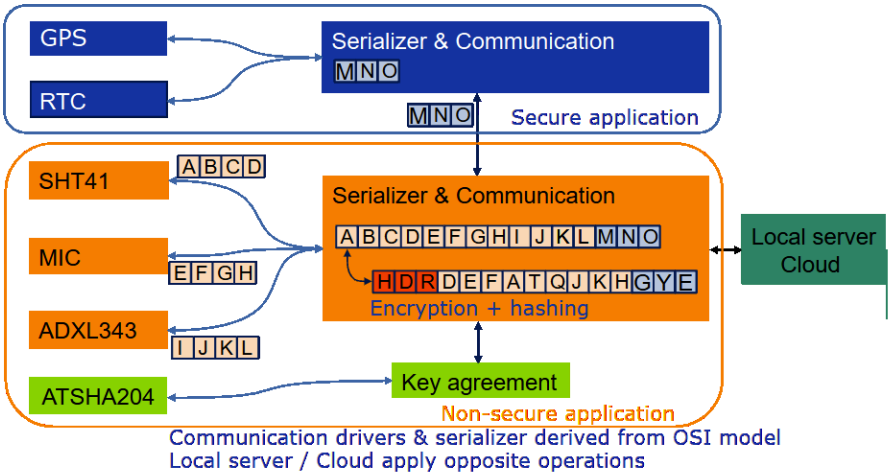


Figure 4.6: Schematic overview on the communication scheme between the low level drivers, the serializer and the remote server or Cloud.

Most of the communication happens in a very similar pattern as described earlier in this report. The only difference is that to communicate with the secure application, indirect

communication shall be preferred. In our case we communicate directly with a lightweight serializer in the secure application written in C.

Key Agreement Protocol

Secure communication between the end-parties is of utmost importance in order to guarantee data integrity and encryption. The encryption on the microcontroller can be achieved using a secret symmetric key which is known by both the embedded device and the remote server/computer. However, the key needs regular refreshing to guarantee an untrusted third party is unable to reverse engineer the encrypted data. This reverse engineering can be achieved via pattern searching in the many encrypted packets, or more directly by extracting the secret key from the firmware. To mitigate this risk, a key management scheme is set up which refreshes the symmetric key for AES encryption. The key agreement scheme should make use of the physical unclonable functions (PUF) principle to avoid generating predictable new secret keys. In most cases, however, PUFs are not integrated in currently available microcontrollers. To circumvent this, the true random number generator (TRNG) of the PIC32CM5164 microcontroller is used in combination with the ATSHA204A crypto-authentication module. This module allows to secretly store the generated random number while it can not be read-out by undesired third parties. The same crypto-authenticator module also allows to generate a new hashing leading to the generation of a new key. The key agreement protocol operates in 3 different phases:

- Phase 1 Registration: The embedded device is physically attached to a trusted third party and exchanges its unique identifier. The trusted third party generates a new secret key with the embedded devices and registers the device information into a database. The embedded device generates a new random number and stores it into the ATSHA204A. The ATSHA204 is then locked so that this random number can not be retrieved. Via this random number it is possible to mimic the behaviour of a PUF since each of the random generated numbers should be unique to each system since it relies on a true entropy source. This phase occurs once.
- Phase 2 Initialisation: The embedded device interacts remotely with a trusted server and exchanges a challenge and response. This pair of challenge-response will be required to start the last phase, the key agreement phase. This phase occurs once.
- Phase 3 Key Agreement: this phase occurs on regular intervals. The key agreement is established by exchanging a challenge and by locally generating a new response on both sides. In case the newly generated challenge-response pairs do match, the new challenges and responses replace the old ones. During this phase, based on the challenge and response, a new secret key is generated. This new key is then used to encrypt the communication between the two end-parties.

[UC4-1]	https://www.microchip.com/en-us/product/pic32cm5164ls00064
[UC4-2]	https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontr

	ollers/general-purpose-mcus/mcx-arm-cortex-m/mcx-n-series-microcontrollers:MCX-N-SERIES?tid=vanMCX-N-SERIES&cid=ad_pro240034g_tac1565250_gle_1&gad_source=1&gclid=CjwKCAjwufq2BhAmEiwAnZqw8mqYgoeT155T7rus6ezjcEcgaslVkLLHWEEssJZfKP4Rl7TsGCvxkhoCARcQAvD_BwE
[UC4-3]	https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc5500-arm-cortex-m33:LPC5500_SERIES
[UC4-4]	https://www.st.com/en/microcontrollers-microprocessors/stm32l5-series.html
[UC4-5]	https://www.st.com/en/microcontrollers-microprocessors/stm32u5-series.html
[UC4-6]	https://www.st.com/en/microcontrollers-microprocessors/stm32h5-series.html

UC 5: Cooperative robotics (GFaI)

System Specification

The specific application scenario involves developing a system that inspects workpieces in the industry. The conceptual system should consist of a server, multiple IoT devices such as robots, cameras, sensors, etc., and corresponding single-board computers. The server acts as the central hub of the system, while the PC can be used to monitor the current system status. The architecture of the IoT network should be dynamically expandable to allow devices to operate in plug-and-play mode. The data stream is tunnelled using SSTP (Secure Socket Tunneling Protocol) to establish a secure connection. Communication between each device and the server is hosted by an individual single-board computer (SBC). The overarching goal is to enable independent communication with all devices.

Threat Analysis

Identified vulnerabilities in the IoT system include the possibility of devices becoming vulnerabilities themselves, especially with inadequate security measures at physical interfaces and in firmware. The choice of communication protocol can also pose a security risk. As a countermeasure, implementing a VPN tunnel is proposed to ensure encrypted communication between devices in the IoT system and minimise potential threats. Additionally, a Trusted Platform Module (TPM) is considered for hardware security [UC5-1].

Requirements Specification

Implementing a secure communication protocol is crucial for creating an IoT environment with actuators, sensors, and single-board computers. Various protocols can be considered, including CoAP (Constrained Application Protocol) with DTLS or TLS for security, MQTT-SN (MQ Telemetry Transport for Sensor Nodes) for many-to-many communication on constrained devices, and SSTP (Secure Socket Tunneling Protocol) as the primary communication protocol in a VPN tunnel. In the present use case, SSTP has been selected, and the implementation was done using the open-source multi-protocol VPN program named SoftEther VPN [UC5-2].

The VPN system consists of a server and client. The VPN server offers six different ciphers for communication encryption, all with Perfect Forward Secrecy for maximum security. To ensure security in the client-server connection, individual certificate authentication is used. The VPN server includes a Virtual VPN Hub for creating individual client accounts, and there is the option to configure a Virtual DHCP Server to enable IP address distribution within a specific range.

The testing scenario to ensure encrypted message transmission involves a laptop, a robot, and two NanoPC-T4s. Specifically, the testing scenario with the robot requires real-time communication between the server and the robot. In a scenario where the VPN connection is interrupted, leading to data buffering, the robot cannot receive commands, resulting in the

robot being unresponsive. Therefore, reliable maintenance of the VPN connection is crucial for the smooth operation of the system.

Trusted Platform Modules (TPM)

Hardware security of a computer or device can be achieved by using a Trusted Platform Module (TPM). A TPM is a specialised chip embedded in the device's hardware, designed to manage cryptographic keys and protect the device.

Each TPM chip contains a key pair, known as the Endorsement Key (EK), managed on the chip and inaccessible to software. When the device is acquired, a Storage Root Key is created, allowing the user to generate additional security keys and establish encrypted communication. The Attestation Identity Key (AIK) protects the device from unauthorised changes to firmware and software by verifying critical sections using a hash process. This ensures that only authorised changes to firmware or software can be made.

Figure 5.1 illustrates a system architecture that integrates both the use of the TPM module with a Raspberry Pi and the use of the SoftEther VPN client.

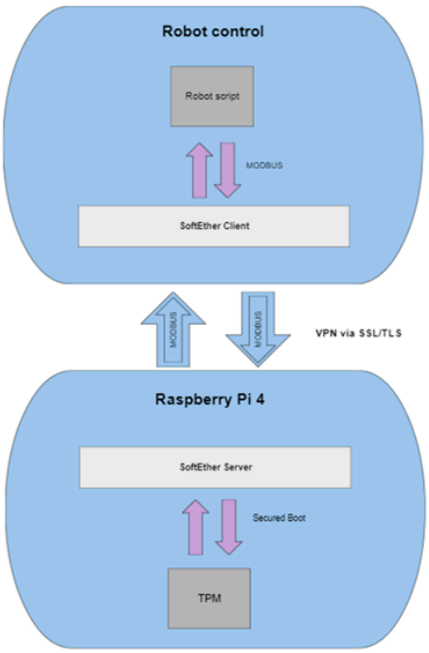


Figure 5.1: System Architecture with TPM and VPN Network

The TPM not only helps prove the user's identity but also authenticates the device. This prevents unauthorised access to the device and the installation of malicious software. Overall, the TPM enhances device security and protects it from potential attacks.

TPMs can be implemented in various ways:

- Discrete TPMs are specialised and dedicated chips, usually containing mechanisms to protect them from tampering.
- Physical TPMs are integrated into the main Central Processing Unit (CPU).
- Firmware-based TPMs run in the Trusted Execution Environment (TEE) of a CPU, nearly as secure as physical TPM chips.
- Software-based TPMs do not provide additional security and carry the risk of being vulnerable to errors or external attacks.
- Virtual TPMs retrieve security codes from a virtual machine.

TPM Software

To facilitate the management of the Trusted Platform Module, software from the chip manufacturer Infineon is used. This software, called "OPTIGA™ TPM 2.0 Explorer," simplifies the operation of TPM functions significantly [UC5-3]. It includes features such as data sealing with policy, remote attestation, cryptographic operations using the OpenSSL library, secure communication with the OpenSSL library, and provision of device certificates and onboarding in AWS IoT Core. Figure 5.2 shows the GUI-based software from Infineon.

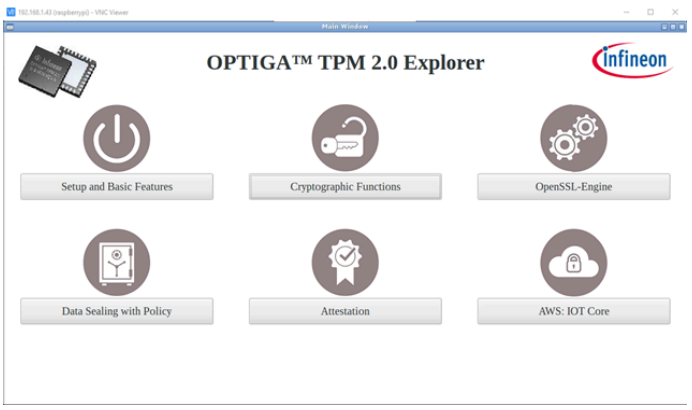


Figure 5.2: OPTIGA™ TPM 2.0 Explorer

U-Boot Bootloader

To fully utilise the TPM module, a U-Boot bootloader has been set up in the Raspberry Pi system. Through the so-called "second-stage bootloader," the user can boot and intercept the system before the kernel is loaded. The TPM module is already operational in the second-stage bootloader, allowing its full functionality to be used. Additionally, a serial-to-USB converter has been connected to the Raspberry Pi for monitoring the boot process from an external device to monitor the boot process and detect potential hazards early.

Application Scenario

To ensure the quality of various workpieces, they are photographed from predetermined perspectives and under defined lighting conditions using a camera. This enables a comparison with defect-free reference images to identify and assess potential irregularities. The comparison process involves three steps:

1. Identify and assign key points in the images.

2. Overlay the images based on keypoints and potentially rectify them.

3. Detect errors by differentiating the captured images.

To achieve an even distribution of viewing points along a hemisphere, an optimised form of the Fibonacci spiral is utilised. A Python script defines the diameter of the desired hemisphere, which can extend up to the maximum movement radius of the UR10. This allows for the generation of various hemispheres with different diameters, ensuring even distribution of points on the hemisphere. This facilitates the inspection of various objects for potential production errors.

The generated points contain coordinates/vectors [x, y, z, rx, ry, rz], describing each point and its position in space. These points create a hemisphere network of positions, which is then passed to the UR10, as seen in Figure 5.3.

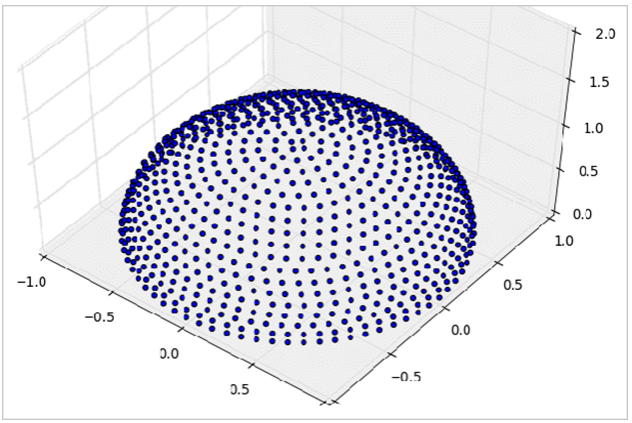


Figure 5.3: Hemisphere network of Positions

Universal-Robot

The Universal-Robot, or UR10, communicates via Modbus with the server and via a static IP address with the mounted camera [UC5-4]. This is necessary for the synchronisation of the entire system. The previously generated hemisphere network of positions is transferred to the UR10 via Modbus.

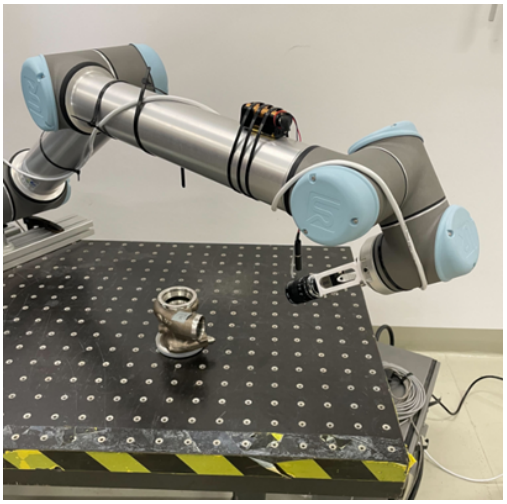


Figure 5.4: UR10 Image Capture

Figure 5.4 shows the positioning of the UR10 at position n on the hemisphere. Through communication and synchronisation of the entire system, the robot stops at this position until the image captured by the IDS camera is successfully completed. Afterward, the robot moves to point n+1 to generate a new image of the workpiece.

Image Processing

By using reference marks, also referred to as keypoints, in images, it is possible to locate image sections or even the entire image in a reference image. Keypoint descriptors such as ORB ("Oriented FAST and Rotated BRIEF") are stable against slight differences in exposure, viewing angle, background, or scaling of the image [UC5-5].

The resulting reference marks from the descriptor are used to identify predefined image sections from a reference image in another image. The ORB descriptor uses two methods for detecting reference features: Oriented FAST feature detection and Rotated BRIEF descriptor. The Oriented FAST algorithm detects features in an image by searching for corners with high intensity changes. The Rotated BRIEF descriptor then generates a binary description for each feature by selecting pairs of points in the feature area and evaluating their relationships.

When corresponding points are identified in two different images, a relationship between the images can be modelled, and the detected features in the images or the entire image can be overlaid based on the reference marks. This overlay is necessary because the used robot (UR10) does not have perfect positioning accuracy, resulting in images taken from slightly different angles, positions, and with different exposures. In the required homography of both images, the previously found corresponding points are transferred to the corresponding points of the second image using a 3x3 transformation matrix.

To perform a homography, there must be at least four corresponding points in the images. Reference marks not found in both images are not considered during the homography

operation. This allows distortions and rotations to be corrected using two images and this operation.

The next step involves warping the images. Warping involves considering images with depth values from different viewing angles using warping equations. Disadvantages of the method include possible occlusion and uncovering errors. The reference marks are then transformed using the warping. The transformation can be done analytically or linearly. At the end of the operation, both images are overlaid, with no change in pixel values at the corresponding points.

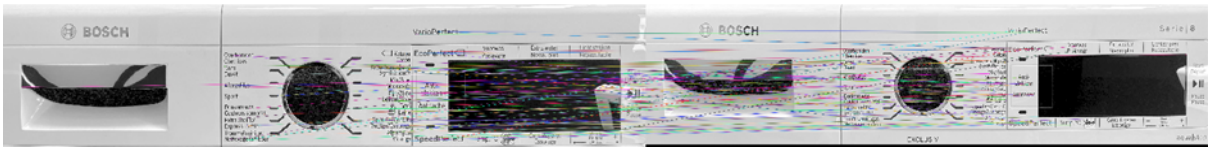


Figure 5.5: Key Points Representation using Washing Machine Blinds

Error Detection

During quality control, the errors of the workpiece are made visible by comparing the reference image with the image to be evaluated using the absolute difference between the two images. To enhance the clarity of the errors, the difference images can be eroded and dilated to eliminate deviations below a predefined threshold.

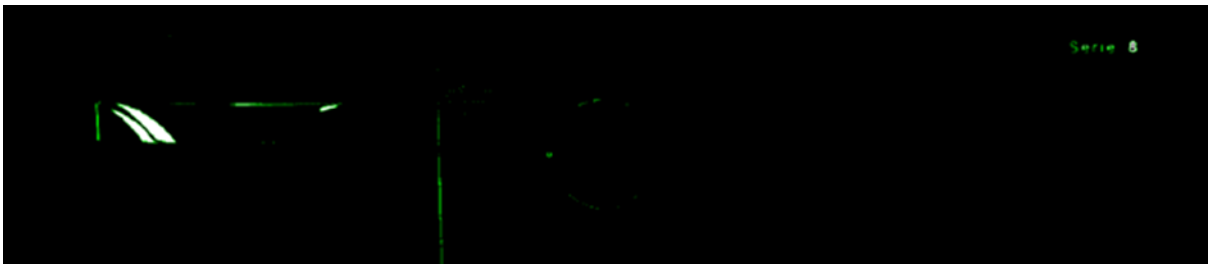


Figure 5.6: Error markings

Valuation

Critical Evaluation of TPM Use Case

Trusted Platform Modules (TPMs) have become increasingly vital due to the rise in cyberattacks, underscoring the need for robust security measures [UC5-6]. Standarized in 2009, TPMs are now standard components in computing systems, akin to CPUs and graphics cards. These hardware modules perform essential security functions such as securely storing encryption keys and executing cryptographic operations without heavily taxing system resources.

Advantages and Limitations of TPMs

TPMs are designed to handle cryptographic tasks efficiently, with minimal impact on system RAM and CPU, which makes them ideal for environments with limited resources, such as IoT devices [UC5-7]. They provide strong security capabilities by managing cryptographic keys and supporting secure boot processes, creating a hardware-rooted trust that is essential for device integrity and security. Their widespread adoption and proven effectiveness across various industries reduce the risks associated with using new or untested technologies, making TPMs a reliable choice for enhancing security.

However, integrating TPMs into existing systems can be challenging due to varying hardware and software environments, requiring careful planning to ensure compatibility. While TPMs are tamper-resistant, they are not entirely immune to physical attacks, and effective key management is crucial, particularly in large-scale deployments, to maintain security integrity. Additionally, although TPMs are designed for efficiency, their cryptographic operations can introduce some performance overhead, which needs to be considered, especially in low-power systems [UC5-8].

Conclusion

In summary, TPMs offer significant security benefits by providing efficient, hardware-based protection against cyber threats. It is essential, however, to address integration challenges, manage keys effectively, and consider potential performance impacts. By doing so, TPMs can be a valuable component of a comprehensive security strategy.

Critical view Use-Cases

The chosen use cases are just a few examples of the many potential applications for TPM modules. In our project, the TPM module is embedded in a Python server, where it handles critical tasks related to security, such as managing cryptographic keys, ensuring data integrity, and handling secure communications. We selected use cases that align with our personal interests in robotics, image processing, and artificial intelligence, aiming for fast and efficient results. This made our use cases a great fit for the Cornet Project.

Using TPM modules in this context is advantageous because they enhance the security of sensitive operations, ensuring that all data processed and transferred is protected from tampering and unauthorised access.

In Use Case 1, a UR10 robot, equipped with a camera mounted on its sixth axis, takes pictures of a given object from various angles. After capturing the images, the data is sent to the server, where the TPM handles the secure transmission and storage of this data, protecting it from potential interception or corruption during transfer. The TPM also manages the cryptographic processes required for secure communication between the UR10 robot and the server, ensuring that only authenticated devices can communicate and that data integrity is maintained.

In Use Case 2, a gripper is mounted on the UR10 robot's sixth axis. This gripper holds and transports a circuit board under a light dome, where images are captured for anomaly detection. Similar to Use Case 1, the TPM is responsible for securing the transmission of data between the robot and the server, managing cryptographic keys, and ensuring that all communications are secure and protected from tampering. The TPM ensures that the data collected for anomaly detection is securely transmitted and stored, safeguarding it from unauthorised access or manipulation.

Both use cases benefit from the TPM's capabilities in managing secure communications and ensuring data integrity, significantly reducing the risk of data breaches and enhancing the overall security of the project. While integrating TPM modules adds some complexity and cost, especially if there are compatibility issues with existing systems, the security benefits—such as improved data protection and secure communication—make TPMs a valuable component in applications that require a high level of data integrity and security.

Bibliography

[UC5-1]	https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.16.pdf
[UC5-2]	https://www.softether.org/
[UC5-3]	https://github.com/Infineon/optiga-tpm-explorer
[UC5-4]	https://www.universal-robots.com/de/produkte/ur10-roboter/
[UC5-5]	https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html
[UC5-6]	https://www.intel.com/content/www/us/en/business/enterprise-computers/resources/trusted-platform-module.html
[UC5-7]	ISO/IEC 11889-1:2015
[UC5-8]	https://trustedcomputinggroup.org/about/what-is-a-trusted-platform-module-tpm