Embedded multi-core systems for Mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D10.5 Final design and implementation (completed and final demonstrations)**	
Work package	WP10	Industrial Applications and Logistics
Task / Use Case	Tasks	T10.1, T10.2, T10.3, T10.4
Lead contractor	Infineon Technologies AG Dr. Werner Weber, mailto: werner.weber@infineon.com	
Deliverable responsible	Danfoss Power Electronics A/S Dr. Juha Kuusela, mailto: juha.kuusela@danfoss.com	
Version number	v1.0	
Date	28/12/2016	
Status	Final	
Dissemination level	Prototypes public	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
01R	NXP	Kiran Shekhar	T10.2
05A	Danfoss	Juha Kuusela	T10.1
09B	UTRC	Juan Valverde Alcala, Stylianos Basagiannis	T10.1
15Q	ITI	Javi Cano	T10.4
04A, 04B	BUT, NXP CZ	Petr Blaha	T10.1
15R	AMBAR	Ignacio Ara Isusi	T10.3

Document History

Version	Date	Author name	Reason
v0.1	30/11/2016	Kiran Shekhar (mailto: kiran.shekhar@nxp.com)	Initial version, content for T10.2
v0.2	30/11/2016	Kiran Shekhar (mailto: kiran.shekhar@nxp.com)	Integrated review comments from Juha Kuusela (WP10 leader) to T10.2
v0.3	12/12/2016	Juha Kuusela (mailto: juha.kuusela@danfoss.com)	Added initial version of T10.1
v0.4	12/12/2016	Stylianos Basagiannis (mailto:basagis@utrc.utc.com) Juan Valverde Alcala (valverj@utrc.utc.com)	Added T10.1 UTRC contribution
v0.5	13/12/2016	Javier Cano (mailto: jcano@iti.es)	Added contents of T10.4
v0.6	18/12/2016	Juha Kuusela (mailto: juha.kuusela@danfoss.com)	Composition, format, editing and conclusion
v0.7	19/12/2016	Javier Cano (mailto: jcano@iti.es)	Updates to T10.4
v0.8	20/12/2016	Petr Blaha (mailto: Petr.Blaha@ceitec.vutbr.cz)	Updates to T10.1
v0.9	22/12/2016	Juha Kuusela (mailto: juha.kuusela@danfoss.com)	Updates to T10.1
v0.95	22/12/2016	Ignacio Ara Isusi (mailto: iara@ambar.es)	Added contents to T10.3
v1.0	28/12/2016	Juha Kuusela (mailto: juha.kuusela@danfoss.com)	Language and style
	28/12/2016	Alfred Hoess	Project internal review; final editing and formatting; deliverable submission

Publishable Executive Summary

The ultimate objective of the EMC^2 project (and WP10 in that context) is to establish Multi-Core technology in all relevant Embedded Systems domains. The embedded systems segment is currently going through a disruptive innovation process. Different kinds of systems are connected to each other, boundaries of application domains are alleviated and interoperability plays an increasing role. Formerly closed systems are forced to be opened up. As Multi-core and Many-core processors increase their visibility in the embedded systems domain, their exploitation for critical and real-time applications is presently too slow, inefficient and expensive.

This development leads to the fact that system components that have previously been embedded and executed in separate hardware can now share the same hardware, thus resulting in new challenges for safety requirements. At the same time multiple cores potentially allow different types of functions to stay separated where necessary.

This document gives a brief description of final prototypes and design of all the four use cases of WP10 within the industrial domain (or Living Lab 10 for Industrial manufacturing & logistics). The four use cases represent four different industrial applications: (1) Variable Speed Drives in Industrial Applications, (2) Identification and Authentication, (3) Tracking and (4) Manufacturing Quality Control by 3D Inspection.

The final prototypes are publicly demonstrated.

Table of contents

1. Introduction

1.1 Objective and scope of the document

This document describes briefly the final design & implementation and eventually demonstration for all 4 use-cases of WP10 Industrial Manufacturing and Logistics. It is part of project milestone 8 (MS8) that is intended to be delivered in project month (M33).

1.2 Structure of the deliverable report

The document is organized as follows: the sections 2, 3, 4 and 5 present the four use cases in WP10. Finally, section 6 concludes the deliverable.

2. T10.1 – UC_Drives and Electric Motors in Industrial Applications

2.1 Description of the final prototypes for the Drives Use Case

Industrial processes place strict requirements on the automation systems. Electric motors and Variable Speed Drives (VSD) that operate these motors – often just called “drives” – are the backbone of industrial automation systems. A typical industrial application is automated and controlled using multiple devices. In general, each such device has one or more mechanical parts, which are driven by an electric motor via a transmission system. VSD can give precise control over speed, torque, and position, which can then be matched to the process under control, yielding energy savings and removing the need for complicated mechanical control devices.

It is typical that in addition to safety requirements – a characteristic of numerous automation systems – VSDs must satisfy other quality related requirements. For instance, a crane manufacturer must also fulfill performance (e.g. vertical speed), user comfort (e.g. smooth acceleration), customer expectations (e.g. ability to serve 50 meters’ vertical height), and reliability (e.g. 2 hours’ maximum maintenance time per year) requirements.

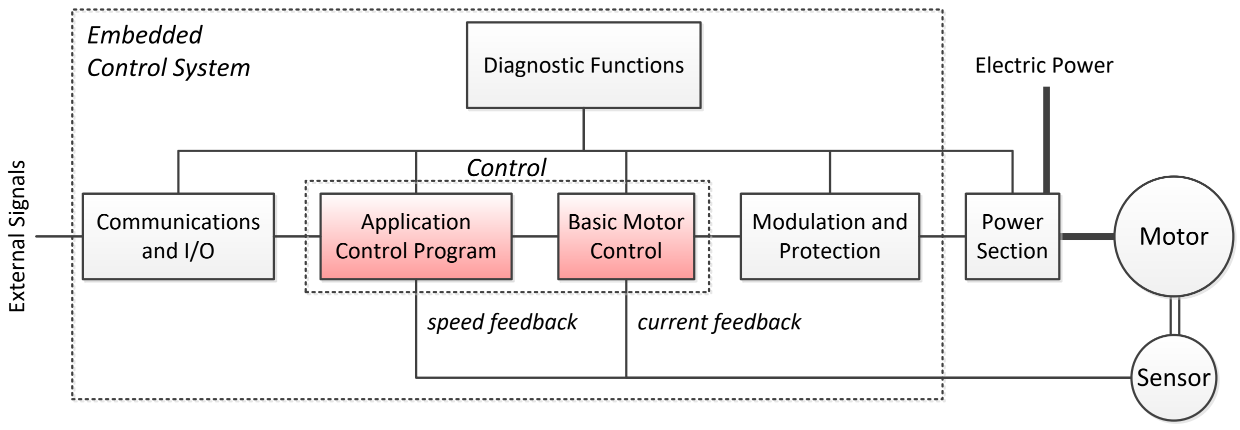


Figure 1 - Functional elements of a VSD

A VSD (see Figure 1) follows a standard high-level design, which includes an Embedded Control System that: communicates with external controllers via the External Signals; actuates the Motor via the Power Section, to fulfil the application requirements and requests received via External Signals; and finally, uses current and speed feedbacks from Sensors to the control system. The functional elements of the Embedded Control System are divided into two groups: safety-related functions, and the VSD application itself. Therefore, the embedded controller encompasses mixed criticality ranging from Safety Integrity Level 1 to 3 (SIL1-3). The safety-related functions consist of drive-based safety functions defined in IEC 61800-5-2, and protection functions defined in IEC 61800-5-1, UL 508C, and UL 61800-5-1. The application control could be split into two domains the Application Control Program (ACP), and Basic Motor Control (BMC). The ACP typically models the application, e.g., conveyer belt, fan, pump, etc. Depending on the application being controlled, the ACP has varying real-time requirements ranging from soft to hard real-time. The BMC is responsible for handling the operation of the electric motor with hard real-time constraints.

VSDs are typically implemented on heterogeneous hardware consisting of dedicated processing elements, like MCUs, DSPs and FPGAs. A commercial off-the-shelf System-on-Chip solution is a suitable VSD hardware platform, as it provides the necessary processing performance with high-level integration resulting in lower costs.

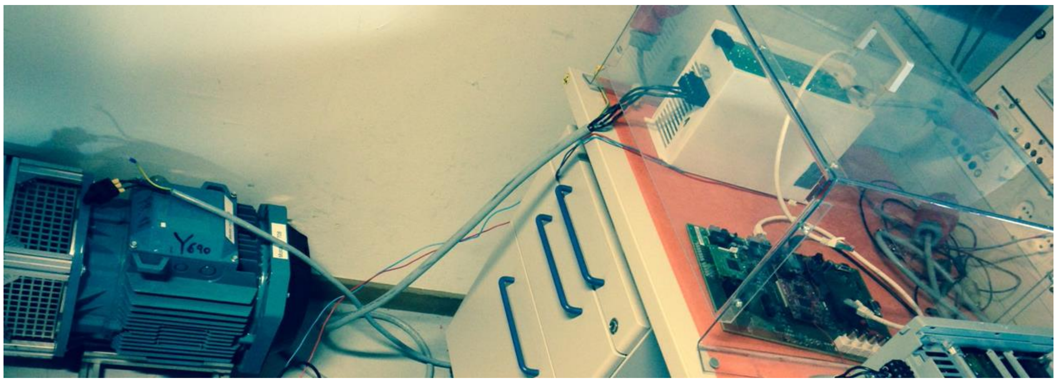


Figure 2 – VSD prototype hardware

VSD prototype is built based on these concepts (see Figure 2). It uses Zync 7010 to run the control software and the FPGA array on that SoC to implement real-time communication protocol between the control card and power card. It also uses the FPGA to provide distributed real-time clock and basis for the functional safety solution.

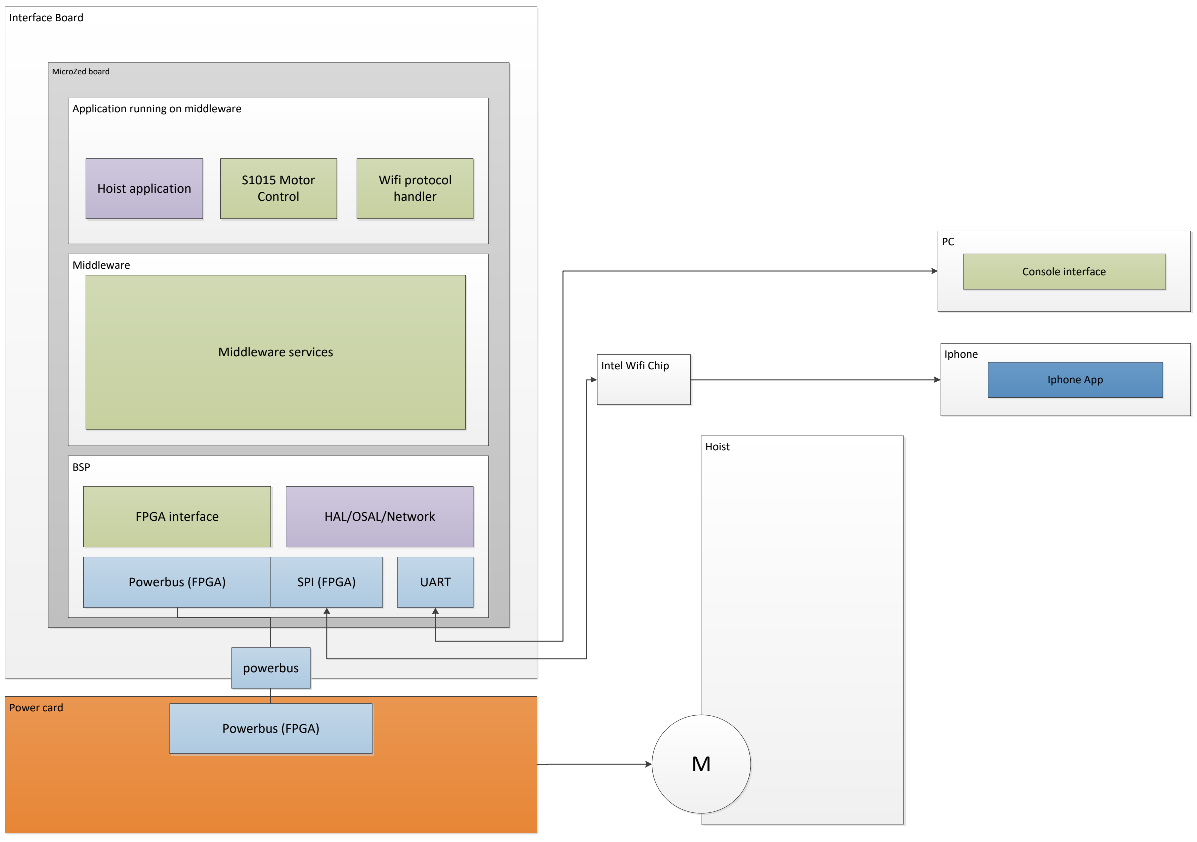


Figure 3 - Deployment of the VSD demonstrator

Figure 3 shows the functional deployment of the VSD demonstrator. The demonstration application is hoist. As a user interface we used iPhone App connected to the VSD with WiFi.

This demonstrator only has simple model based motor control. UTRC-Ireland extended the use of model based design beyond motor control to fully take advantage of the automatic code generation possibilities offered by Mathworks Simulink. For that purpose, it was necessary to primarily set a reference design that includes all the control modules for these interfaces in Vivado (Xilinx development tool), so that the HW created by Mathworks HDL coder can be automatically connected to the external components.

HDL coder will create VHDL or Verilog code for some of the model blocks in the system, while Embedded Coder will create C code for Linux, for those algorithms intended to run within the ARM cores. At the same time, it is possible to set a real-time SW model to interact with the control of the motor from the Simulink model via Ethernet with the computer. An image of the whole system can be seen in Figure 4.

By manually adjusting the automatically generated code for Linux provided by Embedded Coder, it was possible to automatically map the different algorithmic blocks created in Simulink to the two different ARM cores in the Zynq-7000 platform. At the same time, it is possible to directly configure at model level aspects like the way the different tasks exchange data, the periodicity of the tasks, or the core affinity, among others. Real-Time Linux cannot guarantee isolation between critical and non-critical tasks. For that reason, the demo was updated using the automatically generated code with a hypervisor to specifically address the needs of mix-critical systems.

The selection of the hypervisor is a critical task for core temporal isolation. All the available hardware resources need to be mapped and partitioned from a hypervisor point of view in order to safely control their real time operation when accessing shared resources. Time partition with respect to evaluation of critical time-based requirements of the motor control should be defined as well, as an additional control layer. The concept is initiated through year 1 of EMC2 project where we verified simple scheduling algorithms based on traffic-light semaphores running in one of the cores in a master-slave setting. In order to achieve this, we aim in the future to introduce certified real-time operating systems, handling both time and resource partition in an efficient way from engineering point of view.

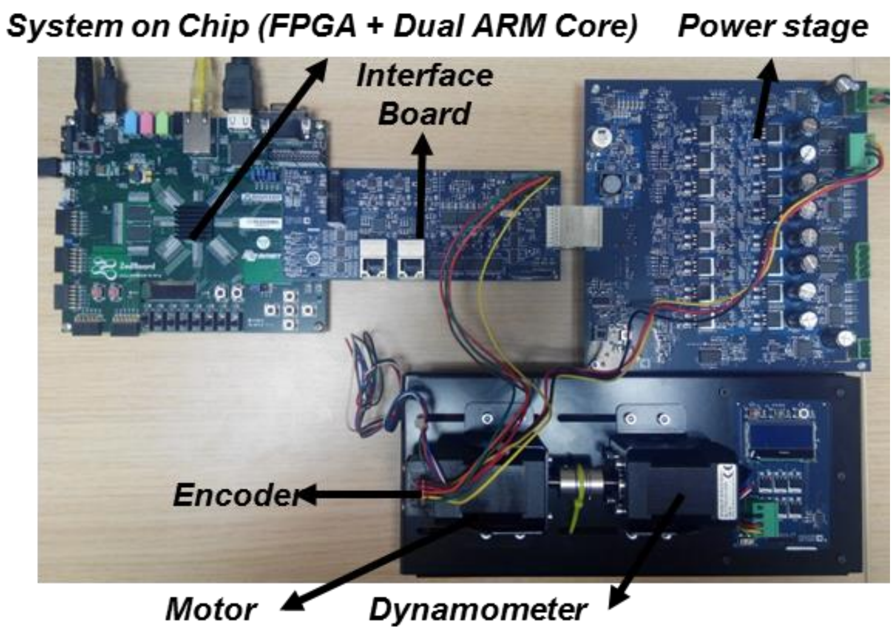


Figure 4 – Test set-up for fully model driven control for the UC1 demonstrator

The selection of which algorithms are to be implemented in SW or HW depends on the application and its requirements. The main idea behind this application is allowing independent execution of tasks with different levels of criticality. For that, it is necessary to allow concurrent execution in both ARM cores without conflicting, as well as deciding which algorithms are more suitable to be implemented in the FPGA for pure parallelization and minimizing interfaces and possible data collision.

2.2 Description of the conceptual architecture

Figure 5 shows the control electronic concept of the demonstrator.

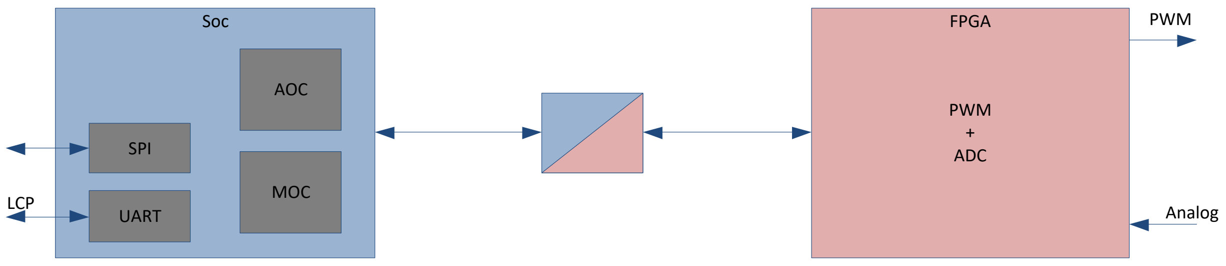


Figure 5 - Control electronic concept of the UC1 demonstrator

The conceptual architecture of the first VSD prototype supports real-time performance characteristics using a single multi-core chip (MCU and FPGA).

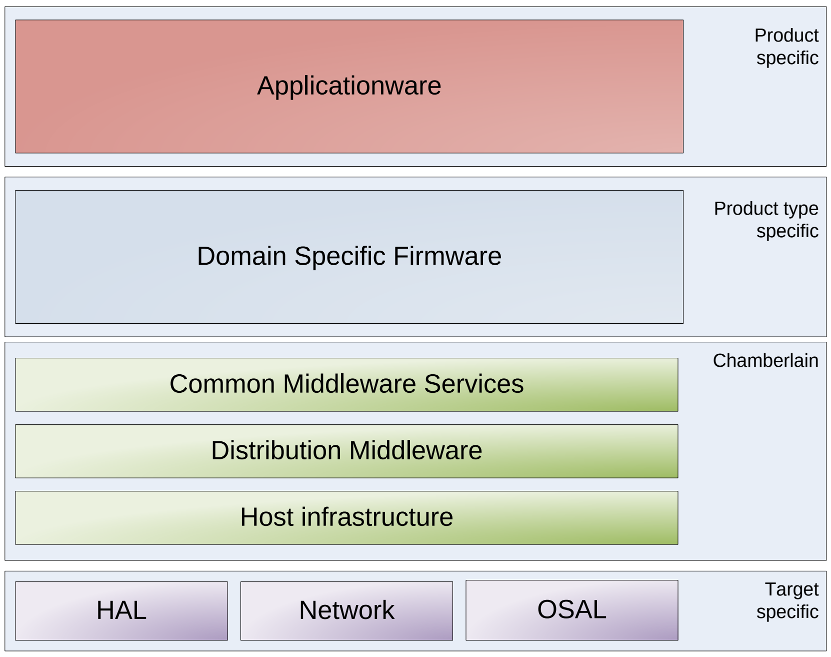


Figure 6 - Layered software architecture of the VSD demonstrator

Software stack has been layered according to expected variation (Figure 6). Services on the lowest layer vary with hardware. Hardware abstraction layer (HAL), Network layer and Operating system abstraction layer (OSAL) hide this variation for the rest of the system. Chamberlain middleware layer provides uniform way to handle distributed computation and different timing requirements and provide a set of commonly used services. Domain Specific Firmware contains firmware part of control functions and Fieldbus interfaces. This is expected to vary between different product types (AFE, INU). Application-ware contains application package, customer software, and all other product specific functionality. It can vary according to the product or customer.

The middleware layer abstracts the communication and co-ordination required by the distributed hardware. The main reason for having a separate middleware layer is to abstract away from different hardware on nodes, different hardware configurations, and different software deployments. This makes it possible to develop applications and services so that they can be deployed across several different products. This architecture also demonstrates declarative components and flexible deployment. Binding of functionality takes place during the system installation or at runtime.

The architecture is based on components with well-defined interfaces. Components communicate using a blackboard. The schedulers provide event-driven and time-triggered execution environments.

- In the time triggered execution environment computations are periodic. There can be more than one execution queue running at different frequency. Scheduling within the queue is co-operative. Scheduling between queues in pre-emptive. Control flow is done through ordering of computations.
- In the event driven execution environment components are tasks. Computations are started by events (inputs, timers). Since scheduling is pre-emptive it is necessary to protect shared resources.

The different nature of these execution environments means that components as a rule cannot be without modifications moved between these environments.

Modulation subsystem is responsible of generating required waveforms on the outputs. The control principles (Flux vector control, voltage vector control or active front-end) generate a voltage vector reference, which is passed to the modulation subsystem. The ambition is to define a feedback handling concept where the high-level functionality is common for both low power drives and high power inverter units, and still allow for hardware variance in the low-level functionality.

The high-level ambition is to create a flexible solution that allows for a variety of modulation schemes such as space vector modulation and any type of Discontinuous Pulse Width Modulation, Optimal Pulse Patterns including Selective Harmonic Elimination (SHE), multi-level inverters, and interleaved DC/DC converters.

Paralleling is taken care by the modulation subsystem (Figure 7). The modulator calculates the switching times $\boldsymbol{T}_{\mathrm{uvw}}$, which are broadcasted to all paralleled power modules. Each power module receives in addition to this a specific power balancing message $\boldsymbol{T}_{\mathrm{uvw,add,n}}$ from the paralleling control.

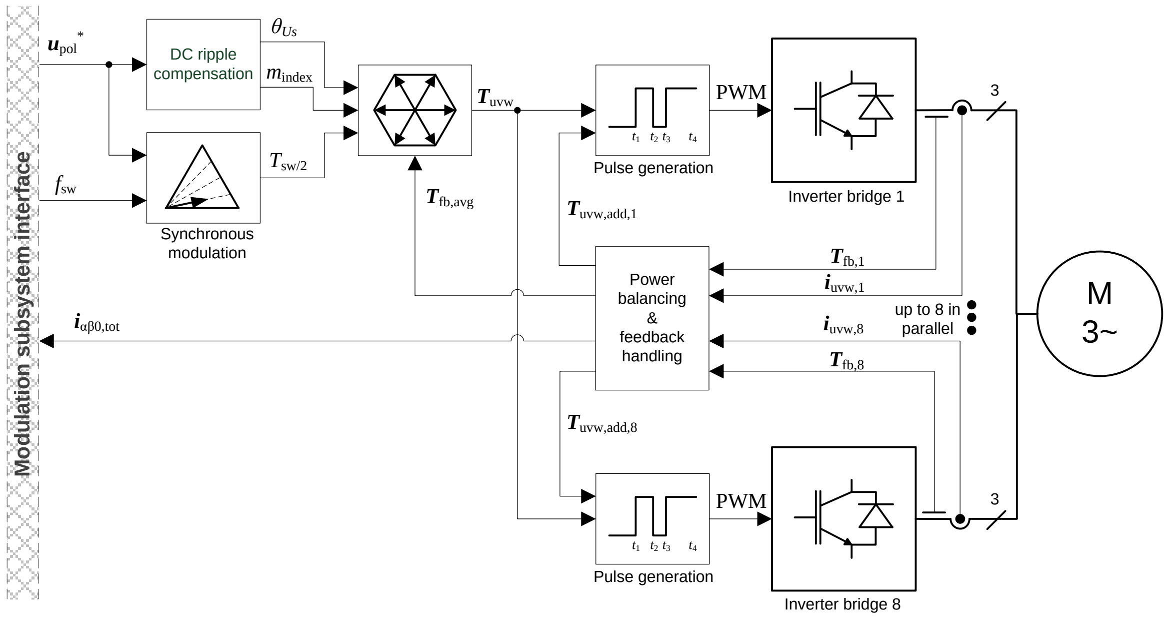


Figure 7 - Block diagram showing parallel control

Paralleling has strict real time requirements and modulation is mostly handled in the FPGA. Also the internal communication bus has to be real-time with known delays and low jitter. This architecture achieves this by handling also the communication in FPGA.

With the different execution domains in the middleware we have been able to keep real-time requirements while at the same time supporting high hardware utilization rates.

Motor control uses this modulation subsystem interface and is independent of the paralleling. In the motor control architecture, we have been striving for model based modular motor control. This means that control mode selection, control cores and auxiliary functions are modularized and separated form reference chain handling (see Figure 8).

Together with our partners in WP10 we have been building model based motor control. This has been very successful. We can now deploy the control from the modeling tool into running prototype set-up in less than 20 minutes. In parallel we have also built a set of motor models. Motor models are used in automating regression testing. The number of different motor types is very large and it is not feasible to test control systems against physical motors in large power sizes. Model based testing will significantly improve test quality and coverage.

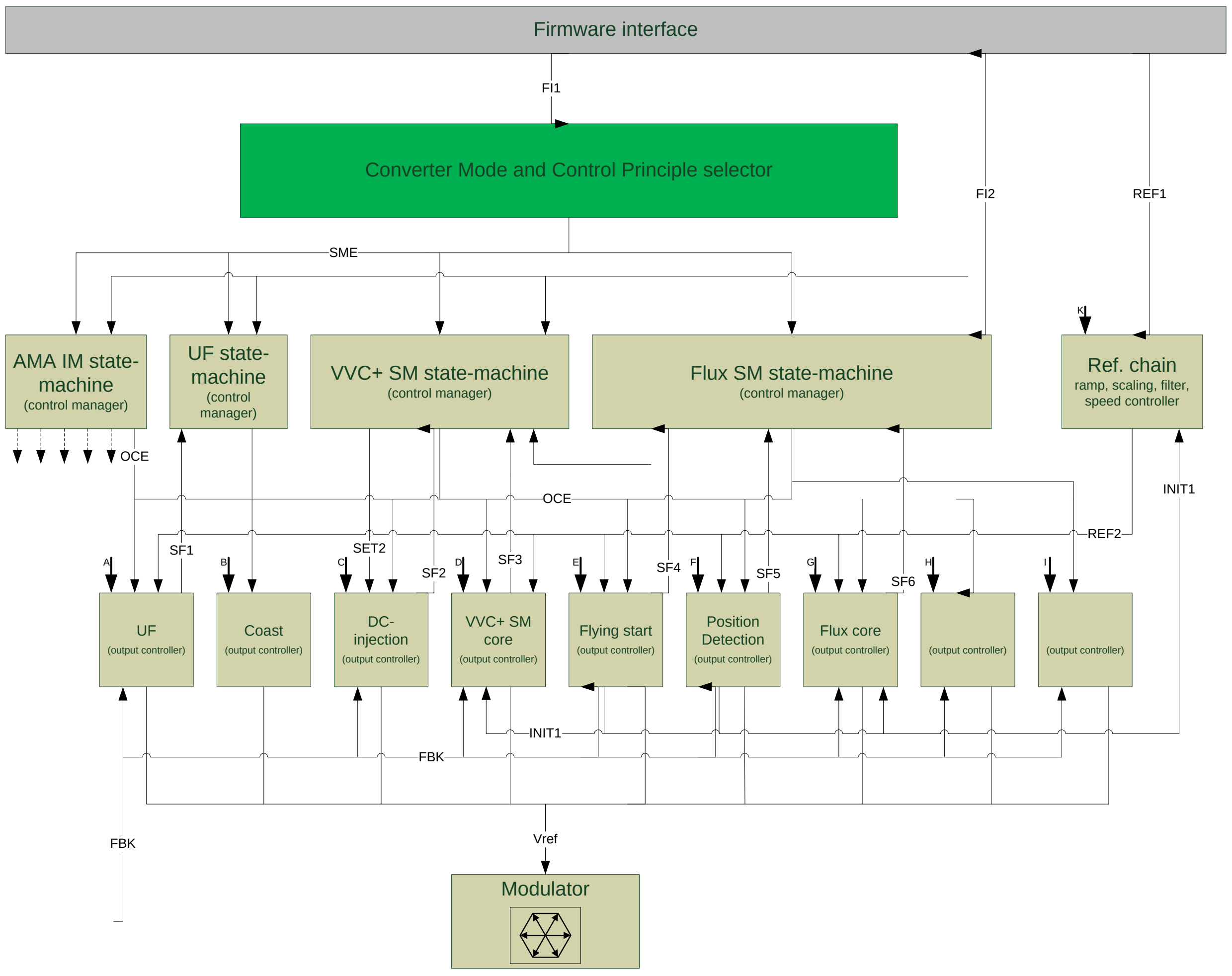

Firmware interface
FI1
Converter Mode and Control Principle selector
FI2
REF1
SME
AMA IM state-machine (control manager)
UF state-machine (control manager)
VVC+ SM state-machine (control manager)
Flux SM state-machine (control manager)
Ref. chain ramp, scaling, filter, speed controller
K
OCE
INIT1
REF2
SF1
SET2
SF2
SF3
SF4
SF5
SF6
A
B
C
D
E
F
G
H
I
UF (output controller)
Coast (output controller)
DC-injection (output controller)
VVC+ SM core (output controller)
Flying start (output controller)
Position Detection (output controller)
Flux core (output controller)
(output controller)
(output controller)
FBK
Vref
Modulator

Figure 8 - Some motor control components

The latest prototype has a more complex motor control and also implements flux control. However, we do not yet have encoder feedback built in and the prototype does not completely fulfill the requirements like torque control (Requirement **05A-4_1_1**) or speed feedback (requirement **05A-4_1_2**).

UTRC studied a possibility to bring model based development further and use model driven design also for FPGA. The processing architecture also in this case consists of 2 ARM cores and an FPGA fabric. One of the ARM cores will be in charge of running the control of the operation mode of the motor, as well as the speed control algorithm. The other core will run some Prognostic and Health Management (PHM) control algorithms. On the FPGA side, data acquisition, encoder data transformation, current control and PWM generation will be in included. Some of the PHM algorithms will need some data pre-processing. This PHM extra processing tasks, such as a voting systems, will be included in the FPGA to take advantage of parallelization. The block diagram of the system is shown in Figure 9.

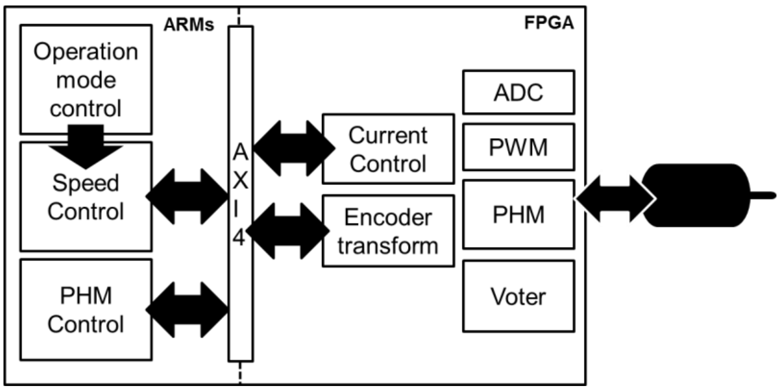


Figure 9 - Block diagram in fully model driven study

Models were developed in Simulink both for FPGA (Figure 10) and for the two ARM cores (Figure 11).

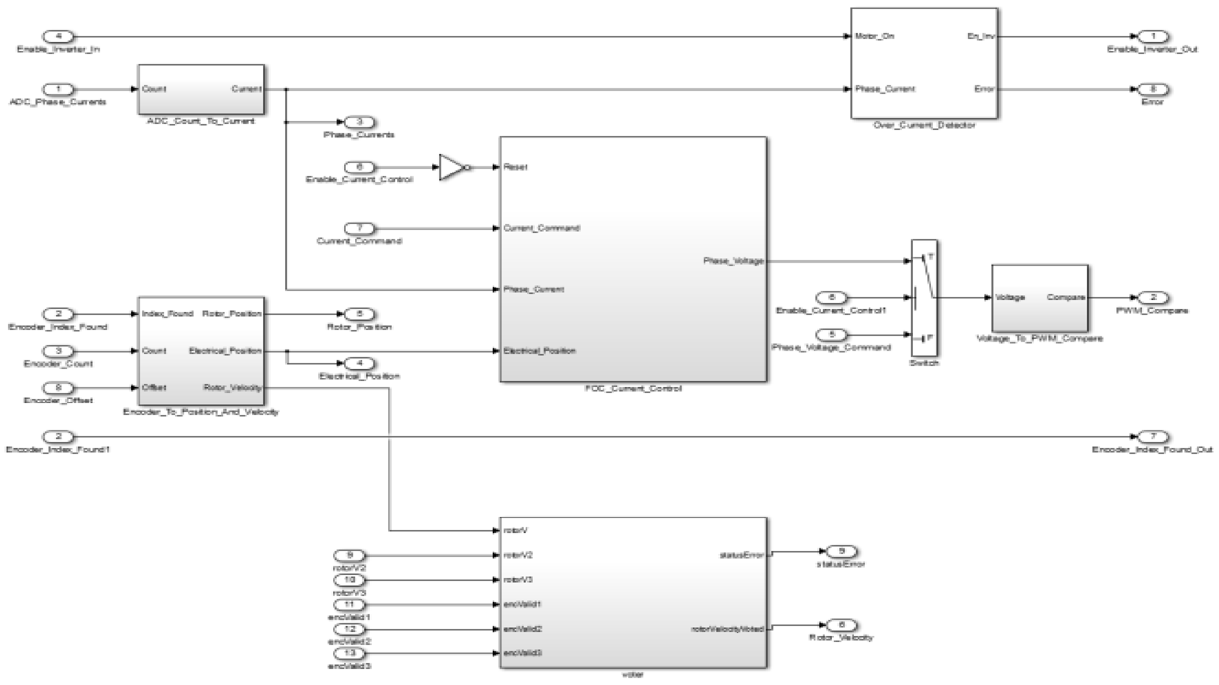


Figure 10 - Models for the FPGA-HW implementation

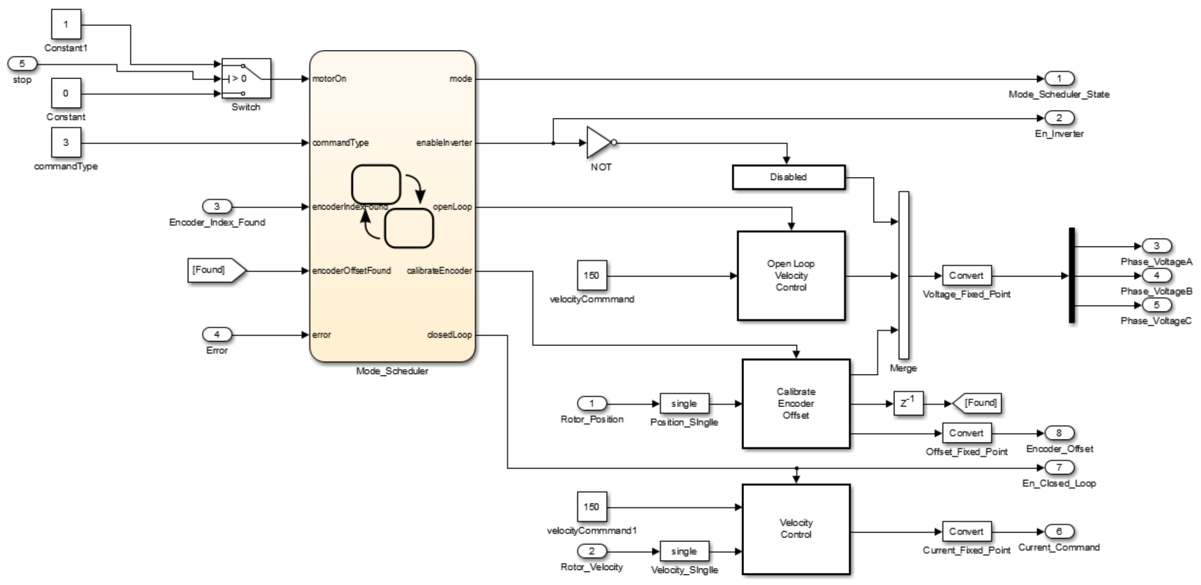


Figure 11 - Models for the SW implementation in the ARM cores

In order to specify concurrency within the tasks scheduled in SW, Simulink allows a task partition and mapping to allow concurrency. Figure 12 shows the different tasks marked with colored blocks.

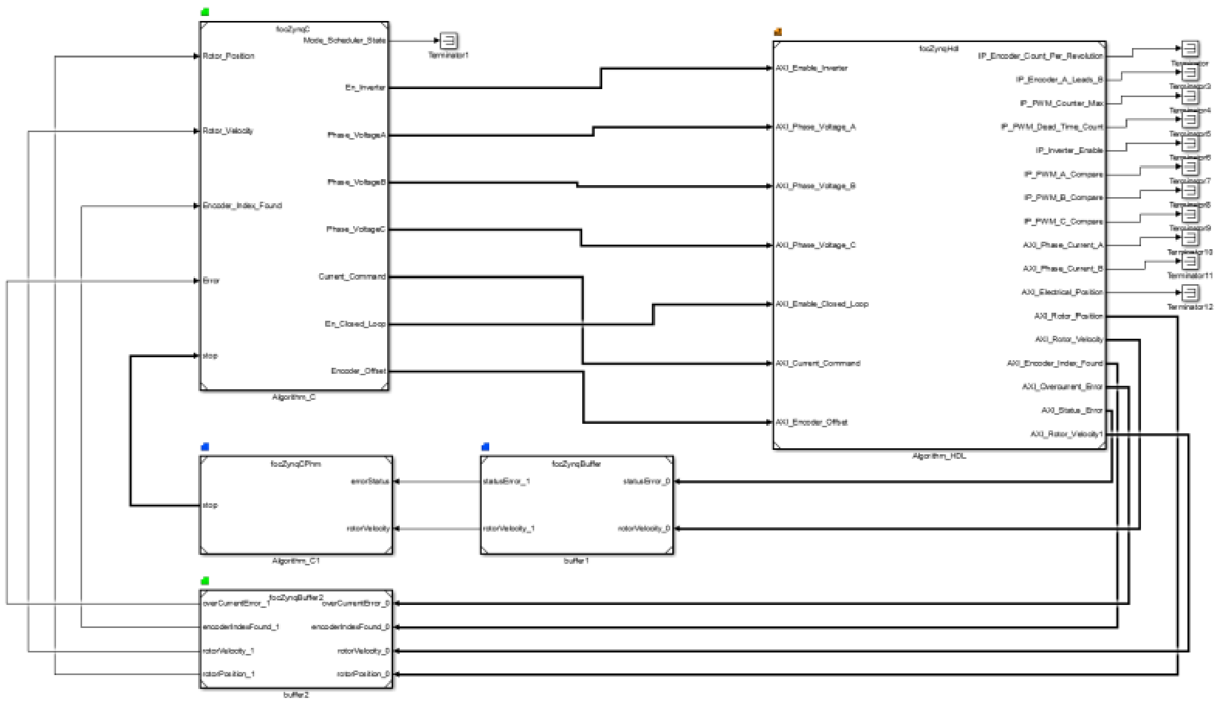


Figure 12 - Task partition and mapping

The general workflow followed for the design and prototyping is:

1. Build the models of the desired algorithms in Simulink using models of the motor and loads.
2. Generate a reference design for the FPGA system to include those HW parts that will not be defined in Simulink, like for instance, interfaces.
3. Simulate the models choosing a configuration as close to reality as possible.

4. Select weather they will be implemented in SW or HW. Take into account data types, algebraic loops, fix point and mathematical operations.
5. Incorporate a hypervisor in order to resource partition the available HW resources. Hypervisor should be previously validated and verified for conflict resolution between HW resources.
6. Simulate again taking step 5 into consideration.
7. Select, apart from the SW/HW partition, the desired partition and mapping of the SW tasks so they can be executed in a concurrent way. Be careful with data transmission among tasks.
8. Use HDL Coder and Embedded Coder to generate both C and VHDL code.
9. Use Xilinx SDK to compile the C code for Linux and Xilinx Vivado to synthesize the VHDL code and generate the bit stream configuration file.
10. Generate a model for the real time interaction with the motor.

This study demonstrated that it is possible to support fast prototyping of motor control systems in heterogeneous multiprocessing environments.

In the first prototype we showed that the integration of Safe Torque Off is relatively straight forward (see Figure 13) as it is possible to bypass the software layers to reach SIL3, PLe, Cat ¾ solution. In this prototype we have developed concepts for supporting STO for paralleled power cards. We have also developed concepts for other safety functions and communication over safe fieldbus.

Mixing safe and non-safe functions is problematic from several perspectives. Safety is supported by redundancy and diagnostics. Redundancy increases costs and those costs are acceptable only when the functionality is needed. This has led to a conceptual architecture which only supports STO as a standard safe function. All other safe functions are optional.

Other safe functions include speed related functions like safe limited speed. This can be supported without additional encoders if the power card has required functionality. In the new architecture this is provided by a small optional element. Additional communication channel between the power cards and control of safe functions is not needed if the communication channel can be monitored. This is possible using so called "Black Channel" terminated in the power card option and in the functional safe option.

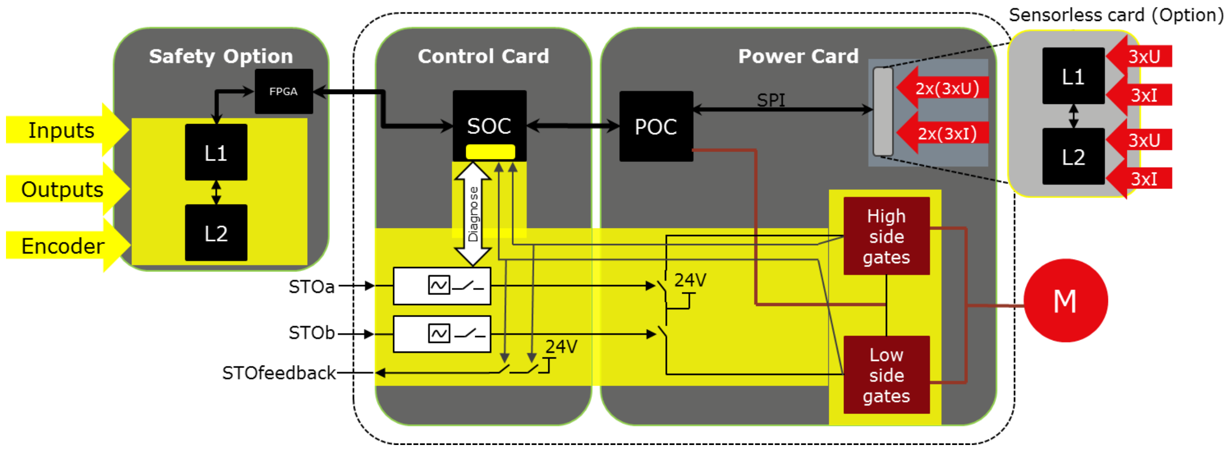


Figure 13 - STO functionality bypassing the software layers

We have also tried to define a common development process for safe and non-safe software. This does not seem to be possible. Requirements are too different. Safe software has to be reviewed and changes strictly controlled. Rest of the software is under competitive pressure to be changed even more frequently that before.

2.3 Multi-core communication using RPMsg protocol

In this part, the multiprocessor design enabling communication between individual processors was integrated on ZedBoard device. It was not initially assumed to be the part of this task but the target Zynq platform being used in it was found interesting to present functional demonstration of Remote Processor Messaging (RPMsg) which was developed by NXP Semiconductors in Czech Republic in WP3, namely in task T3.2, T3.3 and T3.7. With this, connection was strengthened between technological WP3 and this Living Lab.

Two MicroBlaze processors within FPGA part of Zynq device work together. Both processors execute program from their own local memory, which is situated in block RAM memory (BRAM) within the FPGA part. All FPGA designs were synthesized in Xilinx Vivado 2016.3 The RPMsg Lite protocol was implemented as a software for two MicroBlaze processors and compiled in Xilix SDK 2016.3. Both processors also use the common memory which is also situated in block RAM memory within FPGA part. The RPMsg Lite protocol is meant to be a solution for Mixed critical systems applications because of its feature to split and instantiate the system into independent block or subsystems. The design uses wide FPGA flexibility of XC7Z020-1CLG848C device to demonstrate mixed-criticality system solution. The RPMsg can be the basis for embedded Remote Procedure Call (eRPC) which enables to execute code on different places, in different cores or processors.

Achieved results on this implementation were collected in a conference paper for publishing in IEEE ICIT 2017 conference proceeding [ICIT_2017]. The paper is in review process.

2.4 PMSM model predictive control with field weakening implementation

One of the modern approaches to drive control is model predictive control (MPC). While the idea of MPC itself is relatively old, only the recent development of controllers with higher computational powers enables its implementation in systems with fast dynamics, such as electrical drives. An MPC controller includes a model of the plant that is used to predict the behavior of the system, and the related optimal state-space control problem is resolved with optimization methods. The great advantage of MPC is a proper constraints handling, as well as the possibility of defining control objectives in a very natural way. The biggest challenge in MPC implementation for AC electrical drive control is the nonlinear behavior of the drives. A single linear multiple-input multiple-output predictive controller was developed here which replaces both the current and speed controllers. The control algorithm is based on an explicit MPC, in which nonlinear terms are handled in such a way that allows it to perform optimization of the magnetic flux, including flux weakening in a high-speed region. The intrinsic implementation of field weakening is one of the main advantages of the proposed algorithm.

The algorithm was implemented on a National Instruments CompactRIO (cRio) system, which combines a field-programmable gate array (FPGA) and a real-time (RT) controller. The proposed design uses the benefits of the RT and FPGA combination to face the large memory demands of a linear explicit MPC implementation. The results of the successful experiments on a real PMSM (Permanent Magnet Synchronous Motor) can be found in journal paper [IEEE_TIE_2016] which was published in IEEE Transactions on Industrial Electronics. The work is in progress on implementation of this control algorithm on a Zynq hardware.

2.5 Properties of the prototypes and relation to technical WPs

The goal of this prototype is to verify the conceptual architecture both for HW and SW. It shows that the real-time communication works with the middleware. The correctness of the drive operation depends on the timely execution of its functions (**05A-1_0_4**). The demonstrator shows that we have a solution on a predictability supporting this requirement (related to WP3). Our solution is made so that we will be able to use OSSS-MC technology developed by OFFIS in WP2 tasks T2.2 and T2.3. We have conducted a series of measurements. Based on those measurements, jitter in the real-time control stays below 3.85µs under maximal asynchronous load. The main sources of this jitter are data cache invalidation (~2.0µs) and instruction cache invalidation (~0.24µs). Total jitter is way below the requirement of 10µs.

The requirement **05A-1_2_0** specifies parallelism in software but also in hardware using FPGAs. The demonstrator shows that functionality can be deployed for both cores and the FPGA. Development also showed that FPGA software can be developed flexibly and iteratively. Tool support is based on WP2. Successful core to core communication and FPGA based real-time communication bus between computation units show that we have fulfilled this requirement. Jitter on this communication bus is 10ns per switch assuming that there are no higher priority messages that will block the communication.

IEC 61800-5-2 specifies the designated safety functions (**05A-1_1_1**) that are relevant to variable speed drives for the industrial domain. These safety functions and their constraints have to be taken into account when architecting the drive. In this use case, the safety function Safe Torque Off (STO) is integrated as the standard safety function. The STO (**05A-4_3_1_0**) ensures that power which could otherwise cause a rotation of the motor is not applied. We are able to fulfill the safety requirements with the conceived architecture and preliminary feedback from certification authority indicates that some mixed criticality is accepted. We have not been able to fulfill our own goal of highly flexible or even programmable safety solution.

The requirement **05A-1_1_3** specifies that temporal and spatial separation between software elements of different criticality shall be achieved, allowing these elements to co-exist on the same device such that the non-safety related functions can be modified without impacting the certified safety related functions. This requirement is only partially satisfied in this prototype. With black channel communication we are able to share the communication channels. We are working to expand the conceptual architecture for this mixed criticality requirement (**05A-1_1_4)** using the modular safety framework developed in WP6.

The requirement **05A-1_0_3** specifies that the drive function binding can be performed at different times, allowing a certain degree of adaptation to the system. This architecture demonstrates declarative components and flexible deployment. Binding of functionality takes place during the system installation or at runtime. We are in the process of evaluating different deployments. We have been able to demonstrated that different software domains can used different binding strategies. We are also able to support flexible production delaying binding to mini factories close to customer (late dedication sites). Explicit representation of software configuration both from control and data flow perspective seems to open up a lot of potential for optimization. However, more tool support needs to be developed to fully benefit from this. Currently each configuration has to be fully tested before it can be used and this limits the usability of this property.

All of these prototypes have model based motor control core (based on the techniques from WP5). VSD prototype does not yet have torque control (Requirement **05A-4_1_1**) and it does not have speed feedback (requirement **05A-4_1_2).** Primitive protection functions have been implemented but the use of feedback as indicated by requirement **05A-4_1_3** is very limited. Brno University of Technology was able to demonstrate that model predictive control can be successfully used to control frequency converters. We also studied parallel motor control. However, in dual core processors it is more advantageous to use the processor cores to perform independent tasks. This way both the strict timing requirements of synchronous communication and the timing of motor control can be independently fulfilled.

Hypervisor verification for proof of concept core isolation has been completed in WP6 providing insights for real-time operating system solution for the HW platform. Developed models have been analyzed within framework developed in Task 5.3 where transversal services enabled by tool adaptors will allow model results to be assessed and traced according to prototype requirements (**09B-4_6_0, 09B-4_6_1, 09B-4_6_2**). Safety evaluation (WP6) will be primarily based on adopting OSLC solutions for seamless and semantically proven communication between different model-based design tools which will strengthen the certification credit towards the requirement **09B-4_6_3.**

3. T10.2 – UC_Identification and Authentication

NXP-GE is responsible for development of a flexible multi-core root of trust solution for embedded multi-processor/core, mixed criticality applications in dynamic and changeable real-time environments. The task of such a module is to act as a trust anchor that identifies & authenticates every communicating node in order to establish a chain of trust in any given network. For this purpose, such root of trust module needs a crypto coprocessor (also called a secure element) that can handle all cryptographic operations/protocols related to identification and authentication. A secure element is a tamper resistant, certified (like Common Criteria, EMVCo standardization institutes) microcontroller that provides secure/isolated processing environment and secure storage features which finds application in banking cards, passports, transit, insurance cards etc. A single secure element can perform one task and only one type of crypto operation at a time as per the configuration. However, a multi-secure-element-core root of trust (MSECROT) module can process multiple tasks in parallel and support multiple type of crypto protocols simultaneously. Thus, in a network, a MSECROT has faster response time for diverse types security requests and better throughput for higher load scenarios. This is the motivation for developing a MSECROT module.

A typical network with MSECROT can be envisaged as shown in Figure 14.

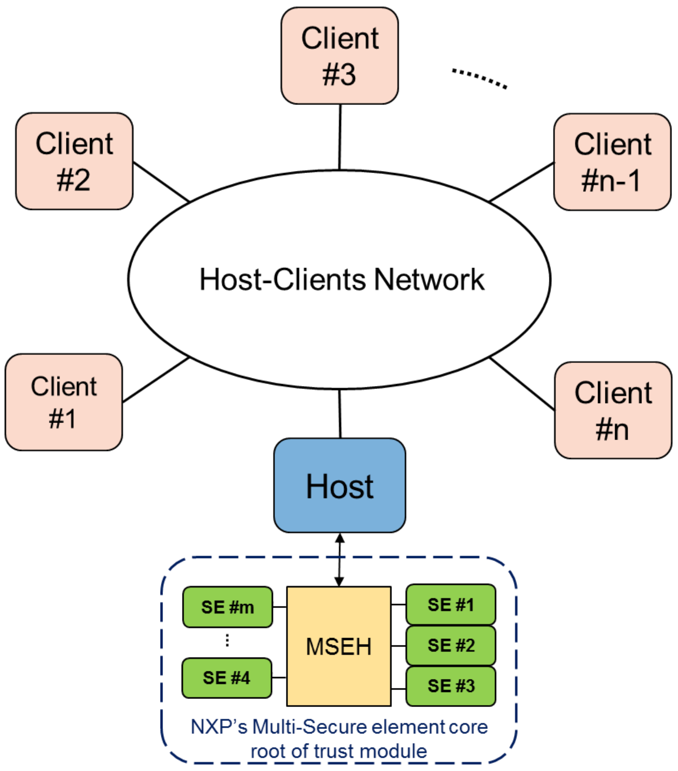


Figure 14 - Host-clients network with MSECROT

3.1 Functionality

It provides security as service to the connected network following the principles of service oriented architecture(SoA). It implements a scale down version of public key infrastructure performing unilateral / mutual authentication of client nodes to a host node or vice versa. Typically, in any network, clients authenticate to a host using some challenge-response schemes (like TLS handshake mechanism) with asymmetric key crypto protocols (like RSA/ECC) which is the same principle employed in MSECROT. It can also negotiate session keys for encrypted message transactions between host and clients (using symmetric key cryptoprotocols like AES). The list of all currently supported functionalities (also called as security specifications) of MSECROT is given in Table 1.

Crypto protocols	Security APIs	Description
True Random Number Generation	RND_GetRandomNum()	Needed for challenge-response scheme. A random number acts as a challenge sent by a verifier which is signed by private key of prover. The resulting signature is verified by verifier using prover's public key
RSA asymmetric key protocol (Supports 1024/2048 key sizes)	RSA_Sign()	Used for signing a challenge
	RSA_Verify()	For verifying a signature against challenge
	RSA_Encrypt()	Message encryption
	RSA_Decrypt()	Message decryption
ECC (elliptical curve cryptography) asymmetric key protocol (Supported Curves are NIST192, NIST224, NIST256, BrainPoolP192r1, BrainPoolP224r1, BrainPoolP256r1)	ECC_Sign()	Used for signing a challenge
	ECC_Verify()	For verifying a signature against challenge
Elliptical curve Diffie Hellmann	ECDH_GenerateEphemeralKeyPair()	Prior to session key creation between client and host, an ephemeral public/private key pair is generated by both and respective public key is exchanged insecurely on the network
	ECDH_GenerateSharedSecret()	After public key exchange, a common shared secret for symmetric key protocol is generated
AES (Supported modes are ECB, GCM, CBC, GMAC with key sizes 128,192 and 256. Key wrapping supported)	AES_Encrypt()	Message encryption
	AES_Decrypt()	Message decryption
General purpose secure storage	SetParam()	Save data (certificates, master keys)
	GetParam()	Retrieve data
	EraseParam()	Delete data

Table 1 - Security Specifications or APIs of MSECROT

3.2 Architecture

The multi-core root of trust module is composed of central ARM controller (NXP ARM Cortex M3 LPC1769) interfaced with multiple secure elements (NXP A7001 or A70CM) on board using I2C as shown in Figure 15. The ARM controller is called multi-core secure element handler (MSEH) as it is responsible for initialization & configuration of secure elements, scheduling, arbitration & load distribution of security tasks assigned by host and provide multiple communication interfaces to host (like USB, Ethernet, SPI, or UART).

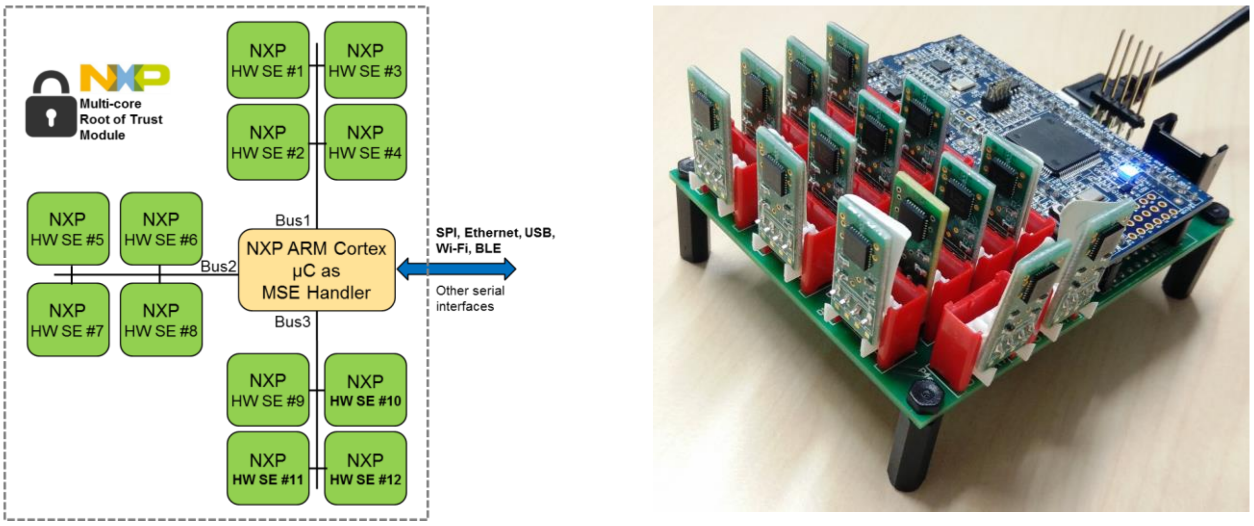


Figure 15 - NXP Multi-secure-element core root of trust module

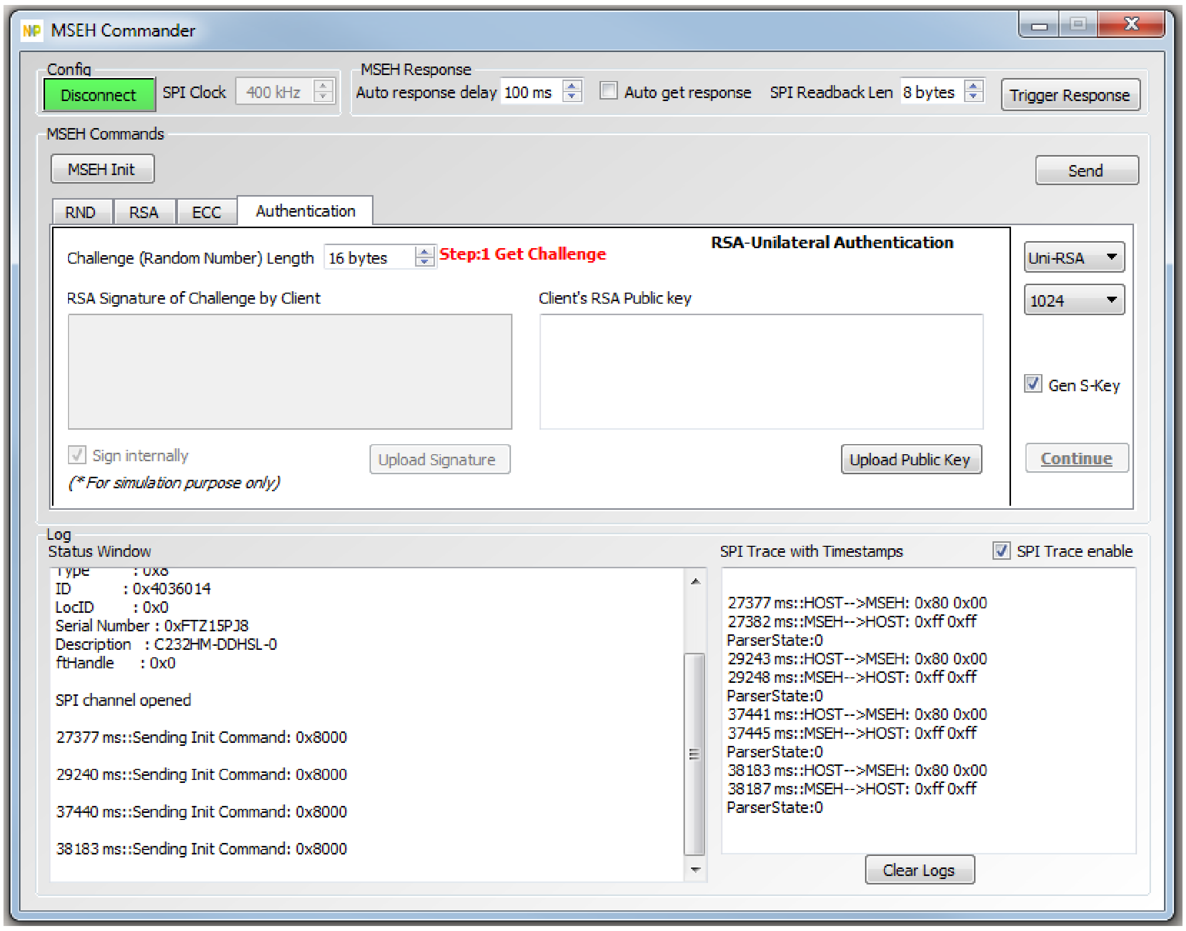
3.3 Assessment of MSECROT with respect to EMC2 requirements

The requirements for such a module as defined by EMC2 consortium is as described in Table 2.

Requirement ID	Category	Short description
01R_NXP_001	Networked security	To adapt the security system to the UC identification and authentication in a multicore environment it needs to be connected to a non-linear decentralized bus system; the interface will be defined with a layered approach to make the hardware layer interchangeable
01R_NXP_002	Security counter parts	each multi-core node needs to be equipped at least with a secure element as trust anchor
01R_NXP_003	needed security levels	the needed level of security of the whole system as well as necessary differentiations in terms of required security need to be defined
01R_NXP_004	Parallel Processing	Scalability of the security system must not rely on an increase of the performance of single processing elements but on adding additional elements
01R_NXP_005	Runtime certification of cyber-physical systems	provide methods for the runtime evaluation of trust certificates and initiation of an asymmetrically encrypted communication;
01R_NXP_006	Identification	Each part of the system, e.g. a service or communication channel, should be identified by a unique name; for low cost security needs symmetric encryption via device pairing shall be considered as an optional aspect;

Table 2 - Requirements of a multi-core root of trust

The final demonstrator fulfills all the above-mentioned requirements as illustrated in the below sections. All the supported functionalities are tested using a PC based graphic user interface application developed using QT framework as shown in Figure 16.



SPI Clock 400 kHz
Auto response delay 100 ms
Auto get response: unchecked
SPI Readback Len 8 bytes
Challenge (Random Number) Length 16 bytes
Step:1 Get Challenge
Uni-RSA
1024
Gen S-Key: ✓
Sign internally: ✓
(*For simulation purpose only)
SPI Trace enable: ✓

```
Type         : 0x8
ID           : 0x4036014
LocID        : 0x0
Serial Number : 0xFTZ15PJ8
Description  : C232HM-DDHSL-0
ftHandle     : 0x0

SPI channel opened

27377 ms::Sending Init Command: 0x8000

29240 ms::Sending Init Command: 0x8000

37440 ms::Sending Init Command: 0x8000

38183 ms::Sending Init Command: 0x8000
```

```
27377 ms::HOST-->MSEH: 0x80 0x00
27382 ms::MSEH-->HOST: 0xff 0xff
ParserState:0
29243 ms::HOST-->MSEH: 0x80 0x00
29248 ms::MSEH-->HOST: 0xff 0xff
ParserState:0
37441 ms::HOST-->MSEH: 0x80 0x00
37445 ms::MSEH-->HOST: 0xff 0xff
ParserState:0
38183 ms::HOST-->MSEH: 0x80 0x00
38187 ms::MSEH-->HOST: 0xff 0xff
ParserState:0
```

Figure 16 - PC GUI application to test security specifications of MSECROT

3.3.1 Key performance indicators of MSECROT

- The current version of MSECROT module can accommodate up to 14 secure elements interfaced to MSEH on its 3 I2C busses. Since SE acts as slave to MSEH, it responds only when triggered by MSEH. Thus, multiple SEs can be placed on the same I2C (however different I2C addresses) without compromising on performance. Incoming security requests are distributed by MSEH to all SE as per their configuration. MSEH polls each SE for response in a round-robin format.
- The performance comparison of single secure element against MSECROT is shown in Table 3. The response time for each crypto operation measured during internal testing, gives a glimpse of performance of MSECROT

Type of Crypto operation	**Response time of Single Secure element (in ms)**	**Response time of MSECROT with 4 SE (in ms)**
Random Number of 32-byte length	23	11
RSA 1024 signature	157	55.5
Unilateral Authentication with RSA1024 [Get_Rnd() + Hash() with SHA1 + RSASign() + RSAVerify()]	631	366

Table 3 - Response times comparison between single SE and MSECROT

An MSECROT with 14 secure elements cores doesn't imply that it is either 14 times faster or can handle 14 times the load of single secure element. There are additional delays due to parsing of incoming request & forwarding to appropriate secure element with right configuration, scheduling, bus transmission delay (transmission times from host to MSEH and MSEH to SE and vice versa, transmission speed: I2C functions as 100kbps, SPI at 400kbps although MSEH runs at 24MHz), processing time etc. Therefore, there is performance is also affected proportionately.

- It can support multiple security protocols at once. Some of the SEs can be configured for RSA1024 and some for RSA2048 and others for different ECC curves. Hence client nodes can send different types of request at once
- It has in-built redundancy feature. If one of the SE malfunctions, then the other SE with same configuration can take over its load. This way it can ensure that quality of service is not compromised
- It can support dynamic configuration/reconfigurations. It is possible to have an implementation on MSEH that constantly evaluates the incoming security loads and based on the frequency of request for each type of crypto-operation, it reconfigures underlying SEs to support most frequent crypto-operation. That way, it ensures that higher throughput is retained during operations.
- It offers interoperability to external networks. The security APIs are abstract enough to encapsulate SE specific commands in its internal SW layers. So, in future, if SE from different manufacturer is used, the security APIs remains the same. Only the internal interface layer will be adapted to SE specific handling.

3.3.2 MSECROT relationship to EMC2 objectives, KPIs & other WPs

It fulfills all strategic targets of ARTEMIS (i.e.)

- *Reduce cost of system design & development cycle times*

  It uses only commercially available off the shelf (COTS) components which reduced the design cost and time to market drastically. MSEH is COTS development board for LPC1769 and SE are COTS A7001 modules. They have been simply plugged on to base board which reduces the integration costs and number of design iterations

- *Reduce time & effort for re-validation & re-verifications*

  Secure elements are EMVCo, Common criteria and other standardization committees certified. Hence is no need to re-certify the complete solution again

- *Manage the increase in system complexity*

  It is a multi - processor system with multiple OS (Java card operating system on SE and FreeRTOS on MSEH), multiple layers of SWs (for both MSEH and SE) which closely coordinate to accomplish the necessary functionalities. The SW is written in modular fashion with test-driven development approach. This ensures plug & play functionality along with less time & effort in V&V stage. Thus, complexity is managed in MSECROT

- *Achieve cross sectorial reusability*

  The security APIs are very generic and can find application in many fields where security is important. The next generation of MSECROT is planned to support wireless protocol such as BLE, Wi-Fi, ZigBee and NFC. It is planned to this module as IoT Gateway in future projects. It can be used in industrial automation sector too. Industry 4.0 is new trend where cyber-physical-systems are turning autonomous and moving towards everything connected and everything secure. MSECROT module can find application here too.

It fulfills **EMC2 objective OBJ63** (i.e. Security as a service) that states that "Hybrid solutions as combinations multiple secure elements and managing processors to provide flexible roots of trust". Although the KPIs associated with OBJ63 (i.e. KPI48- KPI53) aren't directly relevant to MSECROT, it still accomplishes some of it in an indirect fashion. It has better latency (KPI49), efficiency (KPI52) and bandwidth utilization (KPI53) in comparison to single secure core solution as shown in Table 3. The KPIs of OBJ63 are however more relevant to industrial manufacturing and logistics which is the main use case for the living lab work packages.

Technology	**Respective Work package name**	**Description of linkage in MSECROT**
System and service level security (T1.5)	WP1	It can provide different kinds of authentication and identification services to its network based on agreed service levels & desired Quality of service Unilateral /bilateral authentication services are possible Multiple configurable cryptographic routines for handshake (like ECC, RSA) and encryption/decryption (AES in ECB, CBC, GMAC etc.) services possible
Communication services (T3.2) Networking (T4.3)	WP3 WP4	It supports multiple communication protocols for interfacing to the network. It could be Ethernet, USB, SPI, I2C, UART etc. This gives the system architect the flexibility to select any one of the above the communication mode for interfacing to the rest of the system
Virtualization and isolation (T3.3)	WP3	It is composed of multiple secure elements working either in parallel or tandem based on configuration to provide security as a service to the network. It can act as single virtual trust anchor of the system that distributes its security load amongst the underlying cores based on priority or service levels or configurations. Each core can provide an isolated execution environment for a request and thus service as many requests in parallel as the number of cores
Security mechanisms and services (T3.4)	WP3	the individual secure elements have inbuilt security mechanisms to counter physical, side channel and other types of attacks. They have tamper resistant memory units to safeguard the critical data of the system and provide a secure execution environment. Through cryptographic handshake mechanisms they can protect the communication channel between them and client requesting their services.
Heterogeneous Multiprocessor SoC architectures (T4.1)	WP4	Each core can provide a different set of security services compared to its neighbor. Thus, forming a heterogeneous multi-processing architecture. As mentioned before each core can be configured to run different types of cryptographic routines and can also be reconfigured at run time dynamically based on the load and desired quality of service
Verification and Validation Techniques (T4.4)	WP4	Individual secure element cores are common criteria certified (EAL6+), approved by EMVCo, BSI and other international certification authorities. The multi-core root of trust solution is composed of these certified secure elements as individual cores and the central handler simply distributes the incoming security traffic to these cores and play no role in security activities such as handling public/private keys, performing crypto operations etc. Hence the overall system can be considered as certified interconnect of security cores. The validation & verification process of individual secure cores can be extended to multicore root of trust solutions too

Table 4 - MSECROT features linkage to other WPs of EMC2

MSECROT employs concepts and technologies developed in other work packages of EMC2. Although there is no direct technology transfer from other WPs to MSECROT, it based on similar principles. A table

(See Table 4) describing the cross linkage of MSECROT features to the other WP technologies illustrates the connection in detail.

3.3.3 Final Demo of MSECROT

A final demo is planned together with WP10.3 which deals with heterogeneous tracking module (Multiple location / position sensors are connected to a central Linux gateway module) from Ambar Telecomunicaciones. The use case to be demonstrated in the final demo is unilateral authentication of tracking sensor nodes to central Linux gateway. The gateway or the host forwards all security requests from tracking nodes to MSECROT and sends its response back to sensor nodes. The host is connected to MSECROT via SPI. A block diagram of the complete system is shown in Figure 17.

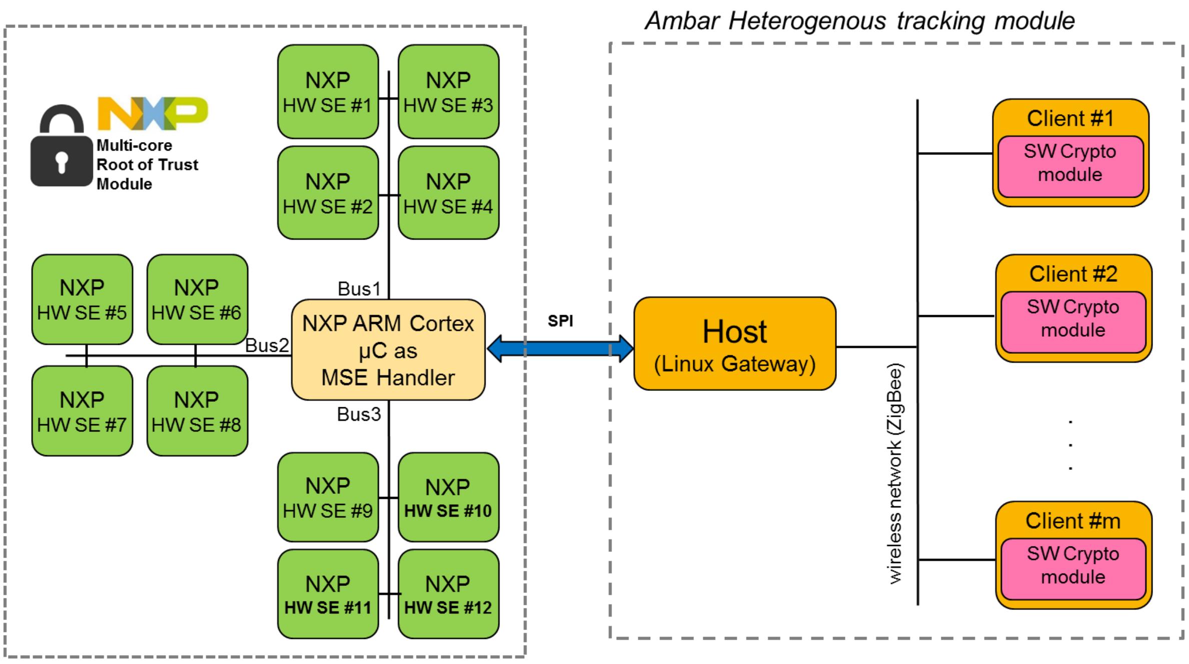

Ambar Heterogenous tracking module
Multi-core Root of Trust Module
NXP HW SE #1
NXP HW SE #2
NXP HW SE #3
NXP HW SE #4
NXP HW SE #5
NXP HW SE #6
NXP HW SE #7
NXP HW SE #8
NXP HW SE #9
NXP HW SE #10
NXP HW SE #11
NXP HW SE #12
Bus1
Bus2
Bus3
NXP ARM Cortex µC as MSE Handler
SPI
Host (Linux Gateway)
wireless network (ZigBee)
Client #1 SW Crypto module
Client #2 SW Crypto module
Client #m SW Crypto module

Figure 17 - Block diagram of MSECROT with AMBAR Tracking module

The unilateral authentication of sensor nodes with host is done using RSA 1024 crypto protocol. A representation of authentication process is shown in Figure 18.

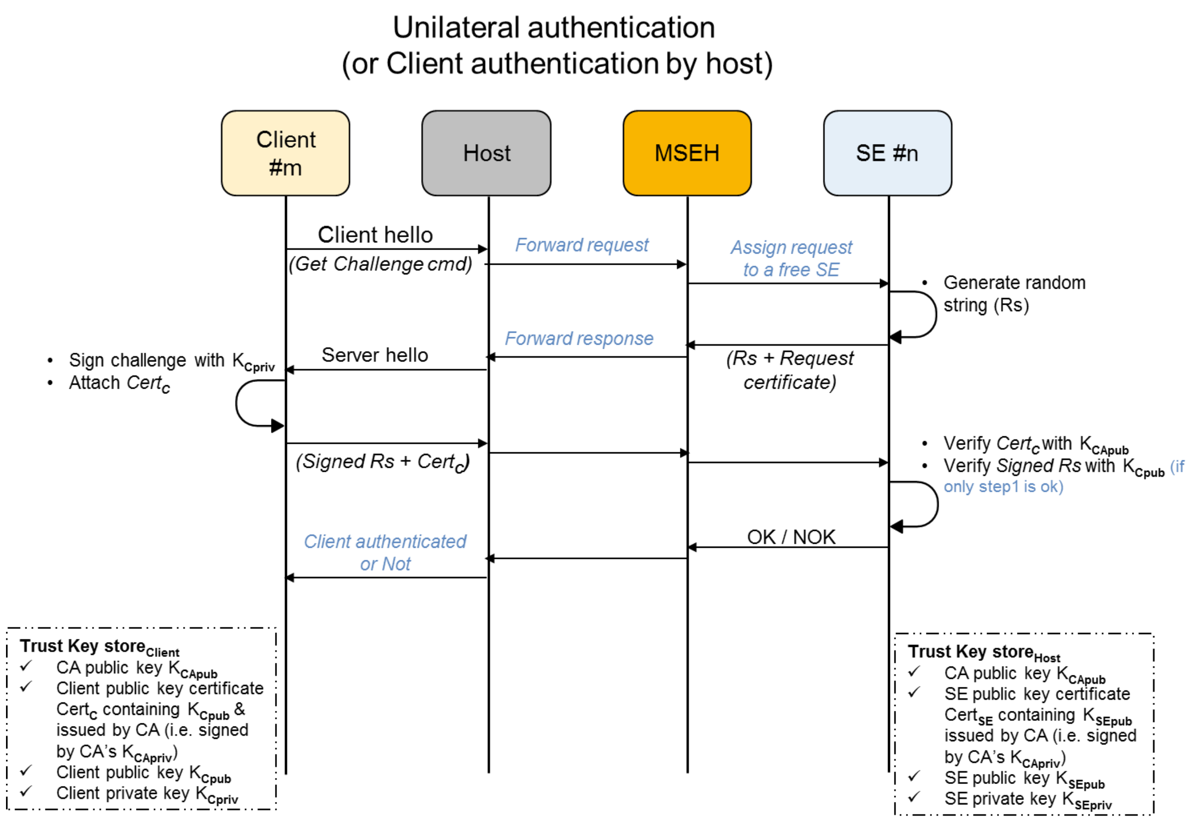

Unilateral authentication
(or Client authentication by host)
Client #m
Host
MSEH
SE #n
Client hello
(Get Challenge cmd)
Forward request
Assign request to a free SE
Generate random string (Rs)
Forward response
(Rs + Request certificate)
Server hello
Sign challenge with K_{Cpriv}
Attach $Cert_C$
(Signed Rs + $Cert_C$)
Verify $Cert_C$ with K_{CApub}
Verify Signed Rs with K_{Cpub} (if only step1 is ok)
OK / NOK
Client authenticated or Not
Trust Key store$_{Client}$
✓ CA public key K_{CApub}
✓ Client public key certificate $Cert_C$ containing K_{Cpub} & issued by CA (i.e. signed by CA's K_{CApriv})
✓ Client public key K_{Cpub}
✓ Client private key K_{Cpriv}
Trust Key store$_{Host}$
✓ CA public key K_{CApub}
✓ SE public key certificate $Cert_{SE}$ containing K_{SEpub} issued by CA (i.e. signed by CA's K_{CApriv})
✓ SE public key K_{SEpub}
✓ SE private key K_{SEpriv}

Figure 18 - Unilateral authentication process

Since sensor nodes are remotely located and battery powered (most of the times), it often lacks computation power and resources to do complex PKI authentication processes. Hence a simplified authentication protocol without a central certificate authority is implemented in the demo.

4. T10.3 – UC_Tracking

The purpose of this use case is providing a platform which integrates different tracking and communication technologies. Finally, this platform consists of the following elements:

- Central server / gateway.
- WIFI nodes.
- Tracking Device (mobile phone as example).
- ZigBee identification modules.

4.1 Gateway improvements

As result of the test performed with the first gateway version a new board has been designed to fix the detected issues and improve the features and performance. The key improvements are:

1. More powerful microprocessor.
2. Reduced size.
3. Wi-Fi/BLE embedded
4. Added GSM/GPRS modem.

The following table summarizes the differences between the first version of the gateway and the current version:

	GATEWAY Vybrid Based (1º versión)	**GATEWAY SoloX Based (2º versión: Mayo 2016)**
Dimensions	115x85mm (without enclosure)	**90x60mm** (without enclosure)
Microprocessor	Multicore: Cortex-A5 (533 MHz) + Cortex-M4 (167 MHz)	Multicore: **Cortex-A9 (1GHz)** + Cortex-M4 (200 MHz)
OS	Linux (Debian)	Linux (Ubuntu)
Interfaces	Micro USB (console)	Micro USB (console)
	Ethernet	Ethernet
	Wi-Fi	**Wi-Fi (integrated)**
	Bluetooth 4.0	**Bluetooth 4.0 (integrated)**
	ZigBee	ZigBee
	6LoWPAN	6LoWPAN
		GSM/GPRS
	USB host	USB host
	SPI	SPI
	I2C	I2C
	GPIO (x2)	GPIO (x2)
Power interface	5V/2A	**12V/2A**

Table 5 – Gateway improvements

As in the first version this gateway runs a Linux distribution that gets the flexibility for performing communication tasks and for executing other activities. Figure 19 shows the new version of the prototype mounted over an UDOO board.

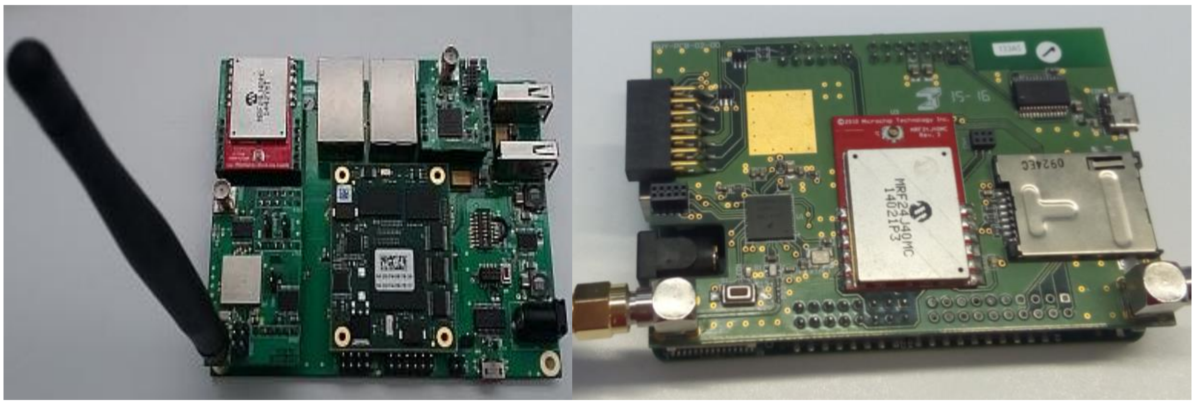


Figure 19 - Gateway v1 (left), Gateway v2 (right)

4.2 Architecture and description of the prototype

The Wi-Fi ESP8266 nodes have been added to the setup to implement the tracking demo test as anchor nodes with known positions. The geographical position is configured as the Wi-Fi SSID embedded in a hexadecimal code in every anchor node. A smartphone is used as tracking device with a customized Android app that retrieves all the Wi-Fi nodes in range and shows each ESP8266 node information in the screen. At the same time all the information is forwarded to the gateway using UDP datagrams.

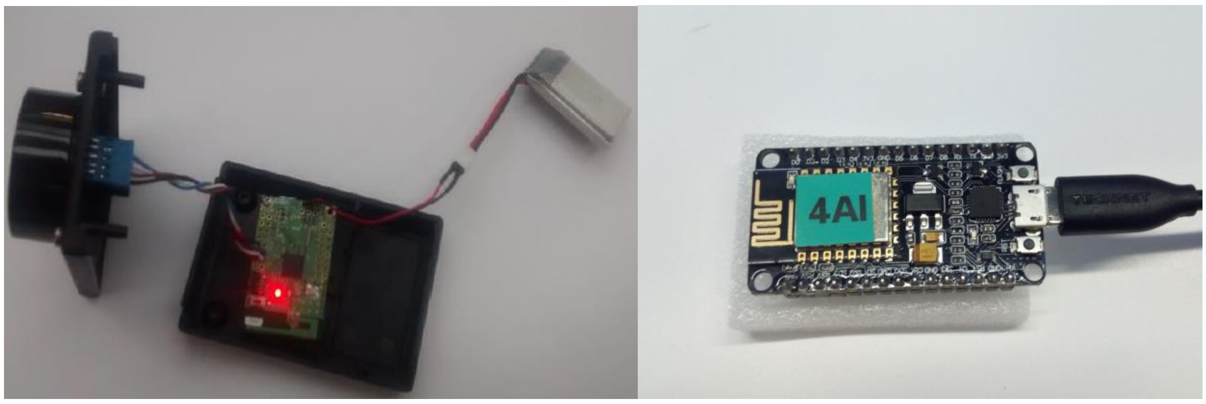


Figure 20 – ZigBee module (left), Wi-Fi Node ESP8266 (right)

The android app uses three main libraries to get an estimated GPS position:

- Subpos (www.subpos.org).
- Trilateration-master.
- Commons-math3.

These libraries are used to perform position triangulation based on the received signal strength from each anchor Wi-Fi ESP8266.

Regarding the Gateway/Server software, a custom application has been designed and implemented using the JavaScript language both in server and client side following a Model/View/Controller (MVC) design pattern. The server hosts a full JavaScript stack based on the MEAN.JS[1] framework: MongoDB, Express, AngularJS and Node.js. MongoDB provides a non-SQL database implementation, Express multiple features to support the web application implementation, AngularJS a MVC client-side MVC framework and Node.js represents an optimum and lightweight JavaScript server-side platform.

[1] https://meanjs.org/

The main features of the Gateway/Server application are:

- serve the requested data that are stored in the database,
- provide html responses including the AngularJS directives embedded to the client,
- receive UDP datagrams, that are sent from the tracking devices with location information, and stored this data in the local MongoDB database.

An overview of the full system architecture is shown in Figure 21.

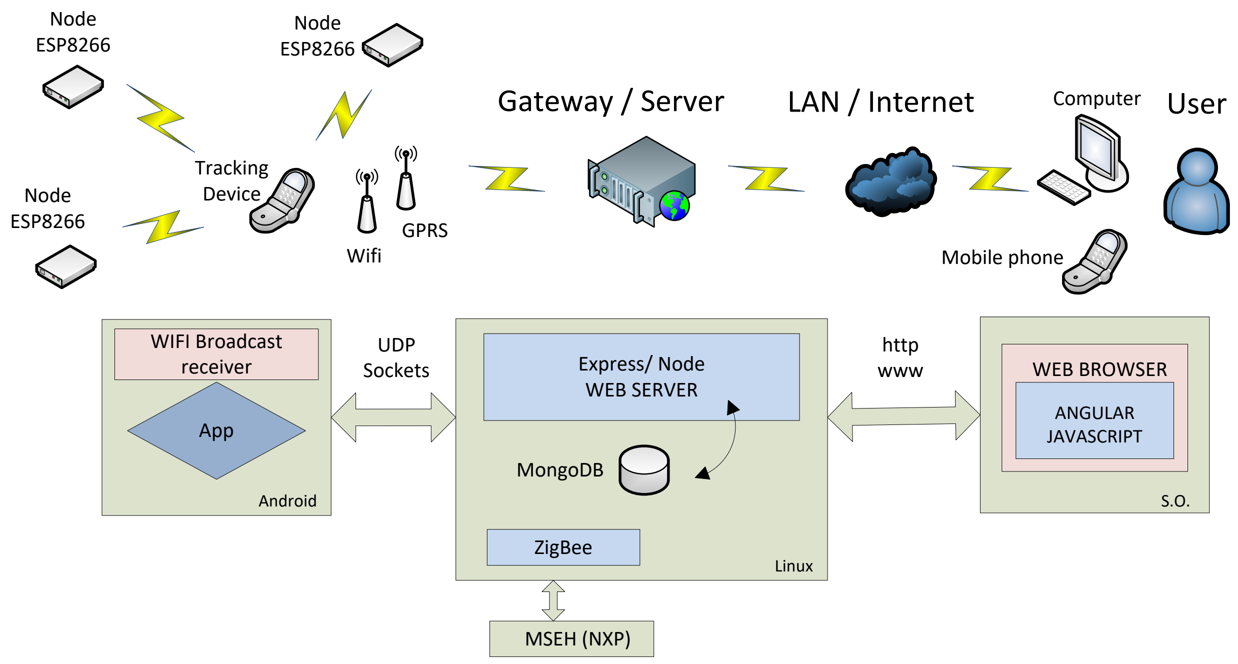
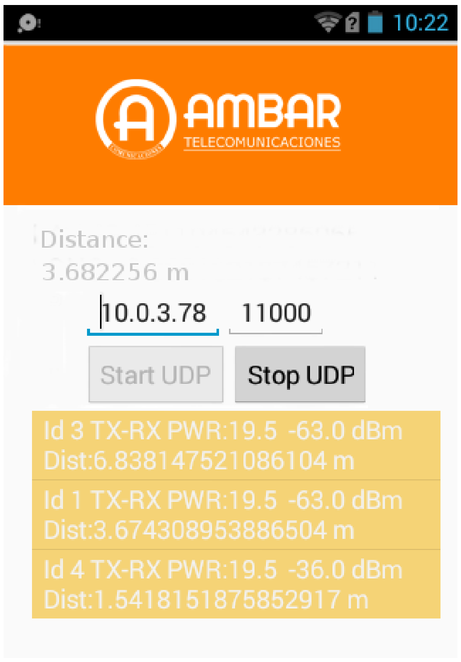


Figure 21 – Tracking full system architecture

Three main layers were detailed in D10.4 that can be identified in the architecture overview Figure 21:

- The smartphone application that provides the GIS calculation is embedded in an Android app.
- The client side or web GUI that depicts all data and GIS positions of the tracking devices. It is written using AngularJS and asynchronous calls. This reduces the server traffic and improves the system responsiveness transferring only the minimal amount of data at a time instead of refreshing whole page.
- The service layer, in charge of all communications, serve the information to the client web GUI, retrieving UDP location packets from the smartphone app and storing the data to the MongoDB database. When server receives UDP datagrams from a tracking device, that contains distances and positions, the information is stored in the database that later can be forwarded to the web clients.

Figure 22 depicts the Android app that can be installed in the tracking devices. In this example the signal of three nodes is received. The received power and the estimated distance to the node are showed in the screen for each node. After the signal strength triangulation process the estimated distance to the reference point is showed in the screen.



Distance:
3.682256 m

10.0.3.78 11000

Id 3 TX-RX PWR:19.5 -63.0 dBm
Dist:6.838147521086104 m

Id 1 TX-RX PWR:19.5 -63.0 dBm
Dist:3.674308953886504 m

Id 4 TX-RX PWR:19.5 -36.0 dBm
Dist:1.5418151875852917 m

Figure 22- Android app screenshot in tracking device

The web GUI goal is to show the requested information in tables and graphs. Currently the application offers a graph where is represented the building, the tracking device, nodes, and distances between them. An example is depicted in Figure 23 below.

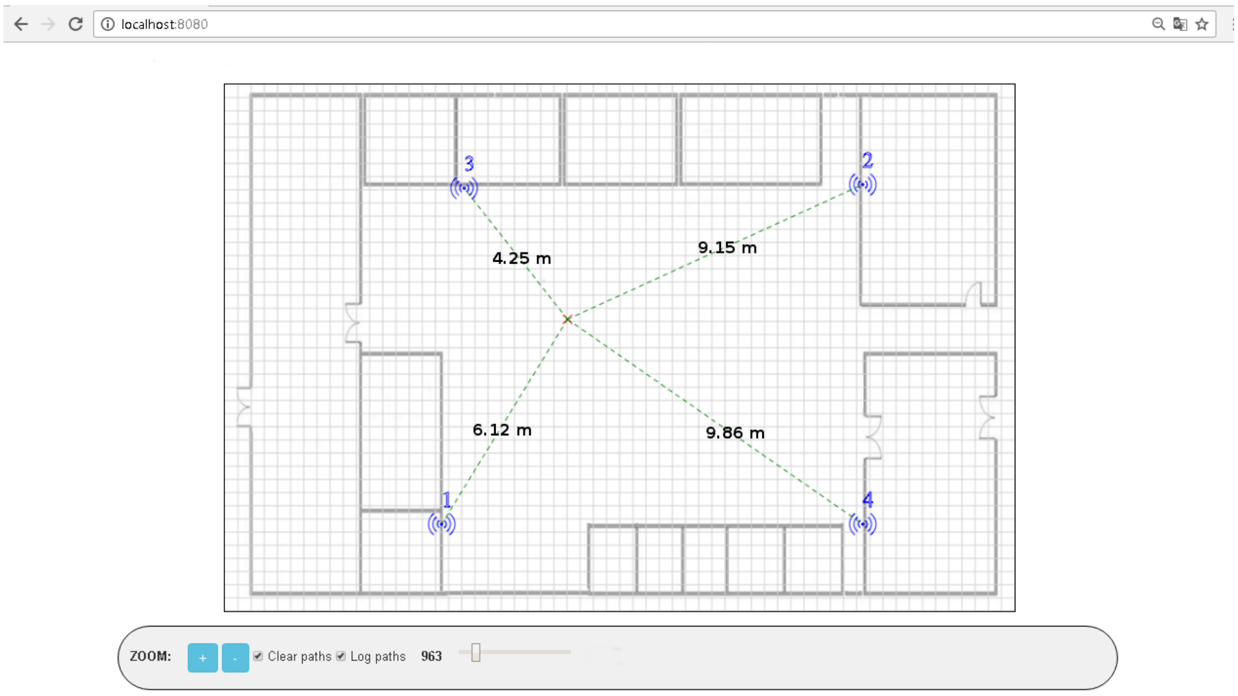
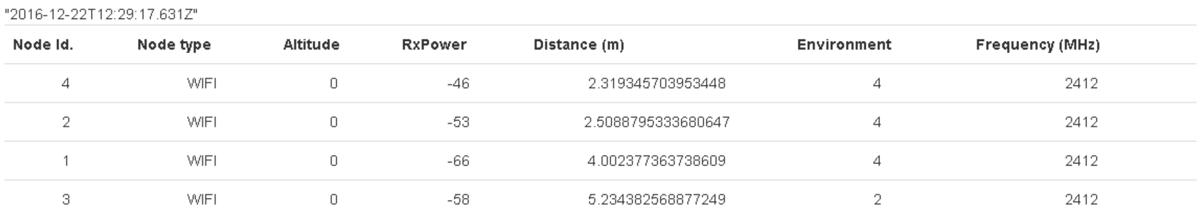


Figure 23 – GUI: distances graph on client's browser

Other information related to the received power, type of RF communication, RF environment model parameter, estimated distances to each node and other parameters are represented for each node. Table 6 shows an example of information table that it was used in this demo.

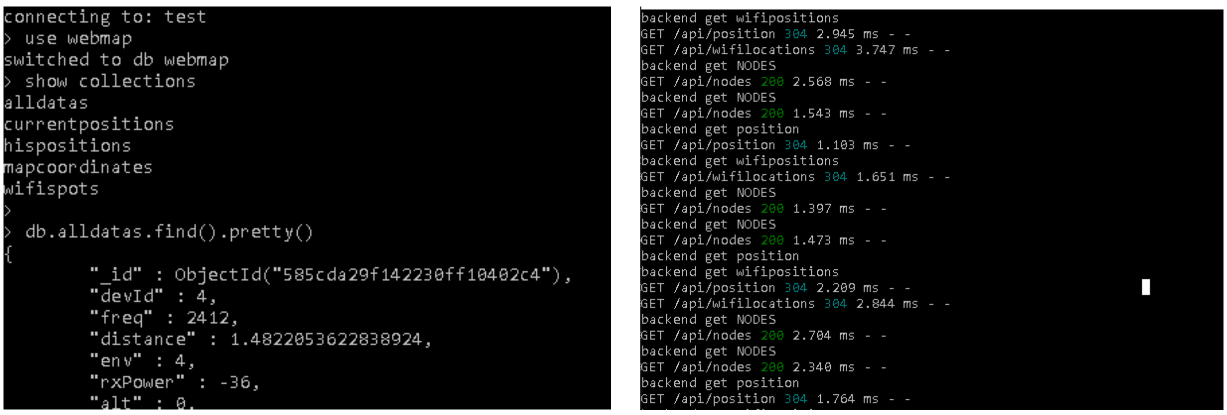


"2016-12-22T12:29:17.631Z"

Node Id.	Node type	Altitude	RxPower	Distance (m)	Environment	Frequency (MHz)
4	WIFI	0	-46	2.319345703953448	4	2412
2	WIFI	0	-53	2.5088795333680647	4	2412
1	WIFI	0	-66	4.002377363738609	4	2412
3	WIFI	0	-58	5.234382568877249	2	2412

Table 6 – GUI: nodes information table on the client side.

The service layer includes a custom Node.JS application that uses Express to provide the web GUI features and UPD connections to retrieve location data from the Wi-Fi nodes. This data is stored in the "collections" that have been defined in the MongoDB or "tables". These tables stores all data that is needed for the tracking application related to identifiers, frequencies, estimated reception power…. The current and previous positions are stored, therefore an historical set of movements is provided that enables the system to implement further location-aware services and tracking application. An example of the connection to the MongoDB database showing part of the nodes "collection" and the web requests that the Node.JS/Express server receivers are shown in Figure 24.



```
connecting to: test
> use webmap
switched to db webmap
> show collections
alldatas
currentpositions
hispositions
mapcoordinates
wifispots
>
> db.alldatas.find().pretty()
{
        "_id" : ObjectId("585cda29f142230ff10402c4"),
        "devId" : 4,
        "freq" : 2412,
        "distance" : 1.4822053622838924,
        "env" : 4,
        "rxPower" : -36,
        "alt" : 0,
```

```
backend get wifipositions
GET /api/position 304 2.945 ms - -
GET /api/wifilocations 304 3.747 ms - -
backend get NODES
GET /api/nodes 200 2.568 ms - -
backend get NODES
GET /api/nodes 200 1.543 ms - -
backend get position
GET /api/position 304 1.103 ms - -
backend get wifipositions
GET /api/wifilocations 304 1.651 ms - -
backend get NODES
GET /api/nodes 200 1.397 ms - -
backend get NODES
GET /api/nodes 200 1.473 ms - -
backend get position
backend get wifipositions
GET /api/position 304 2.209 ms - -
GET /api/wifilocations 304 2.844 ms - -
backend get NODES
GET /api/nodes 200 2.704 ms - -
backend get NODES
GET /api/nodes 200 2.340 ms - -
backend get position
GET /api/position 304 1.764 ms - -
```

Figure 24 - Mongo database collections (left), Module-controller communications log (right).

The main challenge of this tracking system is the estimation of the position from the RF signal propagation characteristics and power reception (Rx) in the tracking devices. The setup had to deal with known issues related to the variance of the Rx power, if the receiver is too close to a node the received power is too high and it changes drastically a few cm farther from the node, while the received power for a distant node is almost constant. The propagation formula employed and the environment corrector coefficient are the most sensitive point in the design and play a fundamental role in the indoor position calculation. Currently the corrector coefficient is coded in the SSID of the node and because of current firmware of the node it can't be changed remotely.

Other important challenge to setup and implement the full system is the setup of the MongoDB in ARM 32-bit architecture. Last versions of MongoDB do not offer support for 32-bit architectures. An older version of MongoDB (v2.1.1) had to be full compiled and manually installed in the system. Since this version was released, MongoDB project has faced a lot of changes and improvements, even in the instructions and procedures that are needed to store a retrieve data from database. As a future improvement for the gateway/server the upgrade to ARM 64-bit architecture will be evaluated to easily integrated MongoDB and other IoT libraries only provided for 64-bit architectures.

4.3 Relationship with other WPs

4.3.1 Contribution from technical WPs

This use case has contributions from the following technical WPs:

- WP1: establishes the base requirements and specifications of tracking devices.
- WP3: introduces the integration of different radio communication technologies into a single system. In case of indoor tracking of goods, typical radio beacons like Bluetooth LE or Wi-Fi and RFID technologies are likely to be used. Additionally, communication between the gateway and the central server requires an internet connection by means of GPRS/3G/4G or a wired network.
- WP5: provides support with tools and technologies to ensure the interoperability of different platforms and testing and validation.

4.3.2 Integration of solutions from other partners

ZigBee nodes and the NXP MSEH prototype described earlier have been integrated in the Tracking system to evaluated the used of NXP platform for the ZigBee node authentication. ZigBee nodes sends an authentication challenge to the gateway which connects through the SPI bus to the NXP MSEH prototype that provides the authenticate hardware engine (Figure 25).

The integration was divided in three steps:

- Phase 1: Software based implementation

  At a first step an adaptation of an embedded crypto library, that include support for standard PEM certificates, was integrated into the ZigBee nodes. File transfer mechanisms between the node and the Gateway were implemented with high-level functions, in both node and gateway, to provide client-side authentication using RSA-1024 bits based on the OpenSSL library.
- Phase 2: Hardware based implementation using an old NXP prototype

  In this second phase of the integration, the software OpenSSL features are replaced by the NXP MSEH platform connected to the SPI port of the gateway.
- Phase 3: Hardware based implementation using new NXP prototype

  In this third phase, the old NXP prototype is replaced by a newer version that supports several MSEH. The following picture shows AMBAR gateway connected to NXP MSEH prototype and two nodes with authentication support.

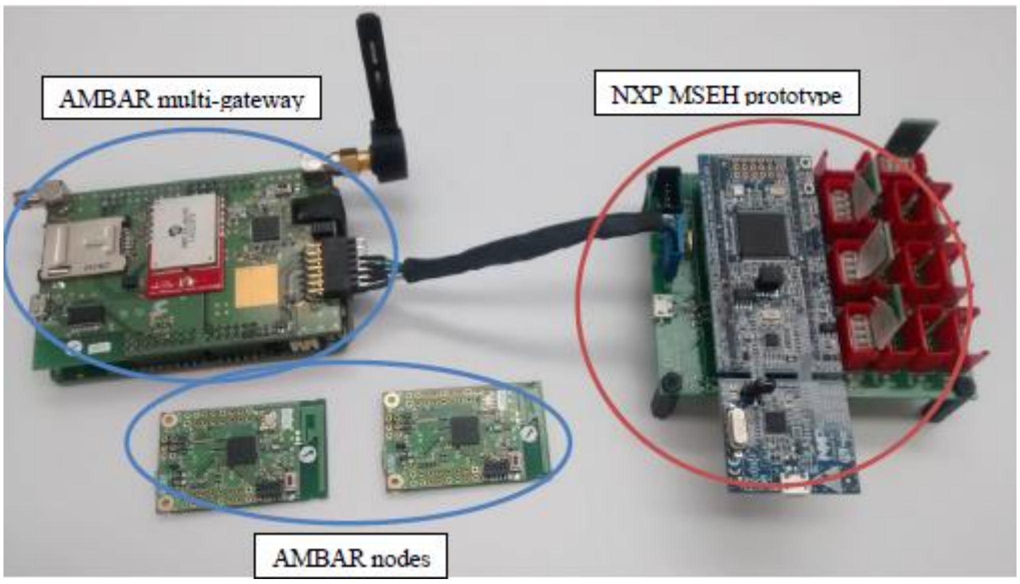


Figure 25 - NXP MSEH integration in the system

4.3.3 Evaluation of prototype against requirements

The following table summarizes how the requirements established in D10.1 are fulfilled by the implementation provided for the use case.

Requirement ID	Description	Prototype implementation
15R_AMBAR_014	The prototype must support heterogeneous communication technologies in a single platform.	This demo makes use of different communications gateway possibilities. Wi-F is used to get an estimated position of the Tracking device, at the same time GPRS can be used for tracking the gateway global position. Currently wired connections are used to client's browser. An SPI bus is used in the MSEH NXP solution and ZigBee nodes are identified by the system.
15R_AMBAR_015	The platform must support heterogeneous indoor/outdoor location	Indoor and outdoor location is supported as long as radio signal is received by tracking devices. Several tens of meters are recommended for the Wi-Fi location. GPRS gives the freedom to locate the server almost everywhere and the gateway can make use of a GIS server. ZigBee can be used in both environments an event Bluetooth low energy or NFC can be used to identify items.
15R_AMBAR_016	Location information must be centrally structured and prepared to be analysed in real-time with third-party algorithms. Results should be presented by using an appropriate user interface and prepared to be consumed by different devices (computer, cellular, etc.)	The utilization of web pages as GUI makes it possible for the end user to access the information from different kind of devices, such as computers, tablets or mobile phones, without requiring development of custom applications for every targeted device. All information is stored in the database of the gateway. Tracking devices and their locations are available in the tracking device and in the gateway/server. The information in sent to the gateway as soon as it's obtained by the tracking device and once is stored by the server can be analysed by third-party algorithms. The user interface is completely customizable to the end user necessities. It can be as simple as a graph for location representations or can be represented over a GIS map.

Table 7 - Requirements of the use case

5. T10.4 – UC Manufacturing Quality Control by 3D Inspection

This section provides a description of the completed final demonstration prototype for Task 10.4 “Manufacturing Quality Control by 3D Inspection”. A conceptual architecture and a block diagram of the final prototype design are presented and explained in detail. Then, a modelling example of the WP2 “art2kitekt” tool suite is presented performing a bus analysis of the final prototype. Moreover, an experiment is devised to be run and reported in the final deliverable aiming at measuring the performance of the prototype. Finally, a brief description of the main achieved requirements of this use case is described.

5.1 Description of the final prototype

The inspection system reconstructs the 3D shape of a captured object as well as its corresponding texture. The aim of this UC is to take advantage of multicore platforms and the tools provided within the EMC2 project (“art2kitekt”) to obtain a highly parallel and scalable version of the inspection system.

5.1.1 Conceptual architecture

The final prototype inspection system in the “Manufacturing Quality Control by 3D Inspection” use case follows a distributed architecture, organized into four different entities (see Figure 26):

- Capture provider
- Backend
- Broker
- Frontend

Each entity provides services that can be reached by requests or subscriptions. Communications among services are implemented with ZeroMQ sockets.

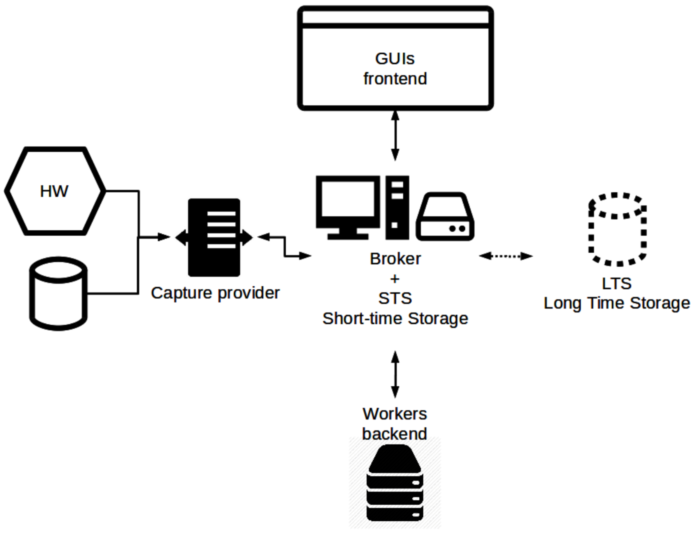


Figure 26 - Conceptual architecture

The **capture provider** transmits images from hardware to feed the backend. The implementation of this executable is known as ***zg3d-capturer*** and only one instance is required. The **broker** assumes the role of system manager. It is responsible for receiving and understanding each entity request, resending it to the corresponding service and finally responding. Its implementation requires two executable, known as ***zg3d-proxy*** and ***zg3d-master***. Only one instance is required for the master, but three proxies are needed. The

backend is the computing centre of the system. It is responsible for the hard computing processes: 3D reconstruction, model training, defect detection and object classification. All the operations that need computing power and high memory consumption are sent to the backend. Its implementation is known as ***zg3d-worker*** and N instances can be required, one for each of the computing nodes. Finally, the **frontend** is the entry point for any user of the system and implements a common language based on JSON requests. The human operator interactions with the system are done by a graphical user interface known as ***zg3d-frontend***.

5.1.2 Communication block diagram

The communication scheme among different entities is shown in Figure 27. Sockets provided by ZeroMQ can work in different modes:

- REQ/REP (Request/Reply): For each request message, a reply is expected.
- PUSH/PULL (Producer/Consumer): A message is distributed by a round-robin policy among the consumers.
- PUB/SUB (Publish/Subscribe): Each time an entity publishes a message; one or more subscribed entities receive it.

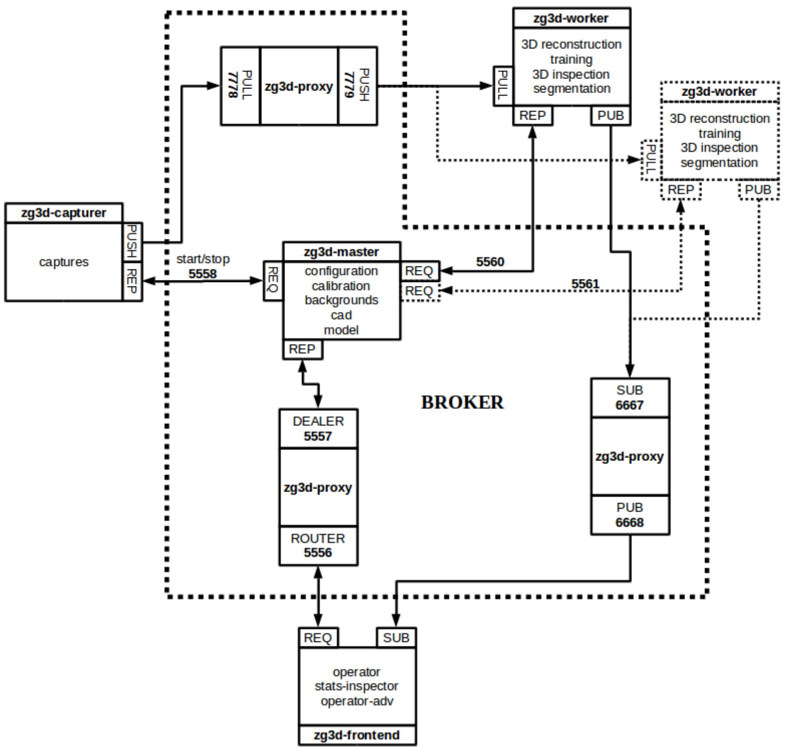


Figure 27 - Block diagram

Every request has a standard JSON format, including 5 basic fields:

- "sender": Request emitter.
- "recipient": Request receiver.
- "command": Request type.
- "encapsulated": Flag for encapsulated data request.
- "encoded": Flag for base64 codified data request.

Next, a list of the different **type of requests** that can be sent and received through the system is provided along with a JSON example:

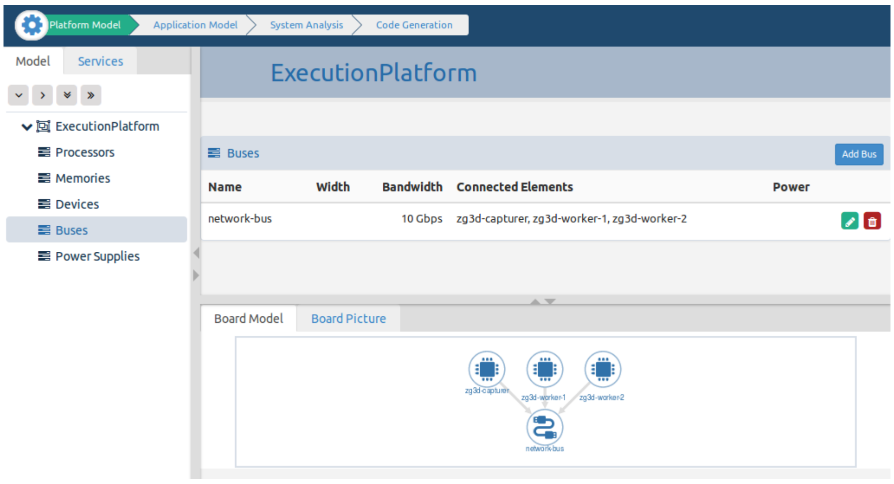
- **Get status -** { "sender": "operator", recipient: "master", "command": "get_status", "encapsulated": true, "encoded": false, "statustype": "start" }
- **Send status -** { "sender": "master", recipient: "operator", "command": "send_status", "encapsulated": true, "encoded": false, "status": "started" }
- **Init workers request -** { "sender": "operator", recipient: "master", "command": "send_init_workers", "encapsulated": true, "encoded": false, "taskname": "tooth"}
- **Start request -** { "sender": "operator", recipient: "master", "command": "send_start", "encapsulated": true, "encoded": false, "taskname": "tooth"}
- **Stop request -** { "sender": "operator", recipient: "master", "command": "send_stop", "encapsulated": true, "encoded": false}
- **Get n images request -** { "sender": "operator", recipient: "master", "command": "get_n_images", "encapsulated": true, "encoded": true, "imagetype": "background", "taskname": "tooth", "num_images": 16, "size": -1 } { "sender": "operator", recipient: "master", "command": "get_n_images", "encapsulated": true, "encoded": true, "imagetype": "target", "taskname": "tooth", "timestamp": "2016-11-21T12:32:03.103279Z", "num_images": 16, "size": 200 }
- **Send n images request -** { "sender": "master", recipient: “operator”, "command": "send_n_images", "encapsulated": true, "encoded": true, "imagetype": "target", "taskname": "tooth", "timestamp": "2016-11-21T12:32:03.103279Z", "images": ["iVBORw0KGgoAA [...]", "GuXydUlohIId95RD [...]", [...]] }
- **Published results -** { "sender": "W0", "recipient": "operator", "command": "send_string", "encapsulated": true, "encoded": false, "string": "{ \"id\": \"2016-11-21T12:32:03.103279Z\", \"object\": { \"num_points\": 5446, \"num_triangles\": 10874, \"bbox\": [-6.175856, 6.393371, -3.407393, 3.150465, -2.840201, 3.240691], \"area\": 209.801270, \"volume\": 225.106750 }, \"class\": \"Unknown\", \"vol_defect_error\": false, \"sur_defect_error\": true }" }

For the basic operation mode of the system previously described, the user just selects a specific task and starts the inspection process. This simple action automatically triggers the 3D capture sub-process and each time an object passes through the inspection system the 16 images are captured and delivered to a worker node to perform the 3D reconstruction and all the steps required to finally provide an inspection result that is published and shown by the graphical interface.

5.1.3 Model and analysis with “art2kitekt”

A tool suite for modelling and analysing real time systems, coming from WP2 and named “art2kitekt”, has been used to represent and analyse the final configuration of the prototype tasks responsible for the capture delivery and its corresponding 3D object reconstruction processes.

The bandwidth of the bus, 10Gbps, is defined at the ***“Platform Model” stage***, as it can be seen in Figure 28.



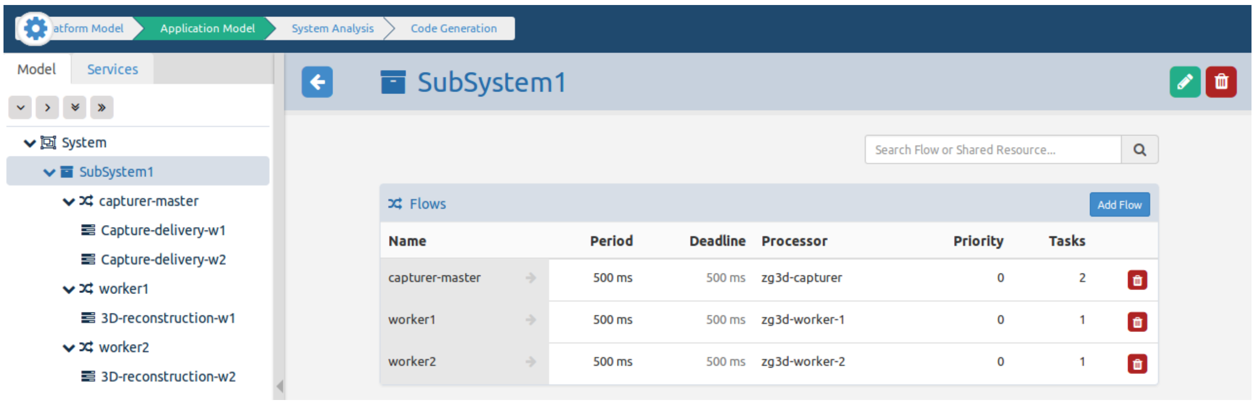
ExecutionPlatform

Buses

Name	Width	Bandwidth	Connected Elements	Power
network-bus		10 Gbps	zg3d-capturer, zg3d-worker-1, zg3d-worker-2	

Figure 28 – "art2kitekt" - Modeling of the execution platform

Then, the size of the image is defined in the ***"Application Model" stage***. Each capture is composed of 16 images of approximately 40Mb per image. Thus, the total size is 640Mbits per capture, equivalent to 80MBytes. A new task set has been modelled including a special flow for the data delivery (*"capturer-master"*), with two tasks (*"Capture-delivery-w1"* and *"Capture-delivery-w2"*) that include the "Sent Messages" information. These messages are used to model the image transmission through the network, and two additional flows (one for each "worker" node) are used to model the processing tasks carried out by each "zg3d-worker" instance to perform the 3D reconstruction computations.



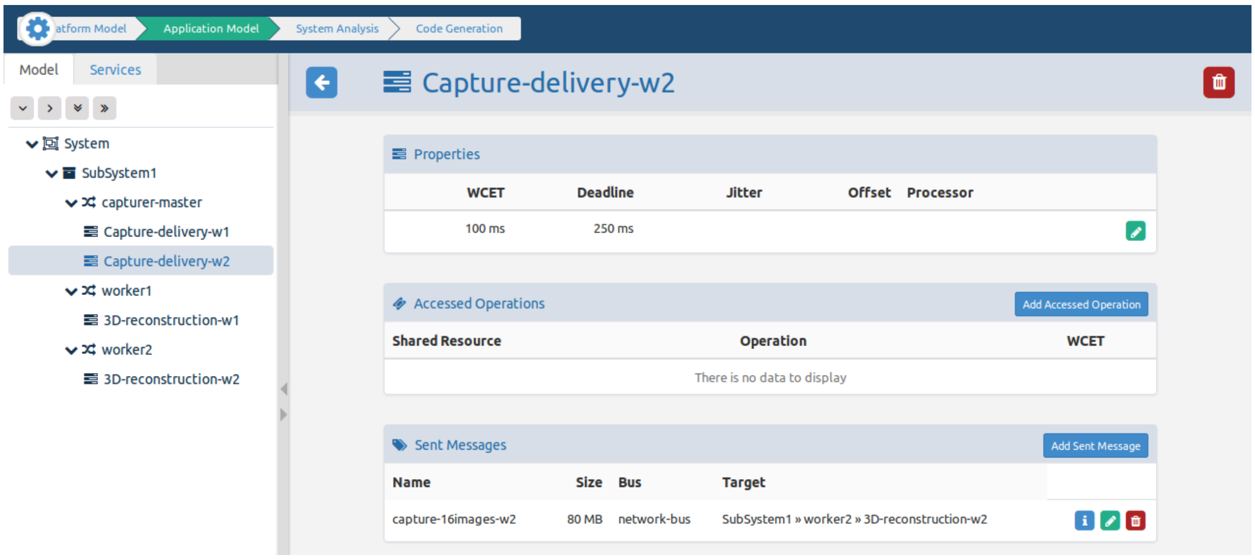
SubSystem1

Flows

Name	Period	Deadline	Processor	Priority	Tasks
capturer-master	500 ms	500 ms	zg3d-capturer	0	2
worker1	500 ms	500 ms	zg3d-worker-1	0	1
worker2	500 ms	500 ms	zg3d-worker-2	0	1

Figure 29 - "art2kitekt" - Application model of the final prototype

At Figure 29 a capture of the *"Application Model" stage* with a summary of the flows modelled with the "art2kitekt" tool suite and its corresponding tasks (tree structure at the left panel) is shown.



- System
  - SubSystem1
    - capturer-master
      - Capture-delivery-w1
      - Capture-delivery-w2
    - worker1
      - 3D-reconstruction-w1
    - worker2
      - 3D-reconstruction-w2

Capture-delivery-w2

Properties

WCET	Deadline	Jitter	Offset	Processor
100 ms	250 ms			

Accessed Operations — Add Accessed Operation

Shared Resource	Operation	WCET
There is no data to display		

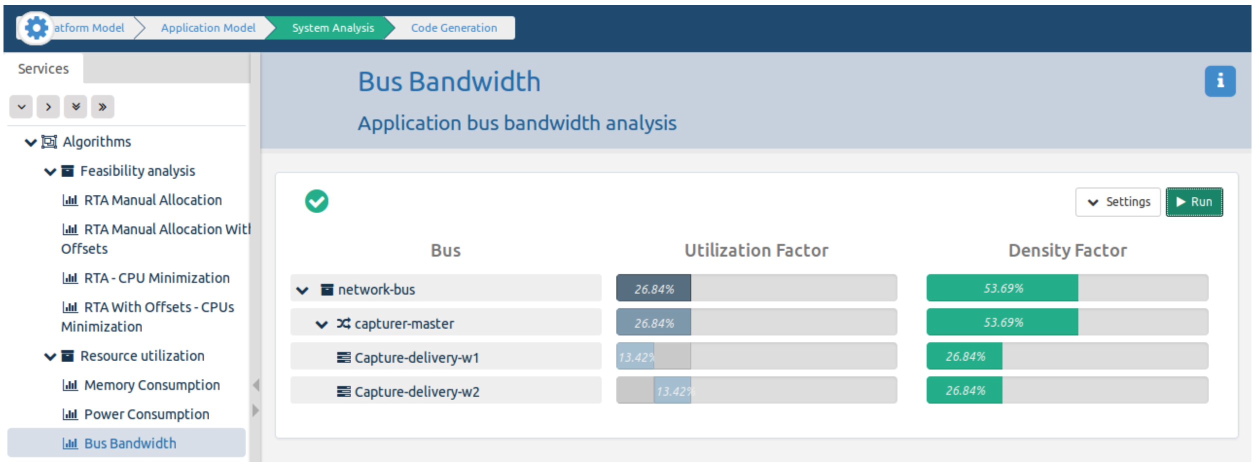
Sent Messages — Add Sent Message

Name	Size	Bus	Target
capture-16images-w2	80 MB	network-bus	SubSystem1 » worker2 » 3D-reconstruction-w2

Figure 30 - "art2kitekt" - Data delivery model with messages

In Figure 30 the detailed information of a task used to model data delivery is presented.

Finally, the "Bus Bandwidth" analysis results can be obtained from the ***"System Analysis" stage***, providing an estimate of the maximum number of captures that can be sent through the network, and thus the maximum number of 3D objects that can be reconstructed.



- Algorithms
  - Feasibility analysis
    - RTA Manual Allocation
    - RTA Manual Allocation With Offsets
    - RTA - CPU Minimization
    - RTA With Offsets - CPUs Minimization
  - Resource utilization
    - Memory Consumption
    - Power Consumption
    - Bus Bandwidth

Bus Bandwidth

Application bus bandwidth analysis

Bus	Utilization Factor	Density Factor
network-bus	26.84%	53.69%
capturer-master	26.84%	53.69%
Capture-delivery-w1	13.42%	26.84%
Capture-delivery-w2	13.42%	26.84%

Figure 31 - "art2kitekt" - Bus bandwidth analysis results

A snapshot of the "System Analysis" stage with the bus bandwidth results (utilization and density factors) it is shown at Figure 31. As it can be seen from the previous figures, the "art2kitekt" software provides helpful information to predict the occupation of the communication channels reached by a given task set and the available hardware resources.

5.2 Experiments with the final prototype

A battery of experiments will be run in order to confirm how **throughput** linearly increases with a growing number of available computing nodes as it is expected. At continuation, a table with a summary of the devised experiments is shown.

Computing nodes (or Workers) will have 4 cores. The available hardware has the following characteristics and provides 32 cores:

- Processors: 2
- Cores (at each processor): 8
- Multithreading capabilities: YES

Number of workers	Throughput
1	6 objects/minute
2	11 objects/minute
4	22 objects/minute
8	45 objects/minute

Table 8 – Increase of inspection throughput with number of "workers".

As our preliminary experiments show, for one "worker" a throughput of almost **6 objects per minute** has been achieved. The system took around 293 seconds to perform 28 3D reconstructions. The other figures only represent an estimation of the results we theoretically expect. We are still working on the whole process optimization and will provide the latest results during the public demonstration in Granada.

To simulate a situation where workers do not have necessarily the same number of cores and neither the same computing power, an experiment with 4 workers is devised. The first worker will have one core, the second 2 cores, the third 4 cores and the last 8 cores. The overall throughput should be a little bit lower than the one obtained in the previous experiment with 4 homogeneous workers.

Number of cores per worker	Throughput
1	< 6 objects/minute
2	< 6 objects/minute
4	**6 objects/minute**
8	> 6 objects/minute

Table 9 – Worker-by-worker throughput with number of cores available.

As explained before, this preliminary experiment shows the throughput of "one-worker-one-core". The row with 4 cores per worker is the same than the first row in the previous experiment in Table 5. A complete final report of this experiment results will be provided for the final public demonstration.

For the new set of experiments, the reconstruction process will include the steps described in D10.4:

1. Load cameras number and calibration.
2. Reading images from disk.
3. Remove lens-object distortion from images.
4. Segment silhouette.
5. Octree computing.
6. Surface marching cubes computing.
7. Centroid and alignment.

In the next deliverable a detailed summary will be presented with the quantitative results of the previously described experiment.

5.3 Fulfilment of Requirements

The main goal of this industrial use case is focused on achieving a **performance increase** by taking advantage of multicore execution platforms and parallelization capabilities of the task set. Next, the original requirements targeted within this use case are listed:

- **15Q-WP10-REQ005** "Algorithm parallelization" main algorithms shall be parallelized to reduce process latency in a multi-core execution platform.
- **15Q-WP10-REQ006** "System scalability" new dispatching stage shall allow the system to balance workload among different processing nodes to reach the required throughput.
- **15Q-WP10-REQ007** describes proper and precise synchronization between *the capture process* and *the data delivery process* to reach a maximum performance.

In previous deliverables, requirement 15Q-WP10-REQ005 was targeted. A first prototype was developed and improved as a proof of concept of how it is possible to enable **algorithm parallelism** for the 3D reconstruction software. OpenMP was applied to some functions of the software and an evaluation of the performance for intensive computation tasks on a variety of execution platforms was presented and compared. All this work allowed for a coarse exploit of **intra-node parallelism** and therefore it was useful to reduce system latency.

Now, also requirement 15Q-WP10-REQ006 has been addressed and preliminary but good results are being achieved with a new software architecture that allows the exploitation of **inter-node parallelism** by delivering full captures to different "worker" nodes and continuing with the next object inspection while a set of already captured and delivered objects are being processed into other "workers". In this way, throughput of the inspection system raises. Moreover, this **system scalability** improvement has also benefited from a technological development outcome from WP02 with the use of an offline analysis tool suite known as "art2kitekt" which has allowed to model and analyse the evolving system prototypes in different ways, from "Best Computation Time" analysis in a hard parallel task set to "Bus Bandwidth" analysis for the last system architecture.

6. Conclusions

Work in the different use cases in work package 10 has proceeded well. Updated prototypes developed in the second phase of the project address the requirements described in D10.1. In the final review we will demonstrate how well the requirements were fulfilled.

In a short summary, results in model driven design and implementation are very satisfactory. We have reached the goals in terms of development speed and quality. New architectures also seem to perform as expected in fulfilling both the real-time requirements and the capacity requirements. Development in security area is solid and will improve the security of industrial systems significantly. Communication and tracking solutions for mixed radio and protocol environment have progressed according to the set requirements.

Safety is the only area were our original ambitions will not be satisfied. We were looking for a flexible safety solution were additional hardware investments would have been minimal and flexibility would have allowed us and even our customers to combine different certified safe components into new designs without the need for recertification. We have made some progress. Use of non-safe communication channels are accepted as long as they are monitored and use of pre-certified components will speed the certification process. However, we will still far away of providing “programmable safety solutions”.

Based on the prototypes WP10 is on track in achieving the commercial goals set for the project. In some cases, the decision to develop new products based on the results has already been made.

7. References

[REQ_01R_NXP_001]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[REQ_01R_NXP_002]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[REQ_01R_NXP_003]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[REQ_01R_NXP_004]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[REQ_01R_NXP_005]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[REQ_01R_NXP_006]: Refer to EMC2_HL_REQ_NXP_WP10_task_10_2.xls in EMDesk Server
[FPP_EMC2_AIPP5]: FFP document of Artemis AIPP5 EMC2 project

[ICIT_2017] Aboelhassan, M. O. E., Bartik, O. and Novak, M.: Embedded Multi-core systems for mixed-criticalapplications with RPMsg protocol based on Xilinx ZYNQ-7000. In proceedings of ICIT 2017, sent for publication, in review process.

[IEEE_TIE_2016] Minar, Z., Vesely, L. and Vaclavek, P.: PMSM Model Predictive Control With Field-Weakening Implementation. IEEE Transactions on Industrial Electronics, vol. 63, no. 8, august 2016