Development of an embedded communication hub for sensor data acquisition in a robotic system

by

Juan Carlos Reyes Andrade

September 2020

Supervised by

MSc. Jannick Brockmann
Head of Electronics at Han's Robot Germany GmbH

First Examiner | Prof. Dr.-Ing. Bernd-Christian Renner
Research Group smartPORT
Hamburg University of Technology

Research Group smartPORT

Hamburg University of Technology

TUHH

Acknowledgment

Table of Contents

TABLE OF CONTENTS

ii

Introduction

This document describes the different stages through the development of an embedded communication hub for sensor data acquisition in a robotic system, that will be the starting point of a framework for the development of devices used within an industrial robot. In this document the prototype will be referred as *Axis Communication Hub* or ACB.

During this first chapter a brief introduction to the Real-Time Ethernet (RTE) industrial networks is presented, as well as a summary of the standards involved with comments about how they are related to each other. The second chapter shows a summary of the state of the art regarding the possibilities for developing open source projects according to the degree of openness of an RTE communication protocol. Moreover, the usage of these RTE industrial protocols in embedded applications and its relation to the Industrial Internet of Things (IIoT) necessities is briefly introduced. At the end of this chapter, a brief comparison of the openness of these protocols and how this is related to the development of devices is presented. The applications introduced have to do mainly with EtherCAT, as it is within the scope of this Research Project, showing advantages that will be detailed as the reader reads through this document. Afterwards, the third chapter deals with the technical specification of this Research Project and its development proposal, including the hardware available, firmware structure and the overall prioritization of the goals. Later on, during the fourth chapter, the main points related to the implementation process are presented. In chapter five the overall results are discussed, where the reader can find comments about the implementation and test challenges and their solutions. In the sixth chapter the conclusions are summarized. Finally, extra information focused on the technical details of the project, such as diagrams or protocol-related specifications, can be found within the appendixes.

1.1 The need of RT within industrial environments

During the last years an increase in the usage of the Ethernet-based field buses within industry has been recorded. This shows the expected adaptation of the industrial automation to the IT infrastructure, which is fundamental for the *Industry 4.0* paradigm and its consequent huge amount of data to be monitored, analyzed and controlled. This data deals at the same time with different time constraints and interconnectivity among the different layers of an industrial system and all their devices. Having in mind that the former *automation pyramid*, see in Fig. 1.1, where the different layers needed various gateways to communicate in a rather vertical approach, has been evolving to a one more flexible structure; it is then understandable that several technologies providing this access have been meeting each other while coming either from the top or the down levels. A good overview of the mentioned structures can be read in [SKJ18]. Nowadays, these technologies offer similar features regarding data access and security, each of them with their own development history, alliances and, therefore, standards.
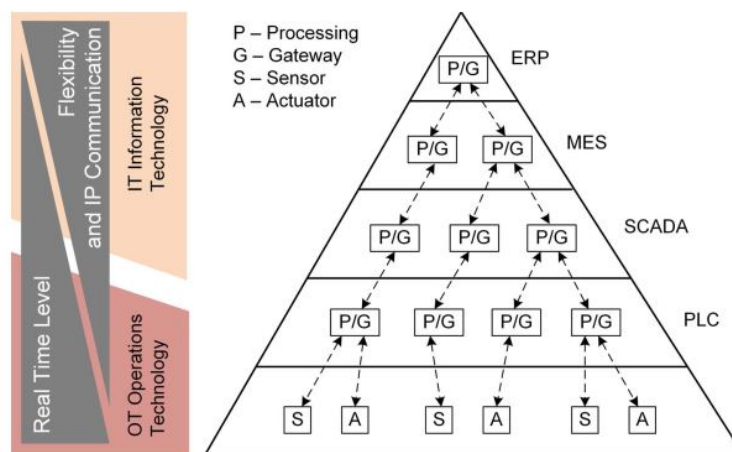
Figure 1.1: Classical automation pyramid structure, source from [SKJ18].

Coming from top-level-related frameworks, there is, e.g., the OPC UA project; whereas names like Profinet, DeviceNET, EtherCAT, Powerlink, etc, come from the field bus side—the lowest level. All of them have developed in an individual way as response to market needs, however meeting in the late decade through the necessity for unified standards to improve interoperability between the incredible number of projects. This happens at a time were information, technical as well, and development tools have become even more available and open to the end-user and the developer. Leading then now to a situation where the private initiatives are not any longer the full owners of the technology development. See Fig. 1.2 for the IIoT's approach.
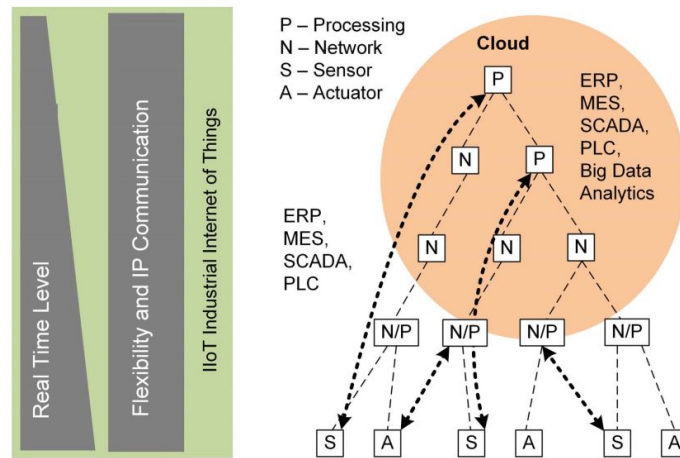
Figure 1.2: *Industry 4.0* a more flexible automation structure. Industrial Internet of Things, source from [SKJ18].

Another line of work, closely related to interoperability, is the Real Time (RT) applications in their both versions with *hard* and *soft* requirements. Nowadays, there is an increasing number of applications in robotics that demand control loops and device chains that require hard real time performance. Although these requirements are more common at the operational technology level, such as, robots, CNCs, servo motors, etc. They all face now the IIoT demands; hence, their networks should meet as well certain degree of RT capability. Moreover, synchronization of time sensitive systems within manufacturing lines, for instance, has been addressed for years by the RTE protocols and now this kind of features are increasingly been demanded as well at upper layers.

The current automation industry has many competitors and close technologies, as a natural consequence for specific processes requirements—depending on the industry; but also as a response of the market. However, the search for standardization can be tracked back to the 1980s, as the field buses were standardized by the International Electrotechnical Commission (IEC). This continued and Ethernet took its place within the industry. As an important note, during the last two years, according to the HMS Industrial Networks' annual study, the total market shares of new industrial nodes in factory automation increased for the Industrial Ethernet from 52% to 64%; while the field buses decreased in the same period from 42% to only 30%. Finally, the industrial wireless remained around the 6%, see Fig.1.3.

It is yet worthy to mention that the name *Industrial Ethernet* is used only as a generalization for the group of protocols that historically developed on IEEE's Ethernet specification; even though they all are almost no further compatible with each other—because of the modified Media Access Control (MAC) layers. Details about the differences is addressed in the following chapters.
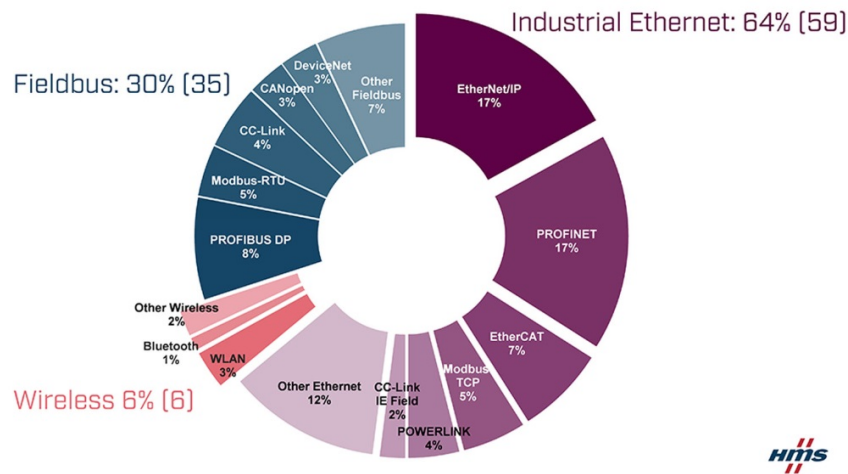
Figure 1.3: Industrial network market shares 2020 according to HMS Networks. Source from [Car20].

Vendor protected technology has its limit when there are plenty of possibilities for automation technologies, even if they are in ongoing development. For instance, as happened during the lifetime of the Open Platform Communications (OPC) project—predecessor of OPC UA—that was started only upon Microsoft Windows and as the time went by, the emerging needs made it change to use open standards and a multiplatform approach.

To introduce the reader to a common ground regarding standardization, the following section will present a brief summary of the standards that are of interest for anyone who wants to start developing using industrial interfaces.

1.2 Industrial standards and the TSN initiative

This section is intended to provide the starting developer a rough but useful reference of the standards related to industrial communication networks.

First, due to the historical and technological process of innovation within the information and communication systems, several parties have been related and, at some extension have merged results, bringing out an interconnected set of norms that thrive continuously onto a global standardization.

The following list is intended to be a quick reference to the current standards for Ethernet, legacy and current field buses, Time-Sensitive Networks in their American and international initiatives. This way, the reader has a roadmap to be taken into account for deeper research within the industrial applications. Furthermore, information related to the similar standardization processes between the IEC, ISO and IEEE, and their unavoidable cooperation, can be read in [JDH02].

ISO/IEC/IEEE 8802-3:2015 Revision of the Ethernet standard for half and full-duplex communication up to $100\,\mathrm{Mbit\,s^{-1}}$. Originally published by American IEEE 802-3 in 1985 and accepted internationally in 1989. The last revision 8802-3-2017/Amd 10-2019 includes MAC controls for $200\,\mathrm{Gbit\,s^{-1}}$ and $400\,\mathrm{Gbit\,s^{-1}}$. More information in [III17]. After 2019 The name Ethernet is not longer used, instead CSMA/CD or a reference to the corresponding ISO standard 8802.3 is the formal name.

IEC 61158:1999-2000 First international field bus standard published in 1999, where 8 *Types* of field buses were introduced addressing the Physical Layer (PhL), the services and protocols of Data Link Layer (DLL) and Application Layer (AL). Some included brand names were the following: H1/HSE/H2, ControlNet, EtherNET/IP, Profibus, Profinet, Interbus. This standard has an interesting story concluding with the signing of the *Memorandum of Understanding* by the main contenders to put end to the field bus war. The latter can be reviewed in [Fel02].The standard's most updated version in 2019 includes 26 Types of protocols, creating out of them the so-called Communication Profiles (CP), likewise grouped into field bus Communication Profile Families (CPF).

IEC 61784-Part 2:2008 It is extension for the RT capable CPs that are based on the IEEE 8802-3 standard. Commercial names included are the next: EtherCAT, Profinet, Ethernet/IP, Ethernet Powerlink, and Modbus TCP. An interesting overview of the current development of the industrial communication networks can be reviewed in [VZS19]. The SERCOS CPF is highlighted, since its third version is, altogether with the Ether-CAT profile, the fastest one in the list; providing as well a more efficient use of the available bandwidth with an open source resources. It shows even advantages over CAN devices due to its original design intended for hard RT motion control; for more information review [Des00] and [Ser20b]. This is a very interesting hard RT protocol whose applications might need further study out of the scope of this Research Project.

IEEE 802.1A/B/C/D/Q Time-Sensitive Networking standards is an initiative to improve the IEEE 802-3 in order to meet the industrial real time requirements, which story can be tracked back to 2005, as the IEEE 802-3 group was merged with the IEEE 802.1 Audio Video Bridging Task Group and started to work for industrial environments. This a response to the vast alternatives of the RTE CPs. About 60 individual IEEE standards oriented to improve the ISO/OSI layer 2, including 13 focused on its security, are within the scope of the TSN project. A nice introduction to this standard's scope can be read from TSN section in [VZS19]. The mentioned project covers the lower layers of the communication system, whereas the upper ones, representation and transfer of data, is addressed by OPC UA. Moreover, it is important to mention that this is an ongoing

project and still around 40% of its standards are in draft or preparation phase. For further information regarding the current standard development, visit [IEE20].

IEC/IEEE 60802 TSN Profile for Industrial Automation is the stand alone TSN base standard that will include the common advancements from IEC SC65C/WG18 and IEEE 802 work groups mentioned in the previous item. See [II20] for the current development group's activities. This is an ongoing project started around 2017, still being in a draft phase. Since this will be the international standard, it would be the equivalent to the effort once given during the creation of the IEC 61158 for the legacy field buses.

IEC 62541:2016-2020 Set of IEC standards for OPC UA. Individually, the IEC 62541-14:2020 defines the OPC Unified Architecture (OPC UA) PubSub communication model. It defines an OPC UA publish/subscribe pattern which complements the client server pattern defined by the Services in IEC 62541-4. IEC TR 62541-1 gives an overview of the two models and their distinct uses. For the concrete list of contents, review [IEC20]. An overview of this technology and how it is planned to complement the TSN can be reviewed in [PERK18].

State of the art

This chapter introduces some current applications mainly focused on robotics, since this area is closely related to the environment with which the ACB will be interacting. Robotics sees various advantages from the RT communication protocols, when it comes to integrate motion controllers and any other industrial peripheral. Afterwards, an overview about industrial development frameworks is given, yet focused not on the RT interfaces, but its specific software. The latter is of great importance, for the Real Time Operative Systems (RTOS) are a corner stone for embedded systems that need to provide a deterministic service within their environment.

2.1 Current applications

As rapidly mentioned in Sect. 1.2, the SERCOS motion control interface has been standardized within the CPFs of IEC 61784-Part 2. Furthermore, it has been even integrated to EtherCAT as a compatible CP. This service is available within the DLL and AL and is called Servo Drive Profile over EtherCAT (SoE), which provides access to motion controllers under the SERCOS specifications and, consequently, offers interoperability within its own RT features and the latter's hard RT capabilities. An example of this compatibility is presented in [XJY11], it shows that jitter of 30 microseconds is feasible in a control loop while the Master uses the SoE service.

Another interesting application has been the characterization of an EtherCAT Master within a RT control loop for Servo Motors, which run CAN devices over EtherCAT (CoE service). The implementation of the Master device ran on different open source Real Time Operating Systems (RTOS) based on Linux, namely, Xenomai and Linux with the *RT_PREEMPT* patch. It was concluded that both of the approaches were capable to achieve update periods of 1 ms , and an average jitter of 1.5 µs. Moreover, Xenomai could averagely achieve execution times around 100 µs; the mentioned data can be reviewed in [DC17].It is worth to mention that the

EtherCAT Master features were available in both RTOS kernels, for the IgH EtherCAT Master stack was running on top of them. This open source stack will be commented in 2.3.

The characterization and optimization of performance for different RTE profiles within TSN is a currently expanding topic, as the TSN standards and the RTE commissions are still working together. In [MGSR20] are presented simulations of TSN topologies with EtherCAT and SERCOS data frames, where the Quality of Service (QoS) is addressed and evaluated through the usage of Software-defined Network (SDN) switches. The approach of this project is to test different scheduling features given fixed cycle times for the data frames, which were proposed to be similar to the current real industrial applications in both technologies. In this manner, the importance of an unified network that supports different protocols is highlighted, but further research in this topic, including tests with other RTE data frames are still to be researched.

Besides robotics, a recent industrial application concerning CBD extraction equipment for high-performance large-scale processing, implemented distributed control and monitoring based on EtherCAT open protocol. This article can be seen in [Nor20].

Addressing the usage of open source tools, such as OS and RTE Protocols, for development of complex robotic systems, in [MHF$^+$20] is presented a *Motion Planning for Quadrupedal Locomotion*. This is roughly composed, besides the hydraulic actuators, mechanics and other peripherals, of two PCs on board with RT capabilities and shared memory. RT Linux (Xenomai) runs on both of them and take care of different levels of the control threads at two different rates depending on the tasks, namely 1 kHz and 250 kHz. The former rate is used for communicating with the motor controllers over EtherCAT interfaces.

Currently, Han's Robot Germany GmbH focuses on enhancing robots' cognitive abilities by developing in the fields of environment perception, drive technologies, control theory, material science, mechanical design and artificial intelligence. Interfaces within the robotic system rely on various industrial protocols to make its interoperability one of the key features. For instance, current motor drivers are linked over internal EtherCAT chain to the main controller.

The above mentioned applications are just a tiny number of examples that shows the importance of an already standardized open industrial communication protocol, within a broad set of fields that cannot be completely covered in the scope of this document. Nevertheless, it paves the road to understand why generating the know-how to any of the mentioned technologies, represents a high-impact resource for any research or development group, regardless of its commercial or academic purposes.

2.2 An overview about the RT capable SW in robotics

As mentioned in the previous section, several resources and examples showed the current usage of RT open source software and its community. Since this Research Project has a goal of introducing the reader a roadmap for RTE communication interfaces and its applications, this section was added to summarize the RT software for development in robotics.

The usage of middlewares within the field of robotics is growing and it relies on *robot software* that exists between the application and an RTOS, as detailed in the following articles [DC17] and [MHF+20].

A list of requirements is suggested in [JYJP20] to address the mentioned middlewares and how to consider them *Real Time Robot Software Platforms*. The list is as follows and it is useful to start getting familiar with the capabilities and features of the so-called Robot Software:

1. Data exchange support.

2. Real time support (strict periodic execution and sporadic events support).

3. Thread and process types for user defined programs support.

4. Easy configuration of applications (robot control SW, PLC SW, vision inspection SW, non-real-time SW, etc.)

5. Multiple periods for scheduling.

6. Threads or processes running in the same period are classified by priority.

7. Check and handle the event through the event handler.

Common names for different projects aiming to create these development frameworks are the following: Common Object Request Broker Architecture (CORBA), Real-Time CORBA (RT-CORBA), Data Distribution Service (DDS), OPC UA, Open Platform for Robotic Services (OPRoS), Open Robotics Technology Middleware (OpenRTM), Open Robot Control Software (OROCOS), and Real-Time Middleware for Industrial Automation devices (RTMIA). Further comments and a comparisson between their features can be seen in the previously referenced paper. As to what concerns to this document, only some of them will be roughly commented as they ended up being somehow related to the RTE profiles. For more information review [MLH+17].

OPC UA As frequently mentioned before, this is an open standard for data sharing among nodes within industrial networks and has been considered in some projects related to robotics. Nevertheless, it is important to highlight that this is not considered a full

middleware, since it only provides a protocol to control the exchange of data between nodes, a good degree of reliability and security. However, it does not provide RT capabilities to the system only compatibility. Hence, it needs an operative system and the consequent lower layers capable of RT scheduling and communication, concerning the latter the TSN set for protocols is an example.

ROS/OROCOS/OpenRTM These are projects that aim to create a suitable middleware for robots by implementing Xenomai or Linux operating systems. ROS prioritizes the final user, avoiding in the way some fine-grained features due to its difficulty, therefore having sometimes issues to meet the hard RT requirements. Whereas OROCOS has further improved its compatibility, similarly to OpenRTM.

CODESYS and TwinCAT To fully meet compatibility with the industry, the so-called PLC Software has been also used in open robotics. These applications essentially need to run both, the robot functional blocks and the robot tasks. For further details on it the following references can be reviewed [MLH$^+$17] and [MLT18].

xbotcore This is an attempt to provide of a highly compatible open middleware for industrial robotics, it runs over the Xenomai and uses a SOEM stack to interface with any compatible industrial device, recall 2.3. Applications have been already mentioned in 2.1, which have reached control loops down to 1khz for 33 axes. More details about the latter application in [MLH$^+$17].

The previous information was presented only to draw an idea for the non-familiar reader about the applications and, since this topic is in ongoing development and, furthermore, many other platforms are addressing similar challenges; the reader is invited to go deeper into these topics, for instance, by reviewing this resource [JYJP20].

2.3 Approaching openness within the RT protocols

Among the industrial standards mentioned in 1.2, there are some related initiatives to include a certain degree of open source software to improve the development of applications. The following is a brief list of a few interesting references to them. However, as expected, most of the software stacks for industrial communication systems are commercial and provided by third-party companies.

OSADL Open Source Automation Development Lab eG (OSADL):It is a German group that intends to lead the development of open source development for industrial automation. Closely related in the developing of OPC UA and other Linux features for industrial applications.

open62541 Within the official scope of OPC UA, there is this Certified SDK project that
is within its second phase, at which it is expected from the research and industrial
community to develop applications to test its performance. Moreover, as the TSN
specification is of huge importance, a set of enhancements for the open62541 project
were developed by Fraunhofer IOSB and series of patches for the Linux kernel have
been released to make it an RT compatible. To review the overview of the project, visit
[OSA19]. The OPC UA is developed under GPL 2.0 license and due to its current phase
implies a further adaptation for the physical node, e.g., ARM arquitechtures to make
them compatible with the mentioned patches.

SOEM/SOES RT Labs Industrial development group focused on Software Stacks for industrial
protocols. Among their commercial communication stacks there are software stacks
under GPL for EtherCAT Master and Slave devices SOEM/SOES. More details about
these stacks can be found in [RT-20].

Sercos Stacks Sercos III technology is able to be operated in a common TSN-based network,
since its development group has been working closing together with the TSN group;
a more detailed scope can be seen in [Ser18]. They also made available open source
software dedicated for development of master and slave devices, namely: Common
Sercos Master API (CoSeMa), Sercos Internet Protocol Services (IPSS), and The Sercos
III SoftMaster, the latter even allows the host to use any standard Ethernet controller.
It is important to mention that there are testing tools to certificate those devices and
achieve a Safety Integrity Level 3 (SIL-3). A description of the different available
libraries is provided in [Ser20a].

EtherNet/IP Stacks There are several commercial stacks that comply with the Open De-
viceNet Vendors Association's (ODVA) EtherNet/IP specification. OpEner is an open
source alternative which targets PCs with a POSIX operating system and a network
interface. Integration examples are provided only in Linux and Windows in [OpE20];
nevertheless, a variation for embedded systems has been presented for an STM32 mi-
crocontroller. In the mentioned project, more tools had to be adapted, for instance,
STM32F4x7 Ethernet Driver v1.1.0, lwIP v1.4.1 (TCP/IP Stack), MicroHTTP v5.1.0.1,
a patched version of OpENer v1.2., among others. A detailed description of the require-
ments can be reviewed in [emb20]

IgH EtherCAT Master This is a bundle of libraries to give a Linux host (LinuxCNC for
example) EtherCAT Master features, it is developed under the GPLv2 license. An
interesting example of this open source resource within an Airbus Test Rig can be
reviewed in the following reference [HWHP].

As the scope of this Research Project is only focused on the industrial communication profiles capable of RT and, so far has been clear how the EtherCAT is a reliable one, yet open and significantly considered in the industry —recall Sect. 2.1. Hence, it makes sense to invite the reader to read the introduction to the protocol itself in the Appendix A. This way it is easier to go sensibly to the implementation of what is one of the basic chain-elements in what could become a very complex application: an EtherCAT Slave device with open source elements.

Design requirements and approach

This chapter presents the main goals of the project and the technical specifications. It is also intended to give the reader a summary of the proposed functional modules and its structure. Any other constraint not mentioned here was adjusted or set while being within the design-test loop, as consequence of the prototype nature of the project.

3.1 Objectives

The main goal of this project is to develop a device using open-source tools to read out sensor data from a robot axis that can be interfaced with an RTE Network. Such that this device can be used afterwards as a test platform within an industrial environment to characterize its compatibility with the ongoing IEC/IEEE 60802 TSN Profile for Industrial Automation.

To achieve the main goal the following has to be carried out:

- To specify the requirements of the system

- Comparison considering the state of the art

- To develop the embedded system as a functional EtherCAT Slave Device

- To design and manufacture a PCB prototype

- To test and report the overall system functionality

3.2 Technical specifications

In the table 3.1 the requirements for this project and their state after the implementation can be observed. This comparison helps as a good summary of the overall achievements that are further detailed in the next chapters.

Feature	Requirement	Implementation	Remark
Upper layer interface	Ethernet/EtherCAT compatibility. Non-safety relevant. Services and synchronization: -	EtherCAT slave. Services: Mailboxing and CoE Synchronization: Free Run and SM	✓ FoE and SD synchronization possible in the medium term.
Display/signaling	LED stripes with serial interface: WS2812 2 Ch	2-4 Ch modifiable in SW Animation capable	✓ Chs can increase up to number of DMA-Timers (8)
Temperature	Data interface for 1-Wire bus	1-Wire Master 15 sensor in bus	✓ 6 Sensors simultaneously tested
PCB	PCB Prototype Layout and size: -	Attachable PCB for LAN9252-EVB-SPI. Size: 55 mm × 38 mm	✓ Second layout with both chips included possible in the short term.
Safety	n.a.	Non Safety-Critical for this prototype	– FSoE could be researched in the long term.
Extra interface	SPI or I2C interface for current/IMU/black channel	Extra SPI considered in PCB and SW JTAG/SWD compatible interface	✓ Currently working
Speed/position	Possible interface of BISS-C type	Not required for this prototype	–
Refresh data cycle	-	No hard RT deadlines. Deterministic refresh cycle of ∼10 ms by RTOS. Timeout faults handling.	⊕ Timing through RTOS
Data structure	-	Functional and parametrization data structure as Object Dictionary. Standard ESI file.	✓ Currently working
FW programming	-	CMSIS - FreeRTOS for thread, event and time management.	⊕ Currently working

Table 3.1: Technical specifications;

Regarding the functional safety features of the device, it is important to mention that, even though this device is considered as non Safety-Critical within this prototyping stage, the means to create a framework that could be extended to address further reliable development are taken into account, namely by considering fault tolerance within the software. More comments about this will be done in the results section.

3.3 Solution proposal

In Fig. 3.1 the proposed layered structure of the functional blocks can be seen, this will help the reader localize how is the level of interaction of each of the parts that are described along the document. For instance, the project itself focuses mainly on the non-dashed blocks; nonetheless, they still need to be configured or adapted according to ACB features. The layered representation shows three general concepts, namely physical or hardware, middleware and application software. Even though, the blocks help the reader differentiate between functionalities, most of them are interacting between each other, mostly the Devices' State Machines (DSM).
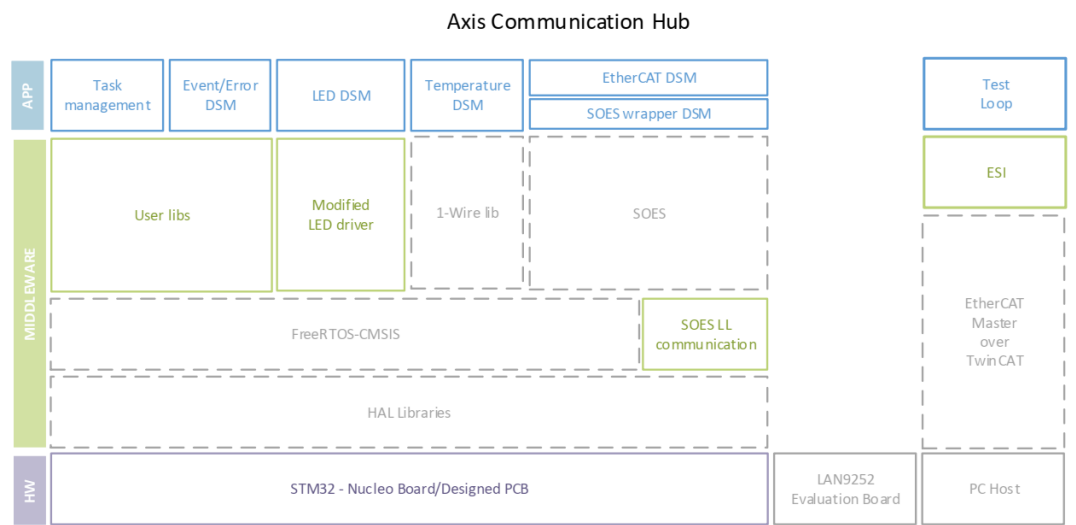
Figure 3.1: Layered structure of the proposed functional blocks.

3.4 Hardware selection

In this part it is presented the base hardware that was available to develop the prototype. The microcontroller (MCU) was chosen due to its active community, resources and current on-develop projects. Other MCUs were though considered, since the overall characteristics were somewhat similar and generic—regarding peripherals like serial interfaces or Direct

STM32F446ZE	LAN9252
ARM®32-bit Cortex®-M4 + FPU + Chrom-ART™ Accelerator Up to 180MHz CPU 512 kB of Flash 128 KB of SRAM	EtherCAT slave controller with 3 FMMUs and 4 SyncManagers Distributed clock support 4KB of DPRAM
General-purpose DMA Up to 17 timers Up to 4 × I2 C interfaces Up to 4 USARTs/2 UARTs Up to 4 SPIs 2 × CAN (2.0B Active) USB 2.0 full-speed device/host/OTG	100Mbps Ethernet transceivers compliant with IEEE 802.3/802.3u (Fast Ethernet) 8/16-Bit Host Bus Interface, indexed register or multiplexed bus SPI/Quad SPI Digital I/O Mode Multifunction GPIOs
LQFP64, LQFP100 and LQFP144 packaging	Pb-free RoHS compliant 64-pin QFN or 64-pin TQFPEP packaging

Table 3.2: Summary of the characteristics of both STM32F446ZE and LAN9252 used in the prototype.

Memory Access (DMA) and processing power and memory, for instance. Therefore, MCUs from Infineon and Texas Instruments were good possible candidates—see technical information in [Inf18], [Gro04] and [Son20], respectively; however, the basic familiarity with the STM32CubeIDE and the related ST technology was a crucial factor. In addition, and concerning the chosen MCU as well, the learning curve is not negligible when it comes to develop any firmware at a fair level, even more when it deals with other to-learn tasks, for instance, development with RTOS, modification of open libraries, EtherCAT protocol itself and the Network Controller chip.

Regarding the Network Controller, the LAN9252 belongs to a set of ASICs that are verified and certified by Beckhoff Automation GmbH. For a further reference for other alternatives visit [Bec20c]. The LAN9252 integrates a so-called EtherCAT Slave Controller (ESC) and it represents a good alternative to the Beckhoff's original ASIC ET1100. This way, the basic hardware is there to fulfill Han's Robot Germany's proposal for developing industrial compatible devices that could enhance the prototyping process within the electronics department. Moreover, the mentioned ASIC has a wide compatible control interface that make it be suitable to any microcontroller with which the developer has experience. The Table 3.2 lists the main characteristics of the above mentioned hardware.

3.5 PCB design

As it can be seen in the figure 3.2(b), the evaluation board includes on-board male pins, this was taken as an advantage and the PCB to be designed consisted on a pluggable PCB that would be mounted on top of it, increasing minimally the volume already occupied by the evaluation board. This idea needed to be designed taking into account the minimum of
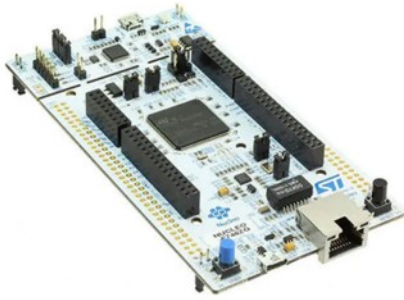
(a) NUCLEO-STM32F446ZE   (b) LAN9252-EVB-SPI

Figure 3.2: Evaluation boards for prototyping.

components based on the Nucleo-STM32F446ZE original design and the requirements of the LAN9252. This means that it had to provide, both 5V and 3.3 V power supply, physical ports for the prioritized communication capabilities, minimally SPI, One-Wire JTAG and the LED ports according to the technical specifications 3.1.

3.6 Firmware structure

Since the compatibility with the EtherCAT protocol is the highest priority, all the tasks related to the adjustment of existing libraries and the synchronization between them are also prioritized, such that the main functionality can operate. Taking this into consideration and recalling the final proposal for the structure of the embedded system, Fig. 3.1, the functional blocks are represented differently. The ones in gray or dashed lines are mainly components that are planned not to be modified at all or not in deep, because of either its complexity or its given reliability. This means, its functionality is almost granted. Nevertheless, the progress relies on documentation that can be either good or poor, for instance, TwinCAT has good resources, whereas SOES does not. Regarding the latter, the block refers to the main library to be integrated, that enables the EtherCAT features; although, it is represented with dashed-lines, it was changed but not restructured. However, detailed information about these concerns are presented along the implementation and result chapters.

The word firmware rather generalizes the set of written/modified code that needs to be carried out through the implementation, for instance, the top layer in blue corresponds to the application-related source files written in C, while the Test loop executes over TwinCAT3 and is written in SText, which needs mandatory an EtherCAT Slave Information (ESI) file written in XML. The name middleware, in this case, represents not only the Hardware Abstraction Layer (HAL) libraries provided by STM, nor the CMSIS-RTOS's, but also represents the user-define libraries for each DSM and the modified libraries, namely those for the LED and

17

One-Wire—as well as the already mentioned case of SOES. This structure is important to have in mind, mainly while going through the implementation chapter.

Additionally, another thing to consider within this document is the abbreviation *DSM* (Device's State Machine) which is used as substitute for State Machine, such that it is not confused with Synchronization Manager, defined as well by Beckhoff Automation as SM.

Implementation

The following chapter documents the different functional modules that were implemented according to the proposal. The tasks related to EtherCAT compatibility and usage of RTOS are highlighted within this and the following chapter, as they were of higher priority.

Moreover, the functional blocks or modules interact with each other through the DSMs, which has been added as well as a module for its understanding.

4.1 LED Control

In order to notify to the user the current state of each axis, the robot includes one or more LED rings. These rings are a serial array of LEDs that are programmed and controlled through a serial communication protocol. Basically, the final implementation is an adaptation of a library that uses a PWM peripheral to generate a signal that is modulated according to the data that controls the LEDs, namely digital 1s, 0s and reset as a specific duty cycle. The input pin of the first LED is connected to the MCU and depending on the modulated data, it will be passed over the next LEDs, being the first LED's output the input of the next til the last component. A summary of the protocol is explained in [Mik20].

Furthermore, the current LED control in the robotic system uses MCU Devices without communication capabilities leading to static status indication that cannot be set from the Master.

In the next paragraphs a summary of the activities that were carried out during the implementation is presented.

First control tests Learning the basics of the interface used to control one LED

Code for one LED control Using the peripherals of the MCU simple routines were written to set different basic colors in RGB. The peripheral used were a two Timers (hardware libraries) to keep control of the data timing and refresh rate.

Search for libraries Once the basic communication was understood, it was clear that the usage of libraries would be more practical, since the first approach were not suitable for higher number of LEDs and effects; moreover, the CPU was kept busy while waiting the timer state polling. The libraries found were aiming roughly either 8/16 bits processors whose main task was controlling the LEDs, or more complex libraries that used DMA modules within 32-bit processor. One of the latter implemented modulation of the data over a PWM and a DMA peripheral for one LED strip.

First library selection Due to the inexperience working with DMA modules and as the LED control was not of a higher priority compared to other tasks, it was decided to run first one of the basic libraries to achieve a multi LED control. The implementation was able to control a set of 20 LEDs with the processor mainly polling the Timer states.

Further control tests Despite the success of the library with one channel, the overall structure of the basic library was not easily portable to the proposed solution using FreeRTOS. Additionally, the DMA libraries showed afterwards being easier to modify as they were designed with a more abstracted and multi-purpose approach.

Second library selection As the usage of DMA became clearer, it was decided to improve the approach by selecting a 32-bit processor based library, that implemented only general control functions to avoid massive code lines related to effects and other rather unnecessary features, yet structured enough to be easily adapted. The final result was based upon the WS2812 Library for STM32F4 from Uwe Becker, see [BB17]. Main modifications were the addition of multiple channels capability and global flags needed for synchronizing with DSMs.

4.2 Temperature acquisition

Similar to the LED control's library, the temperature readout needed a library to be modified to match the current project.

First readouts Working with the temperature sensors was the first task in schedule, so it helped train the basic usage of the IDE STM32Cube, along with the hardware configuration of the MCU and the HAL libraries. The sensor uses a one-wire serial protocol, which similarly to LED Control's first approach was implemented by using timers for controlling 1s and 0s high levels, a continuous polling of data and a general while loop approach. This method worked as intended but it was known from the beginning that it would not match the multi-tasking proposal. However, it was of great importance to get to know the hardware and software, besides more functions were needed to access the sensor's ROM needed e.g. for identification.

FreeRTOS first tests Short after the working code was used to do the first tests with the RTOS, in this manner the code was translated as a Task (Thread as called by CMSIS) and some features like prioritization, task attributes, task handling and signals were tested with other generic functions, e.g. clocks and PWM generators. However, this implementation was not able to handle multiple one-wire devices due to its absence of CRC comparison.

Integration of library Finally, it was decided to adapt one open source library designed for STM32 processors. This is based on the principle that UART speeds at $9600\,\mathrm{bit\,s^{-1}}$ and $11\,200\,\mathrm{bit\,s^{-1}}$ suit the One-Wire timing, such that the detection of One-Wire devices and communication process can be downloaded to hardware already included in many general-purposes processors using USART. The integration of this library is from design compatible with RTOS, namely with CMSIS-RTOS. The library selected was developed by Tilen MAJERLE, review in [Maj20], and it was barely modified as it contained already the desired functionalities and the development focused only on the integration into the DSMs and usage of it.

The strategy the final library is based on is rather interesting and more details can be read in [Max02].

4.3 EtherCAT Slave communication: SOES adaptation

This functional module had the highest priority, therefore most of the effort given was focused not only on the library itself but the protocol and the hardware commissioning, it is hence recommendable to read through the introduction to the protocol in Appendix A. In this sense, the current section was structure as follows: first, a bit more technical details are presented regarding the EtherCAT specification, and some constraints for the prototype such that the library could be tested accordingly to the scope of the project; afterwards the EtherCAT Slave Controller (ESC) is briefly summarized and finally, the main points of the implementation are presented.

4.3.1 EtherCAT data consistency and constraints for design

This subsection describes both the features with which the *Axis Communication Board* has compatibility, and a summary of the mechanism that the protocol implements at the low level to work with the data exchange between Master and Slave. The constraints that were set were part of a live process that ran all along the learning process of the protocol. This is important to mention, since the understanding of the protocol leads to a sinful selection of the features

that a device should have implemented. Therefore, the understanding process was a natural consequence of the integration of the SOES library.

The reader may recall the set of Communication Profiles that are available within the EtherCAT field bus. A slave device must comply at least with CoE and the Mailbox, whereas the Master may comply with all the communication profiles. This of course needs to be suited to the requirements of the application and the degree of flexibility that is to be achieved. From it, the Mailbox and CoE are the main features with which the Axis Communication Board works. Leaving aside for future integration the FoE and EoE, the former would make possible update the device by sending a firmware binaries to the device's bootloader; whereas the latter would make the ACB accessible for any IT tool based on TCP/IP. Moreover, the scope of this prototype covers the Free run and SM-Synchronous mode, as they are the basic ones for communication Master-Slave, recall the graphical representation of synchronization modes in Fig. A.3.

The Synchronization Managers play a key role, therefore the correct setup of the SMs ensure the consistency of the data and needs to be linked correctly depending on the specifications of each type of ESC and SW Stack that are being used, this information is also linked to the CoE Object Dictionary (OD) and EtherCAT Slave Information (ESI) file—a roadmap for the Slave development can be reviewed in [Bec18]. The last mentioned activities were the main challenge within this project.

4.3.2 SOES library

As briefly commented in Sect. 2.3, the types of licenses allow open development and integration of software. SOES software stack was written in C and published based upon the GPLv2, which is a Copyleft License. However, the tools developed by the Open EtherCAT Society which support the design, implementation and certification of EtherCAT slaves using the mentioned stack are commercial ones. A significant part of the challenge covered by this Project Research was to achieve the EtherCAT Slave functionality in the prototype without those tools, as the protocols are open. In the Table 4.1 can be seen the main features abstracted and available in the stack, as well as the overall tasks to carry out for a device to work properly.

4.3.3 EtherCAT Slave Controller (ESC): LAN9252

As part of the available hardware introduced in Chapter 3, the LAN9252-EVB-SPI is an evaluation kit for the ASIC LAN9252 manufactured by Microchip. This IC is an EtherCAT Slave Controller with 4 kB of Dual Port memory (DPRAM) and 3 Fieldbus Memory Management Units (FMMUs). Each FMMU performs the task of mapping logical addresses to physical addresses. The EtherCAT slave controller also includes four SyncManagers to allow the exchange of data between the EtherCAT master and the local application, further technical

Features	Requirements
EtherCAT State Machine	Build up the SII-EEPROM Data-Layout
Mailbox Interfaces	Create the ESI-file
CoE	Port libraries to the STM32 using HAL drivers
FoE + bootstrap template	Use FreeRTOS for scheduling (Hardware Requirements RAM>64 kB)

Table 4.1: Features of SOES library and the overall requirements to make it work.

details in [Mic15].As briefly summarized in Sect. 4.3.1, each SM direction and mode of operation is configured by the EtherCAT master. Two modes of operation are available: buffered mode or mailbox mode. In the buffered mode, both the local microcontroller and EtherCAT master can write to the device concurrently. The buffer within the LAN9252 will always contain the latest data. If newer data arrives before the old data can be read out, the old data will be dropped. In mailbox mode, access to the buffer by the local microcontroller and the EtherCAT master is performed using handshakes, guaranteeing that no data will be dropped. The overall structure of the ASIC can be seen in Fig. 4.1. This prototype works with the buffered mode and uses SPI as Process Data Interface (PDI).

4.3.4 Development

Once explained the general information regarding the Communication Profile, the library and the hardware, the following lines will list and expose some of the most relevant information during the integration.

Porting of low-level functions All the variations of functions for reading and writing ESC's registers (directly and non-directly addressed) are needed to be defined. HAL libraries can be used, DMA, interruption or timeout based, nevertheless, tests are required for library performance.

First tests Before integrating the SOES library, basic tests with self written functions over SPI-DMA and SPI-timeout were compared by accessing to test registers available in the LAN9252.

Selection of the features At the same time, in order to have the SOES library running, the features it includes needed to be selected depending on their usage, complexity and other things. For example, there are some MCUs that include EEPROM memory, that is mandatory for the implementation of FoE service. In the case of the STM32F4xx
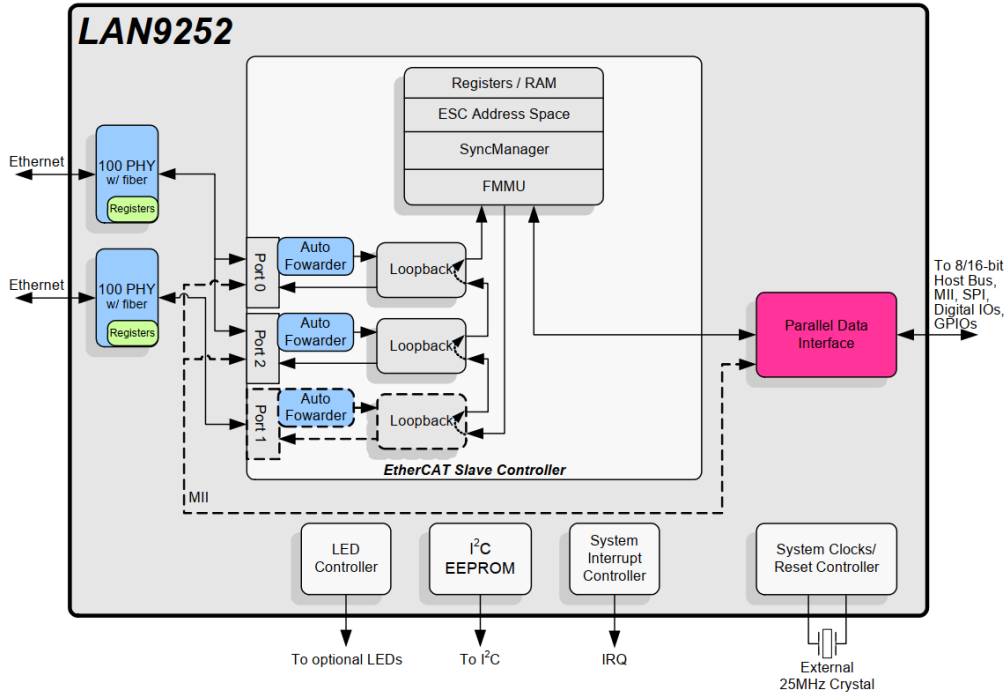
Figure 4.1: Internal structure of the LAN9252, notice the PDI and EEPROM's location.

that does not have any EEPROM, the flash memory can be used instead. However, this approach represented extra effort in this early stage of the development. Therefore, in addition to the initial requirements, this service was avoided and the SOES integration could continue in a relatively lighter way.

Second tests Once understood the logic behind the library, SPI commands are sent in interruption mode and communication with the ESC was tested for the directly addressable registers, namely the ones related to configuration of the PHY and general chip configuration, not the ESC functions.

Third tests with the Master At this stage a compliant EtherCAT Master was configured through a PC running TwinCAT3. In order to ensure a reliable configuration two different EtherCAT devices were connected synchronizing their data with the Master. Namely, a commercial 3-Phase Motor Controller and an in-house multi-protocol end-effector tool. For those different devices, data structures were declared and very simplistic update loops were programmed within the XAE (TwinCAT) environment using *SText* programming language.

Creation of an ESI file Since the EtherCAT Core registers are indirectly addressed and only available till the ESC has been firstly readout by a Master, the ESI file needs to be

defined and loaded to the EEPROM of the ESC. In this way, the Master can compare and verify its compatibility and the described Data Object Dictionary. The latter is closely related with the mentioned ESI file, since they are both in the same file. The available information and tools provided by Beckhoff are originally designed for Beckhoff's ET1100. Microchip in turn provides some test examples running on PIC32 MCUs in specific development boards. The challenge was to analyze the available information to adjust the configuration files to the LAN9252 interfacing with the STM32 MCU. In D the main structure of the created ESI file can be consulted.

Object dictionary The object dictionary was also included in the SOES library, matching it to the one contained in the ESI file, but mapped according to the few documentation available of SOES.

Fourth tests Longer tests and configuration loops were located at this stage due to the deepening on the protocol. Constant comparisons between the data read by the Master and the data received by the MCU host took place.

4.4 Device State Machines (DSMs)

In order to have a deterministic behavior of the embedded system, a set of State Machines (DSM) -not to be confused with Synchronization Manager- were proposed and implemented as part of the project library. The DSMs software implementation follows a *switch case* comparator approach, since it was simple, yet effective and flexible enough, to work during the prototype. These characteristics were very important, since the DSMs structures were in constant change as the integration of new libraries and the functionalities developed. The proposed DSMs are as follows and the diagrams can be reviewed in Appendix B.

LED Initializes and updates periodically the RGB value of the LED Rings. See in Fig. B.1.

Event Handler Its purpose is to react to notifications or errors that could appear within other DSMs and notify to update the LED rings in accordance. The approach of having defined this DSM was mainly thought for fault handling and will be the base for the inclusion of future features, e.g., receiving commands or interruption requests from any of the interfaces. See in Fig. B.2.

ECAT Initializes the EtherCAT communication and activates the SOES App. It is important to mention, that this DSM is rather focused on synchronization with the SOES state machine. The latter changed as the development advanced, since the native infinite loop the SOES library is based on had to be adapted. The two involved DSMs can be seen in Fig. B.3.

Temperature Initializes and runs the temperature related functions. It relies primarily on the
open-source library described in Sect. 4.2. The DSM can be seen in Fig. B.4.

As to the representation of the DSM, it is important to consider the general two approaches
for Finite State Machines, namely Mealy and Moore, since both of them provide advantages
while abstracting the desire logic of the different functional blocks. In this rather practical
approach the formalities are not fully met, for instance, to choose strictly for transitions
dependent on the state and inputs with actions, but using exit actions as part of each state,
when it is convenient. There are though transitions that explicitly executes a synchronization
edge. This flexibility was opted due to the inherent interconnection of the DSMs as they are
not fully independent. To meet a suitable representation, the previously said is integrated
with the approach of UPPAAL software to model timed automata and the reader is invited to
look it up in [BDL06]. As a summary, since the modelling of automata for real time systems
demands a synchronization feature, this is represented and attached to the edges between
locations—the latter in the sense that a state is a location constrained to a valuation of time
and other variables—with the symbols ? and !, the former acting as a wait action, while the
latter as a notification. Those, clearly can be understood as signals between different threads.

4.4.1 Scheduling

In the present section, the main points regarding thread management and scheduling is
presented. All the DSMs were implemented as *Threads* using CMSIS-RTOS on STM32. All
threads have fixed priorities and the desired execution time (*release*) is controlled to each
thread through the OS-native delay function. The previously mentioned function is not to be
confused with the *HAL* version of it, since using HW-related functions while executing an OS
is conveniently avoided. The OS-function allows the scheduler to allocate CPU resources to
any next-priority tasks. For further information regarding HAL and CMSIS implementation
of delay can be seen in [STM17] and in section *Time Management* of [Arm20] respectively.
The time constraints are defined as *desired*, since the system is non Safety-Critical—recall
the safety specification in Table 3.1; hence, it has no hard real time constraint and the overall
execution follows a best-effort approach. This, however, opens the door to further improvement
in the sense of characterizing and optimizing the reliability and task execution; the latter will
be commented in Sect. 5.2.

In Table 4.2 are presented the basic timing requirements depending on the functionalities
of each DSM related to the thread. Each parameter that sets up the duration can be changed
in a header file, see the Appendix E, and the individual functionality can be reviewed in the
previous DSM section.

A final list of priorities and threads is presented in the Table 5.3 within Chapter 5.

Thread	Release period (ms)
SOES APP SDM	5 – 10
LED SDM	33
ECAT SDM*	100
Temperature SDM	1000

Table 4.2: Basic timing requirements for threads, deadlines are rather desired since the device is non Safety-Critical. *ECAT SDM is mainly event driven, nevertheless, in the connected state it has a periodic update*

4.5 PCB development

In this section straightforward information regarding the manufacturing of the PCB proposal is given. The schematics and PCB layout can be consulted within the Appendix C. The main idea around this design was introduced in Sect. 3.5 and has the purpose of providing experience in embedded hardware design and a base work for coming projects, where the functionalities of this prototype will be merged with other boards. Therefore, to have a physical prototype to recognize possible opportunity areas, such as physical connectors, sizes, power source quality and signal integrity, is a corner stone for any future design. The overall stages of this design are as follows:

Design Altium Designer was used to prototype the PCB for this project. During the process the first approach was based on the open files for both the NUCLEO-F446ZE Development Board by STM and LAN9252-EVB-SPI by Microchip. By analyzing the general diagrams and selecting and adapting the different modules to adapt the requirements was the main challenge. To ensure usage of less extra devices as possible, two voltage regulators were included for 5 V and 3.3 V. To meet the routing needs a four layered PCB was selected. The final 3D model can be seen in Fig. 4.2.

Manufacturing Due to practical reasons, the board manufacturing process was in charge of an external PCB manufacturer. In respect to soldering, it was made in-house.

Testing I The overall integrity and functioning of the Power-on and SWD-Programming of the STM32 MCU via SWD/JTAG connector on-board was firstly tested with good results.

Testing II By this stage, the readout of directly addressed memory space, specifically test and ID register, of the LAN9252 had been already done with the NUCLEO board. In this manner, the code was programmed onto the ACB and so, the SPI communication gave good results. Moreover, the PWM Outputs over the two channels for WS2812 LED control and the 1-wire connection were also physically tested.
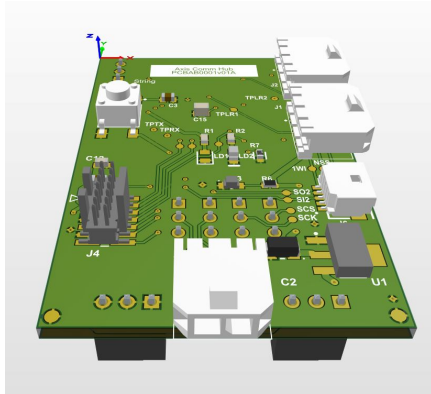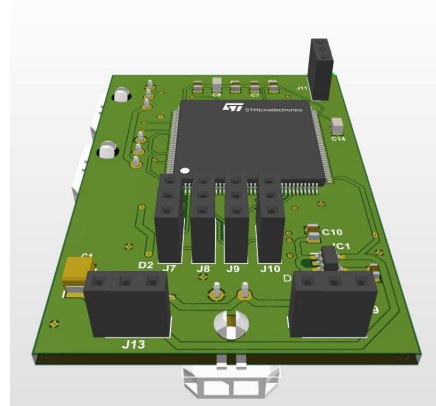
(a) ACB 3D view top

(b) ACB 3D view bottom

Figure 4.2: 3D model generated by Altium for the final design.

Testing III This phase is an ongoing task, since depending on the development of all the features, communication speeds and hardware configurations, different scenarios are continuously emerging. A deeper analysis will be taken into account for next versions.

Results

This chapter starts where the previous chapter left for each of the functional modules, having in mind that the results for the temperature are mixed with those of SOES, as the data that is being transferred is now only generated by the sensors. Some photos and screenshots are presented as well. Challenges, their solving and further improvements are also commented along the chapter.

5.1 PCB, SPI and LED control

In Fig. 5.1 the final PCB prototype can be seen. It is worth mentioning that one issue emerged related to the physical requirements of the temperature sensors. According to the library integration process commented in Sect. 4.2, during the first month a rather simplistic approach was coded to have access to the temperature values, the sensors in this approach used an external power source. In contrast, the final sensors were cabled in such a way that they rely on parasite-powered circuitry. Additionally, the final library that was included made use of the UART peripheral in a full duplex mode, which means that two independent pins were explicitly needed, namely RX and TX. That also differed from the first approach's configuration mode. Given this differences, only one available GPIO pin with no extra pull-up transistor in the PCB side, was not appropriate to test correctly the new one-wire setup. Nonetheless, arrangements were carried out to use the UART RX/TX pins available in the JTAG connector and add external resistors. This approach was merely for testing and will be corrected in any further development out of the scope of this Research Project.

The Process Data Interface (PDI) took into consideration the minimal data rate calculated for unsigned integers of 16-bit long that would be interpreted by the Master. This information is summarized in Table 5.1.

Another issue that is good to remark, is the noisy communication that emerged at SPI higher speeds. Generally, the monitoring of SPI signals was very frequently, since the possibility of a fault increased after any new modification, and those needed to be traced back to their
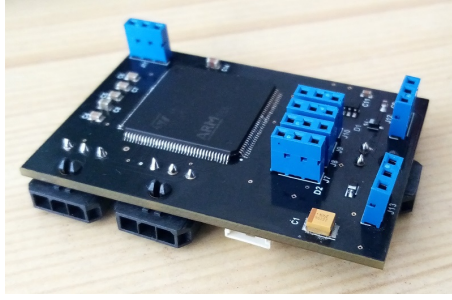
(a) PCB unmounted
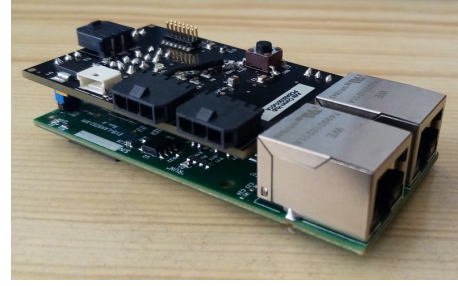
(b) PCB mounted

Figure 5.1: Manufactured PCB.

Data	Considerations	Size
Temperature	15 sensors	30 B
System	Status, events, errors and parameters data chunk	8 B
Master	Commands data chunk	8 B
IMU	Acceleration, Magnetometer and Gyroscope for 3 Axis	36 B
BISS-C	Data chunk	12 B
Total		94 B $\sim$ 128 B
Data rate @ 10 ms		12.8 kB s^{-1}=0.1 Mbit s^{-1}

Table 5.1: Data chunks considered for calculating a minimal data rate at the required refresh rate of the device.

origin, being sometimes the Host MCU or the LAN9252 chip. Nevertheless, hooking up to the SPI bus introduced another fault source that was not identified until the communication speeds and their quality were generally characterized. This is, if the GPIOs were adjusted to be visualized through a logic analyzer or oscilloscope the communication with LAN9252 was not possible. Hence, in order to keep a reliable test environment, it was decided to decrease the SPI data rate without affecting the minimal settings previously mentioned in Table 5.1. Note that that higher data rate configuration does not imply higher data transmission, as the library introduces an almost constant software delay due to the processing of the stack functions, namely between 2.7 – 2.9 µs; which starts to be significant as the interface speed goes up. The observations regarding this issue are presented in Table 5.2, where symbols imply how good or bad was the communication between the MCU host and LAN9252, and the visibility through an oscilloscope or digital analyzer. As to the noisy channels at high speeds, two screenshots can be seen in Fig. 5.2.

This overall signal integrity problem is rather common within hardware design, therefore, longer times are needed to fulfill the requirements of an optimized hardware when it comes to

Config	GPIO setup	LAN9252 comm	Signal visibility
1	PU: SS NPU/NPD: SCK/MOSI/MISO	@10 Mbit s^{-1} ⊖	⊕
2	PU: SS NPU/NPD: SCK/MOSI/MISO	@20 Mbit s^{-1} ⊖	○
3	PU: SS NPU/NPD: SCK/MOSI/MISO	@40 Mbit s^{-1} ⊖	⊖
4	NPU/NPD: SCK/SS/MOSI/MISO	@2.5 Mbit s^{-1} ⊕	○
5	NPU/NPD: SCK/SS/MOSI/MISO	@20 Mbit s^{-1} ○	⊖
6	NPU/NPD: SCK/SS/MOSI/MISO	@40 Mbit s^{-1} ○	⊖
Current	NPU/NPD: SCK/SS/MOSI/MISO	10 Mbit s^{-1} ⊕	⊖

Table 5.2: SPI GPIO configurations affected the communication performance with LAN9252. PU stands for pull-up, whereas NPD, no pull-up nor pull-down.

sensitive data signals. The mentioned working line itself is very broad and an introduction for how grounding affects the signal integrity can be consulted in [For12].

With reference to the LED results, besides the challenge that personally represented the library adaptation and the usage of the DMA peripheral, the functionality was stable.

An open point which did not represent an obstacle for the project, but keeps the door open for further improvement is the possibility of using more than one channel per PWM generator. The reason why this was not implemented lies with the time required for testing. The straightforward option is to use one DMA channel and one PWM Channel per ring, for the MCU host contains two DMA peripherals, each with 8 streams and each stream in turn with 8 channels—technical specifications can be seen in [STM19]; therefore, it did not represent any problem. Nonetheless, it could be argued that only one PWM generator could control up to four Led Rings, as long as each PWM channel is updated through an individual DMA channel, this way the hardware would be used more efficiently. The MCU should only have ready each memory buffer to start the transmission of the serial data. This approach would apparently imply larger space, but if the color data is taken out from an unique buffer that is updated on-the-fly for the next LED interface, the efficiency, at least regarding usage of HW, will be better. Consequently, this could also lead to use two DMA streams at the same time for a total of eight rings with only two PWM generators and two DMA streams.

Anyhow, the before mentioned approach needs expertise with DMA peripheral and formal evaluation of the benchmarks related to the utilization of the processor and memory footprint

(a) Non Pulled-up signal
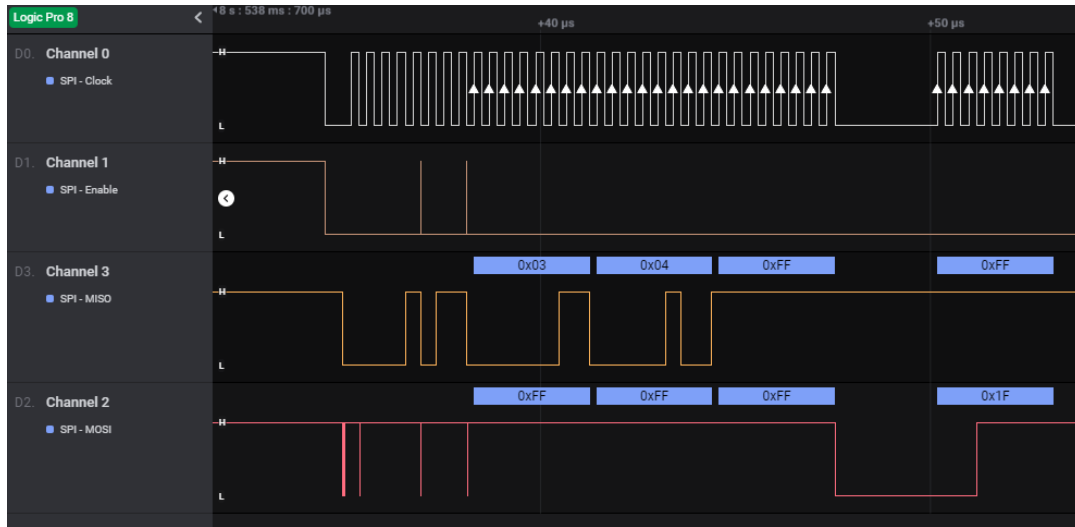
(b) Spikes at higher data rates

Figure 5.2: Issues with SPI bus. In 5.2(a) the green signal corresponds to SS pin and demands a pull-up resistor to proper visualization, while LAN9252 demands open-drain pin for correct communication. Whereas in 5.2(b) channels 1, 2 and 3 show spikes that emerged frequently at speeds higher than $5\,\mathrm{Mbit\,s^{-1}}$, leading to an incorrect recognition of the clock (CLK) signal.

Thread	Priority	Release period
User Task Manager	osPriorityHigh	Event driven
osTimer Daemon Task	osPriorityHigh	Event driven
Event Handler SDM	osPriorityAboveNormal	Event driven
SOES APP SDM	osPriorityAboveNormal	5 – 10 ms
LED SDM	osPriorityNormal	33 ms
ECAT SDM*	osPriorityNormal	100 ms
Temperature SDM	osPriorityNormal	1000 ms

Table 5.3: Final priorities' arrangement for main threads. *ECAT SDM is mainly event driven, nevertheless, in the connected state it has a periodic update*

moreover, the optimization of this LED control is not part of the scope of this project, and better use of this peripheral is apparently exploited in other applications such as graphics rendering—see application in [Weg19]—or transferring of memory chunks at way faster speeds. The latter could be though interesting, since the LAN9252 counts with a Quad-SPI interface.

5.2 DSMs and RTOS integration

As introduced during the implementation description, Sect. 4.4.1, this part will comment a bit further the results of the usage of FreeRTOS to schedule and synchronize the execution of the DSMs, such that a deterministic behavior could be reached. Similar to the previous section, some issues and comments about it are presented along the way.

5.2.1 Final scheduling and synchronization

To start with, in the Table 5.3 the current list of threads can be seen. Important to point out is the necessity to define carefully priorities besides the previously given *desired* release times. In the same table an OS-defined thread is included that was crucial for the current SW structure, namely *osTimer Daemon Task*.

Regarding the *User Task Manager*, its implementation was meaningful because of the suspension/termination of threads was taken place within the synchronization of ECAT and SOES DSMs—recall the synchronization signals in the DSM in Sect. 4.4. For instance, due to the way the library is coded, every time there is a communication breakdown, SOES goes into an infinite loop as it is polling continuously ESC state registers instead of using and IRQ pin—review SM-Synchronous in Appendix A—. Other sources of this infinite loops that demands forcing a thread to stop are the physical connection problems, such as the ESC power-off, cable disconnection, power source drop, etc. Given these scenarios, clearly a

solution with timeout control was tried out but the low level timers (HW peripherals) induced problems as it is of high caution mixing hardware interrupts when running an OS. Therefore, the Task Management events are sent with help of callback functions from osTimers (Timers implemented by the OS). As to the latter, they are really managed by the so-called osTimer Daemon Task and are fully configurable. However, a few characteristics need to be taken into account as the Daemon runs as hidden Task for the user, in addition, if this Task is not correctly set the callbacks can be lost as the Damon tries to interrupt a higher priority task. Some parameters to configure are callback stack depth, queue length and priority. More information regarding this topic can be found in *Timer Management* section of [Arm20].

Coming back to the infinite loop issue, once the timers are set and can correctly interrupt, if one task is within an infinite loop, keeping its own state machine from continue and consequently others, then a fault is emerged. Then the osTimer creates an event and suspend or restart the faulty threads. However, this is actually something that the Event Handler should take care of, or any other thread, since the OS-defined functions related to thread management cannot be executed from ISR —not even software interruptions. Furthermore, the suspension or termination of threads need to be handled with care, since if a thread is suspended or terminated right after a higher priority task preempts the callback function—imagine the Event Handler SDM being of higher priority than SOES App SDM—as soon as the higher priority task finishes, the callback function would be executed but as the OS would try to go back to the thread where it was originally called, and this has been put out of the execution queue, it would result in an OS hard fault.

Therefore, using a simple user-defined Task Manager thread that is constantly waiting for event flags or periodically updating their states, makes possible to manipulate threads, and therefore it has been the current solution to avoid some problems. However, less complicated approaches are being taken into account for further generalization, since deleting and creating tasks might have stability problems regarding the fragmentation of the heap. Sometimes, hard faults may appear and would be very hard to identify which memory overflow might have caused it.

Just as the previous example, hard faults—not surprisingly—are to be avoided, but doing this efficiently demands rather *good strategy*. What is more, this situation could repeat in other conditions, even due to implementation of the OS. For example, there are even OS-functions that are explicitly described as being compatible with execution from ISRs, though they might have no effect, as it was the case for Event Flags—even ensuring their respective volatile definitions; or in the case of waiting states within a DSM. The last-mentioned can also lead to hard fault, since the execution of an unique Task that ends up calling apparently endless osDelay functions is considered as faulty condition by the OS.

In relation to the mentioned synchronization events, the *OS Event Flags Signals* feature was used. This is basically 32-bit long data shared between threads and managed by the OS,

DSM Event	Main use
SYS_EVENT	Events and errors sent to the Event Handler DSM
LED_EVENT	Internal LED DSM events
TSENS_EVENT	Internal Temperature DSM events
ECAT_EVENT	Synchronization between ECAT DSM and SOES APP
TASKM_EVENT	Thread management/update request from any DSM

Table 5.4: Events between DSMs are implemented over *Event Flag Signals* managed by the OS.

ACB Event	ACB Error
EV_TEMP_DSM_INIT	ERR_TEMP_DSM_FAULT
EV_LED_DSM_INIT	ERR_TEMP_SENS_OVERHEAT
EV_ECAT_DSM_INIT	ERR_TEMP_SENS_LOST*
EV_ECAT_CMD_ACK	ERR_LED_DSM_FAULT
EV_ECAT_APP_OP	ERR_ECAT_DSM_FAULT
EV_ECAT_APP_INIT	ERR_ECAT_CMD_FAULT*
EV_ECAT_CMD_ACK	ERR_ECAT_CMD_SOFTFAULT*
	ERR_ECAT_COMM_LOST
	ERR_SYS_UNKNOWN

Table 5.5: DSM's events and errors considered by Event Handler DSM. *Currently being implemented.*

that can be related to suspension and resume of threads according to the bits (flags) that are set or clear with *OS Wait* functions. Some details could be not as clear as they should be, since working conditions vary among OS layers, this is for example, the FreeRTOS defines the signals as 32-bit unsigned integer with the 31st bit reserved for internal error flag; nonetheless, on top of it the CMSIS v2 defines in turn the same signal as the same integer but only 24 bits available. If the application works with any flag bit between the 24th and the 30th bit, the final OS layer returns no error but does not notify any thread. A list of the main signals used for synchronization between DSM is presented in Table 5.4, whereas a list of the events and errors so far implemented in the Event Handler DSM's logic is showed in Table 5.5. However, a complete list of events and errors can be seen in the header code within Appendix E.

5.2.2 Further development

Finally, there are two topics open for future, first one has to do with shared memory, while the second is about optimization of the scheduling approach. Regarding the former, so far no complication has appeared in the rather non-critical data being transmitted, namely the buffer that both the Temperature and the SOES App update. Currently, an approach an easy

approach that can be implemented is using again the events to notify the SOES APP whenever the Temperature App has updated its buffer. Other is using the MUTEX feature implemented in the OS. Both approaches are planned. Going back to the mentioned topics, the latter could be at this stage difficult, as it needs the precise execution time of each Task to be known. Without it is not possible to think about optimizing the utilization or the OS configuration in a more detailed manner -if even required-. For instance, in the future calculation of the *Utilization* factor will be helpful for optimizing scheduling, but it could be also needed in more practical design cases, e.g., while considering heat sinks for processors within enclosed devices. Therefore, the following is a list of proposed activities that could take place in future stages out of scope of the current Research Project.

Execution Time estimation per task Each task can be isolated by software. Then, by adding a piece of code to toggle a free GPIO at the end of the thread, a signal can be traced with a fair digital analyzer. Omitting the rather small HAL overhead added with the GPIO control, an estimation of the execution time can be achieved.

Live thread tracing A trace debugging like SEGGER SystemView, see [SEG20],can be used to debug freeRTOS applications running on ARM Cortex Mx based Microcontroller such as STM32Fx. With this tool it could be possible to have at runtime a trace of the thread allocations, knowing in consequence the duration of the threads.

Optimization of threads By knowing the Worst Case Execution Time (WCET) of each thread an optimization of the utilization could be carried out by using different OS-native features to improve the scheduling, as long as the application demands it.

5.3 SOES library integration

The results presented in this section implies the correct functionality of the temperature functional block. In the Fig. 5.3 the transition from safe-operation (SAFEOP) to operation state (OP) of the EtherCAT device is shown. The operation state can only be reached when the Object Dictionary has been correctly matched between the Master and the device through the SMs, a synchronization mode has been correctly configured and the AL registers can be correctly read and written by both devices. The protocol makes use of DLL commands to broadcast state change requests to the slaves, afterwards it broadcasts a read command. Each device connected will react and update its corresponding AL state register.

In addition, it is important to recall that the working counter at the end of the data structure that points out whether a device in the network has processed the data frame or not, review the frame structure in Appendix A. Having the frame in mind, it is possible to recognize all

No.	Time	Source	Destination	Protocol	Length	Info
514	1.841079	Beckhoff_01:00:00	02:00:00:00:00:00	ECAT	93	3 Cmds, 'LRD': len 1, 'LRW': len 38, 'BRD': len 2
515	1.841112	Beckhoff_01:00:00	02:00:00:00:00:00	ECAT	60	'FPWR': Len: 2, Adp 0x3e9, Ado 0x120, Wc 1
516	1.841120	00:00:00_00:00:00	Beckhoff_01:00:00	ECAT	93	3 Cmds, 'LRD': len 1, 'LRW': len 38, 'BRD': len 2
517	1.841130	00:00:00_00:00:00	Beckhoff_01:00:00	ECAT	36	'FPRD': Len: 8, Adp 0x3e9, Ado 0x300, Wc 0
518	1.850239	Beckhoff_01:00:00	02:00:00:00:00:00	ECAT	93	3 Cmds, 'LRD': len 1, 'LRW': len 38, 'BRD': len 2
519	1.850266	Beckhoff_01:00:00	02:00:00:00:00:00	ECAT	60	'FPRD': Len: 8, Adp 0x3e9, Ado 0x300, Wc 1
520	1.850276	00:00:00_00:00:00	Beckhoff_01:00:00	ECAT	93	3 Cmds, 'LRD': len 1, 'LRW': len 38, 'BRD': len 2

Figure 5.3: Transaction of data frames between Master and Slave during a state change request.

its parts, as the the communication trace was monitored with Wireshark. Hence the detailed interaction between parties through the updated data frames is clearly seen in Fig. 5.4.

```
∨ EtherCAT datagram: Cmd: 'BRD' (7), Len: 2, Adp 0x1, Ado 0x130, Cnt 1
    ∨ Header
        Cmd          : 7 (Broadcast Read)
        Index: 0x00
        Slave Addr: 0x0001
        Offset Addr: 0x0130
      > Length       : 2 (0x2) - No Roundtrip - Last Sub Command
        Interrupt: 0x0000
    > AL Status (0x130): 0x0004, Al Status: SAFEOP
      Working Cnt: 1
```

(a) BRD command to read current AL state register

```
∨ EtherCAT datagram(s): 'FPWR': Len: 2, Adp 0x3e9, Ado 0x120, Wc 1
    ∨ EtherCAT datagram: Cmd: 'FPWR' (5), Len: 2, Adp 0x3e9, Ado 0x120, Cnt 1
        ∨ Header
            Cmd          : 5 (Configured address Physical Write)
            Index: 0x8f
            Slave Addr: 0x03e9
            Offset Addr: 0x0120
          > Length       : 2 (0x2) - No Roundtrip - Last Sub Command
            Interrupt: 0x0000
        > AL Ctrl (0x120): 0x0008, Al Ctrl: OP
          Working Cnt: 1
    Pad bytes: 00000000000000000000000000000000000000000000000000…
```

(b) FPWR command to write a request to AL control register

```
∨ EtherCAT datagram: Cmd: 'BRD' (7), Len: 2, Adp 0x1, Ado 0x130, Cnt 1
    ∨ Header
        Cmd          : 7 (Broadcast Read)
        Index: 0x00
        Slave Addr: 0x0001
        Offset Addr: 0x0130
      > Length       : 2 (0x2) - No Roundtrip - Last Sub Command
        Interrupt: 0x0000
    > AL Status (0x130): 0x0008, Al Status: OP
      Working Cnt: 1
```

(c) BRD command to read current AL state register

Figure 5.4: Detailed request and response for a state change. SAFEOP to OP state is the last change starting from INIT state.

Finally, in Fig. 5.5 the input and outputs part of the object dictionary is shown after it can be accessed through TwinCAT3. A whole definition of it is though included in the Appendix D.

6000:0	Temperature		> 15 <
6000:01	TEMPERATURE0	RO P	0x0000 (0)
6000:02	TEMPERATURE1	RO P	0x0000 (0)
6000:03	TEMPERATURE2	RO P	0x0000 (0)
6000:04	TEMPERATURE3	RO P	0x0000 (0)
6000:05	TEMPERATURE4	RO P	0x0000 (0)
6000:06	TEMPERATURE5	RO P	0x0000 (0)
6000:07	TEMPERATURE6	RO P	0x0000 (0)
6000:08	TEMPERATURE7	RO P	0x0000 (0)
6000:09	TEMPERATURE8	RO P	0x0000 (0)
6000:0A	TEMPERATURE9	RO P	0x0000 (0)
6000:0B	TEMPERATURE10	RO P	0x0000 (0)
6000:0C	TEMPERATURE11	RO P	0x0000 (0)
6000:0D	TEMPERATURE12	RO P	0x0000 (0)
6000:0E	TEMPERATURE13	RO P	0x0000 (0)
6000:0F	TEMPERATURE14	RO P	0x0000 (0)

(a) Process data input

6001:0	SystemStatus		> 3 <
6001:01	STATUS	RO P	0x0000 (0)
6001:02	EVENT	RO P	0
6001:03	ERROR	RO P	0x0000 (0)
7000:0	MasterCMDs		> 4 <
7000:01	COMMAND	RW P	0x0000 (0)
7000:02	TEST VALUE0	RW P	0x0000 (0)
7000:03	TEST VALUE1	RW P	0x0000 (0)
7000:04	TEST VALUE2	RW P	0x0000 (0)
F000:0	Modular Device Profile		> 2 <

(b) Process data output

Figure 5.5: Object Dictionary from TwinCAT interface.

Conclusions and further development

Through this document a very helpful introduction to the industrial communication protocols with real time capabilities, their standards and current applications has been provided. Highlighting as well its importance in current applications, not only at the industrial field but in academic research, mainly in the field of robotics. An overview of some of current projects that are being developed has been commented as well, making clear how important the relation between open source software and open protocols is, mainly for any company or academic group that wishes to develop devices or improve current technologies. As the Industrial Internet of Things grows continuously, so the frameworks and projects do; this is the case for the TSN initiative that looks for homogenizing the rather populated industrial protocols and their messy compatibility due to their historic development—full of commercial interests that eventually lead to protocols similar in features and still not compatible between each other. Before the TSN initiative, during this document it has been frequently commented that the EtherCAT protocol is making its way by holding the open and the multi-platform strategy, this of course has changed through the years, but currently the development group is fully integrated with the TSN initiative. The mentioned protocol's multi-platform approach and the specific necessity of developing interfaces—embedded devices—compatible with industrial robots, lead to this project.

The project itself, as detailed throughout the document, covers hardware and software development of a prototype that marks the starting point for a flexible embedded system. Important to mention is the usage of open libraries and their modification and synchronization of their functionalities through the implementation of State Machines and FreeRTOS. The latter, besides the overall EtherCAT software stack integration, is of great importance since both are tools that allow the developer to build up firmware in a modular and structural way, such that it provides a platform for further development, with several advantages, such as flexibility for integration of new features, multi-task approach, deterministic behavior through event handler and the time management functions of the OS, capabilities for periodic or

event-driven tasks, experience in management of libraries, hardware and industrial protocols, among others.

The development helped open the doors to new challenges regarding characterization of the firmware performance and capabilities assessment. Furthermore, this device represents development potential, for it could expand functionalities and test compatibility under specific standards, for instance—as the device achieved its goal of establishing a stable connection with a EtherCAT master—features like FoE service for firmware update over the field bus, DC synchronization for hard-real time capabilities and EoE for Ethernet/IP based cloud access can be planned for the future. It could be even candidate for an in-depth dependability analysis, an EtherCAT compliance certification or even FSoE for black channel integration; the previous depending of course on the application, since each of those features would imply different paths that—even when they're obviously related—the development cannot be mixed neither done at once. Regarding hardware, optimizing the design for integrity of multiple digital signals—asynchronous and synchronous between $5 - 90\,\mathrm{Mbit\,s^{-1}}$—is also an important point.

As to the software for embedded systems, new abstraction layers for RTOS can be explored, since the code uses CMSIS that theoretically provides flexibility to change the underlying RTOS to different ones. The latter would make only sense if, one, the path to follow is to exploit EoE, there are then unikernel projects that basically are tiny linux kernels running on low-cost MPUs; or, two, to explore the safety related functionalities, for which multi-core platforms are appropriate to redundancy approaches—with their related multi-core RTOS. Additionally, also in the sense of embedded systems, more complex architectures can be explored, not because the STM32F4xx MCU is not capable enough, but Texas Instruments MCUs emerged during this project as an interesting alternative, as they sometimes offer more features aiming multi-protocol peripherals, even for those that are currently and commonly integrated through ASICs like IO-LINK or BISS-C interfaces. Finally, it could be summarized that the Research Project achieved the main goal of developing a flexible device for further industrial interfaces.

Bibliography

[Arm20]   Arm Ltd. Real-Time Operating System: API and RTX Reference Implementation. https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html, April 2020. Last visited: 20/09/2020.

[BB17]   U. Becker and M. Becker. Mikrocontroller. https://mikrocontroller.bplaced.net/wordpress/, January 2017. Last visited: 20/09/2020.

[BDL06]   G. Behrmann, A. David, and K. Larsen. A tutorial on uppaal 4.0. *Department of computer science, Aalborg university*, 2006.

[Bec13]   Beckhoff Automation GmbH. Urban Automation PC-based Control for Urban Environments. https://m.beckhoff.com/english/applicat/urban.htm, April 2013. Last visited: 16/09/2020.

[Bec15]   Beckhoff Automation GmbH. ETG.1020 S (R) V1.2.0: Protocol Enhancements. https://www.beckhoff.com/english/downloadfinder, 2015. Last visited: 24/09/2020.

[Bec17a] Beckhoff Automation GmbH. ETG.1000.1 S (R) V1.0.4: Overview. https://www.beckhoff.com/english/downloadfinder, September 2017. Last visited: 24/09/2020.

[Bec17b] Beckhoff Automation GmbH. ETG.1000.4 Data Link Layer protocol specification. https://www.beckhoff.com/english/downloadfinder, November 2017. Last visited: 16/09/2020.

[Bec17c] Beckhoff Automation GmbH. ETG.1000.6 Application Layer protocol specification. https://www.beckhoff.com/english/downloadfinder, November 2017. Last visited: 16/09/2020.

[Bec17d] Beckhoff Automation GmbH. EtherCAT Slave Controller: Technology. https://www.beckhoff.com/english/downloadfinder, February 2017. Last visited: 16/09/2020.

[Bec18]   Beckhoff Automation GmbH. ETG.2200 V3.1.0. https://www.beckhoff.com/english/downloadfinder, November 2018. Last visited: 16/09/2020.

[Bec20a] Beckhoff Automation GmbH. ETG.2000 S (R) V1.0.12: ESI Specification. https://www.beckhoff.com/english/downloadfinder, May 2020. Last visited: 24/09/2020.

[Bec20b] Beckhoff Automation GmbH. EtherCAT - the Ethernet Fieldbus. https://www.ethercat.org/en/technology.html, January 2020. Last visited: 16/09/2020.

[Bec20c] Beckhoff Automation GmbH. EtherCAT Slave Controller Overview. https://www.ethercat.org/en/downloads.html, January 2020. Last visited: 17/09/2020.

[Bec20d] Beckhoff Automation GmbH. TwinCAT/BSD: operating system for Industrial PCs. https://www.beckhoff.com/english.asp?highlights/twincat-bsd/default.htm, June 2020. Last visited: 16/09/2020.

Bibliography

[Bor08]   D. Borchers. Open Source trifft die Industrie. https://www.heise.de/newsticker/meldung/Open-Source-trifft-die-Industrie-201855.html, April 2008. Last visited: 16/09/2020.

[Car20]   T. Carlsson. Industrial network market shares 2020 according to HMS Networks. https://www.anybus.com/about-us/news/2020/05/29/industrial-network-market-shares-2020-according-to-hms-networks, May 2020. Last visited: 11/09/2020.

[DC17]   R. Delgado and B. Choi. Real-time servo control using ethercat master on real-time embedded linux extensions. *International Journal of Applied Engineering Research*, 12:1179–1185, 11 2017.

[Des00]   Design News. SERCOS picks up the pace. https://www.designnews.com/sercos-picks-pace, August 2000. Last visited: 11/09/2020.

[emb20]   emb4fun. emb4fun's OpENer patch homepage. https://www.emb4fun.de/archive/chibiosopener/index.html, January 2020. Last visited: 11/09/2020.

[Fel02]   M. Felser. The fieldbus standards: History and structures. 2002.

[For12]   M. Fortunato. SUCCESSFUL PCB GROUNDING WITH MIXED-SIGNAL CHIPS - FOLLOW THE PATH OF LEAST IMPEDANCE. https://www.maximintegrated.com/en/design/technical-documents/tutorials/5/5450.html, October 2012. Last visited: 17/09/2020.

[Gro04]   G. Gross. Infineon to pay 160M fine for DRAM price-fixing. https://www.computerworld.com/article/2566841/infineon-to-pay–160m-fine-for-dram-price-fixing.html, September 2004. Last visited: 17/09/2020.

[HWHP]   W. Hagemeister, Weber, R. Hacker, and F. Pose.

[IEC20]   IEC. OPC unified architecture - Part 14: PubSub. https://webstore.iec.ch/publication/61108, July 2020. Last visited: 11/09/2020.

[IEE20]   IEEE. Time-Sensitive Networking (TSN) Task Group. https://1.ieee802.org/tsn/, January 2020. Last visited: 11/09/2020.

[II20]   IEEE and IEC. IEC/IEEE 60802 TSN Profile for Industrial Automation. https://1.ieee802.org/tsn/iec-ieee-60802/, January 2020. Last visited: 11/09/2020.

[III17]   ISO, IEC, and IEEE. 8802-3:2017 Part 3: Standard for Ethernet. https://www.iso.org/standard/72048.html, 2017. Last visited: 11/09/2020.

[Inf18]   Infineon Technologies AG. XMC4700 / XMC4800 Datasheet. https://www.infineon.com/cms/en/product/, September 2018. Last visited: 17/09/2020.

[JDH02]   B. Johnson, D. Dunn, and R. Hulett. A comparison of the ieee and iec standards processes. In *Record of Conference Papers. Industry Applications Society. Forty-Ninth Annual Conference. 2002 Petroleum and Chemical Industry Technical Conference*, pages 1–12, 2002.

[JYJP20]   Sanghoon Ji, Donguk Yu, Hoseok Jung, and Hong Seong Park. Real-time robot software platform for industrial application. In Antoni Grau and Zhuping Wang, editors, *Industrial Robotics*, chapter 6. IntechOpen, Rijeka, 2020.

[Maj20]   T. Majerle. LwOW Lightweight, platform independent library to interface 1-Wire devices over UART protocol. https://majerle.eu/index.php/projects/lwow-one-wire-uart-communication, January 2020. Last visited: 20/09/2020.

[Max02]   Maxim Integrated Products. USING A UART TO IMPLEMENT A 1-WIRE BUS MASTER. https://www.maximintegrated.com/en/design/technical-documents/tutorials/2/214.html, September 2002. Last visited: 20/09/2020.

[MGSR20]   M. Metaal, R. Guillaume, R. Steinmetz, and A. Rizk. Integrated industrial ethernet networks: Time-sensitive networking over sdn infrastructure for mixed applications. In *2020 IFIP Networking Conference (Networking)*, pages 803–808, 2020.

[MHF+20]   C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini. Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping, and whole-body control. *IEEE Transactions on Robotics*, pages 1–14, 2020.

[Mic15]   Microchip Technology Inc. DS00001909A: LAN9252 Datasheet. https://www.microchip.com/wwwproducts/en/LAN9252, January 2015. Last visited: 22/09/2020.

[Mik20]   Mikrocontroller Artikelsammlung. WS2812 Ansteuerung. https://www.mikrocontroller.net/articles/WS2812_Ansteuerung, January 2020. Last visited: 20/09/2020.

[MLH+17]   L. Muratore, A. Laurenzi, E. Hoffman, A. Rocchi, D. Caldwell, and N. Tsagarakis. Xbotcore: A real-time cross-robot software platform. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 77–80, 2017.

[MLT18]   L. Muratore, B. Lennox, and N. Tsagarakis. Xbotcloud: A scalable cloud computing infrastructure for xbot powered robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[Nor20]   M. Norris. A Cannabis Equipment Supplier Turns to Automation to Extract CBD at Commercial Scale. https://www.automationworld.com/home/article/21173096/a-cannabis-equipment-supplier-turns-to-automation-to-extract-cbd-at-commercial-scale, August 2020. Last visited: 14/09/2020.

[OpE20]   OpENer community. OpENer - Open Source EtherNet/IP(TM) Communication Stack. http://eipstackgroup.github.io/OpENer/index.html, January 2020. Last visited: 11/09/2020.

[OSA19]   OSADL. Building an Open Source OPC UA/TSN Ecosystem. https://www.osadl.org/uploads/media/OSADL-OPC-UA-TSN-Open-Source-Ecosystem-LoI-2nd-edition-V6.pdf, January 2019. Last visited: 11/09/2020.

[PERK18]   Pfrommer, Ebner, Ravikumar, and Karunakaran. Open source opc ua pubsub over tsn for realtime industrial communication. 07 2018.

[RT-20]   RT-LABS. EtherCAT Software Stacks. https://rt-labs.com/products/ethercat/, January 2020. Last visited: 11/09/2020.

[SEG20]   SEGGER Microcontroller GmbH. What is SystemView? https://www.segger.com/products/development-tools/systemview/technology/what-is-systemview/, April 2020. Last visited: 20/09/2020.

[Ser18]   Sercos International eV. Ethernet TSN heralds a new era of industrial communication. https://www.sercos.org/technology/sercos-in-connection-with-tsn-opc-ua/sercos-in-connection-with-tsn/, January 2018. Last visited: 11/09/2020.

[Ser20a]   Sercos International eV. Driver Software. https://www.sercos.org/technology/implementation/driver-software, January 2020. Last visited: 11/09/2020.

[Ser20b]   Sercos International eV. SERCOS performance. https://www.sercos.org/technology/advantages-of-sercos/performance/, January 2020. Last visited: 11/09/2020.

Bibliography

[SKJ18] S. Schriegel, T. Kobzan, and J. Jasperneite. Investigation on a distributed sdn control plane architecture for heterogeneous time sensitive networks. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–10, 2018.

[Son20] M. Soni. EtherCAT on Sitara Processors. Technical report, Arm Microprocessor Group Texas Instruments, April 2020. Last visited: 17/09/2020.

[STM17] STMicroelectronics NV. UM1725: Description of STM32F4 HAL and LL drivers. https://www.st.com/en/, February 2017. Last visited: 22/09/2020.

[STM19] STMicroelectronics NV. DS10693 Rev 7 STM32F446xC/E. https://www.st.com/en/microcontrollers-microprocessors/stm32f446.html, October 2019. Last visited: 20/09/2020.

[VZS19] S. Vitturi, C. Zunino, and T. Sauter. Industrial communication systems and their future challenges: Next-generation ethernet, iiot, and 5g. *Proceedings of the IEEE*, 107(6):944–961, 2019.

[Weg19] J. Wegrzyn. OPTIMIZING RENDERING PERFORMANCE ON STM32 USING DMA2D. https://auto.siili.com/blog/optimizing-rendering-performance-on-stm32-using-dma2d, March 2019. Last visited: 20/09/2020.

[XJY11] H. Xing, H. Jia, and L. Yanqianga. Motion control system using sercos over ethercat. *Procedia Engineering*, 24:749–753, 12 2011.

Introduction to EtherCAT

As mentioned in the introduction, EtherCAT is an industrial Communication Profile developed by Beckhoff Automation GmbH that is standardized in the IEC 61588 under the RTE CPF. The development within this company is oriented to the use of open standards to increase its impact within the industry, but not only reduced to it but the overall smart cities field, in a certain degree this approach eliminates the need for many expensive *black boxes* as mentioned in [Bec13]. This implies that the interoperability of devices is almost guaranteed, at least from the specification perspective, not only for private development centers but also for any other developer that follow the standards; if the standards are of public access, then this is a mean of empowerment of any group that might be willing to create its own industrial-compatible technologies.

The OSADL emphasized in 2008, see article in [Bor08], for example, a vision for leading the integration of open source in the industry by using the Linux Kernel as a certified Industrial RT (IRT) operative system for industrial embedded applications. Back in that day, Beckhoff Automation was involved in that discussion representing the contrary model. Nonetheless, in the last months the same company has apparently retaken the open source initiative by the introduction of the FreeBSD compatible version of the TwinCAT Runtime—visit this newsletter [Bec20d].

In comparison with other RTE profiles, EtherCAT has shown a higher performance, more flexible topology and lower costs than other field bus technologies. This protocol applies a master-slave mode, in which the master device uses standard 100BASE-TX Ethernet adapter and the EtherCAT Slave Controller (ESC), that implements an EtherCAT IP (intellectual property) core within an ASIC or an FPGA to process the frames. As the working cycle starts, the Master publishes a frame encapsulated a standardized 8802.3 frame. When it reaches an ESC, it analyses the address and location on the frame, decides which parts of it are useful sections and then reads or writes data on it. As the read-write operation finishes, the Working Counter (WKC) at the end of the frame is added by one, this way the data on the frame has been processed. This cycle repeats for each ESC within the topology.

EtherCAT supports almost all kinds of topology structure, such as ring, line, star and tree. The transmission speed of the interface is fixed to $100\,\mathrm{Mbit\,s^{-1}}$ with full duplex communication. The network is able to connect maximally 65535 devices via switch and media converter. The EtherCAT system can update 1000 I/Os in just $30\,\mu s$ or exchange $1486\,B$ contents in $300\,\mu s$. The previous technical data can be reviewed in [Bec17d] and an overview of the protocol in [XJY11].

Important to highlight is that other CPs are also integrated as services inside the protocol, as mentioned before for the SERCOS specification in Sect. 2.1. Other examples of these integrated CPs are File over EtherCAT (FoE) or Ethernet over EtherCAT (EoE), which make possible support a wide variety of devices and application layers in the same network. A complete list of the communication profiles that are on hand through the protocol's mailbox is given below, as to the overall layered integration can be seen in Fig. A.1.

- CoE: CAN application protocol over EtherCAT

- SoE: Servo drive profile, according to IEC 61800-7-204 (SERCOS protocol)

- EoE: Ethernet over EtherCAT

- FoE: File Access over EtherCAT (HTTP,FTP,etc)

- AoE: Automation Device Protocol over EtherCAT (ADS over EtherCAT)
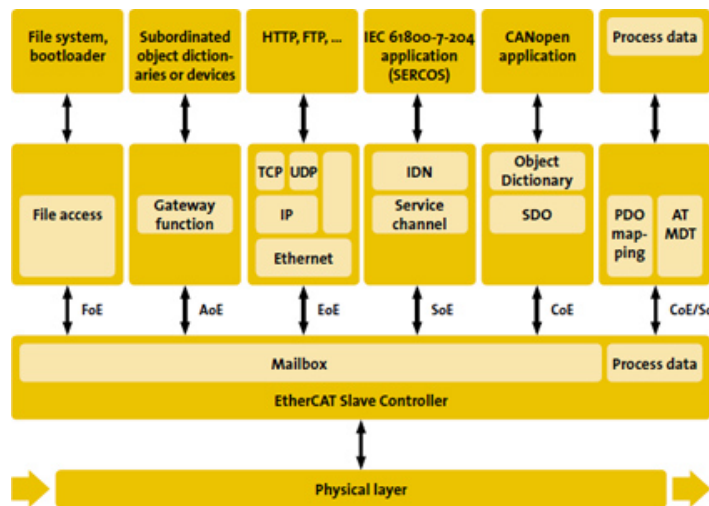
Figure A.1: Different communication profiles can coexist in the same system. Source from [Bec20b].

An EtherCAT device with switch port properties using EoE would be the equivalent of the TSN compliant switches, since they would insert any non time-sensitive TCP/IP fragment into the RTE traffic preventing in this way the real time properties from being affected.

Furthermore, the architecture of the protocol itself and its early cooperation with the IEEE 802.1 group and the OPC Group ensure its continuous compatibility with the standardization of TSN, OPC UA and the IoT paradigm. See the following article about compatibility [Bec20b].
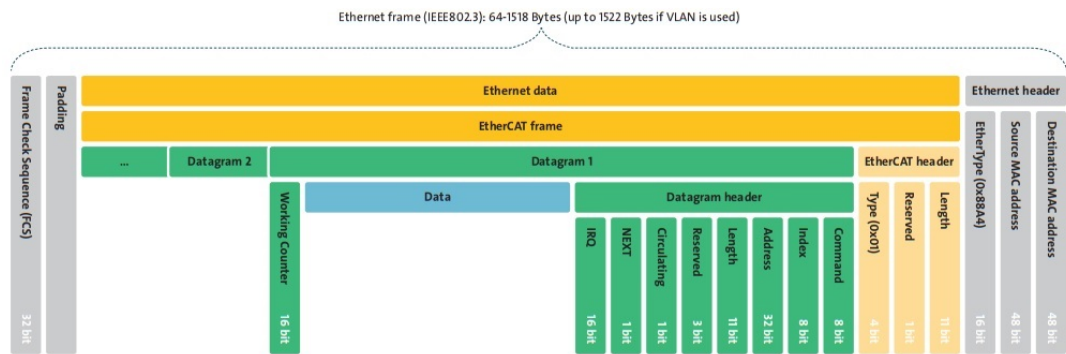
Figure A.2: EtherCAT datagram within the Ethernet frame. Source from [Bec17b].

The data frame and the Synchronization Managers

Besides the challenge of setting up the hardware and basic firmware for a correct data transmission between ESC and the host MCU; the description of the EtherCAT Slave device is a task that demands, at least, a basic understanding of the data frame exchange and how the protocol demands its synchronization. From here on, the following topics are going to be summarized: Synchronization modes and managers. Whenever there are Real Time constraints, and the device takes part of a control loop, synchronization modes are needed to be set correctly between the Master and any Device present. For this task the Distributed Clocks (DC) are need to be synchronized. See the section 20 in [Bec15].

There are three synchronization modes:

Free Run Application is triggered by local clock and runs independently of EtherCAT cycle.

SM-Synchronous Application is synchronized whenever there are process data being written to the Synchronization Manager 2 (SM2). Moreover, any event generated by the Master is mapped onto an internal register or physically triggering an IRQ Pin of the ESC.

DC-Synchronous Within this synchronization mode the frame jitter can be even reduced down to ns and use two different synchronization units within the ESC, namely the SM2 and SYNC/LATCH UNIT.

Synchronization Managers 1,2,3—as many as the hardware includes and sometimes referred as SMXs—coordinate access to the ESC memory from both sides, Master and Host MCU through the Process Data Interface (PDI). In case of process data communication it ensures that

(a) Free Run   (b) SM-Synchronization   (c) DC-Synchronization modes
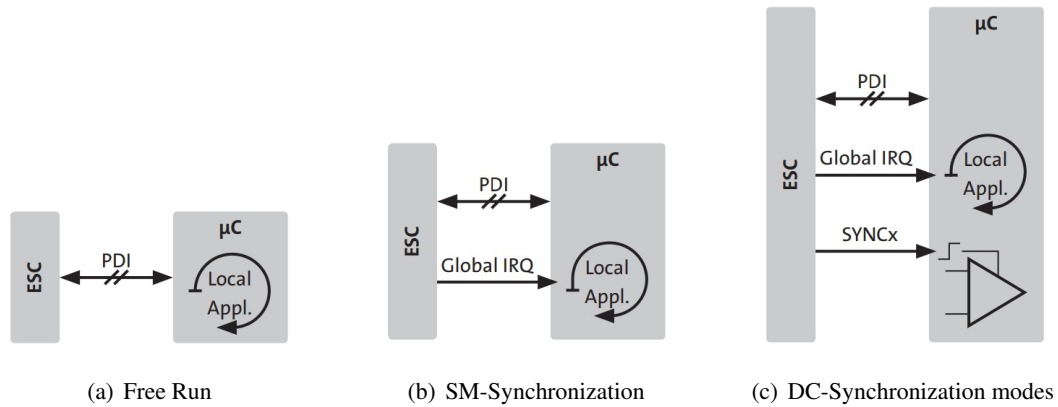
Figure A.3: Synchronization modes. Definitions can be reviewed in depth within [Bec17a].

process data objects (PDO) can always be written to the memory by EtherCAT and can always be read in the PDI side and vice versa (3-buffer mode). SyncManager 2/3 length is equal to the Data Object lengths defined for receive and transmit data chunks respectively. SyncManagers are explained in detail in [Bec17b]. The mapping of the process data objects within the Ethernet Frame can be seen in figure A.2 and A.4. The correct setup of the SMXs ensure the consistency of the data and needs to be linked correctly depending on the characteristics of each ESC type, and SW Stack that are being used, this information is also linked to the CoE Object Dictionary (OD) and EtherCAT Slave Information (ESI) file.
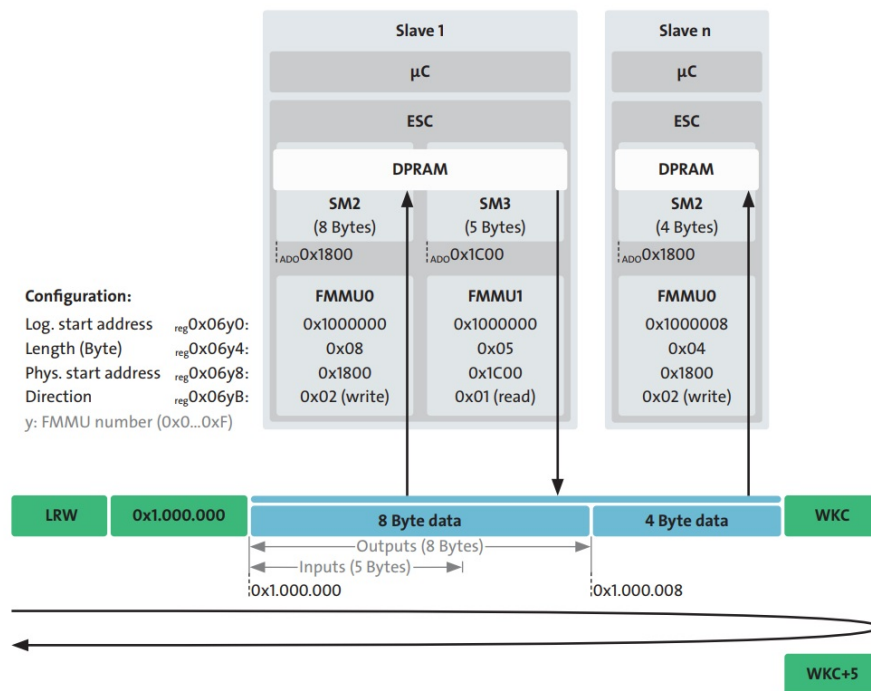
Figure A.4: Depending on the different states of the Slave, there will be different data frames being exchanged with the Master. The above one corresponds to the PDO which is updated continuously by the SM2/3 during Operation State (OP). Resource from SDO section in [Bec17c].

Device State Machines (DSMs)
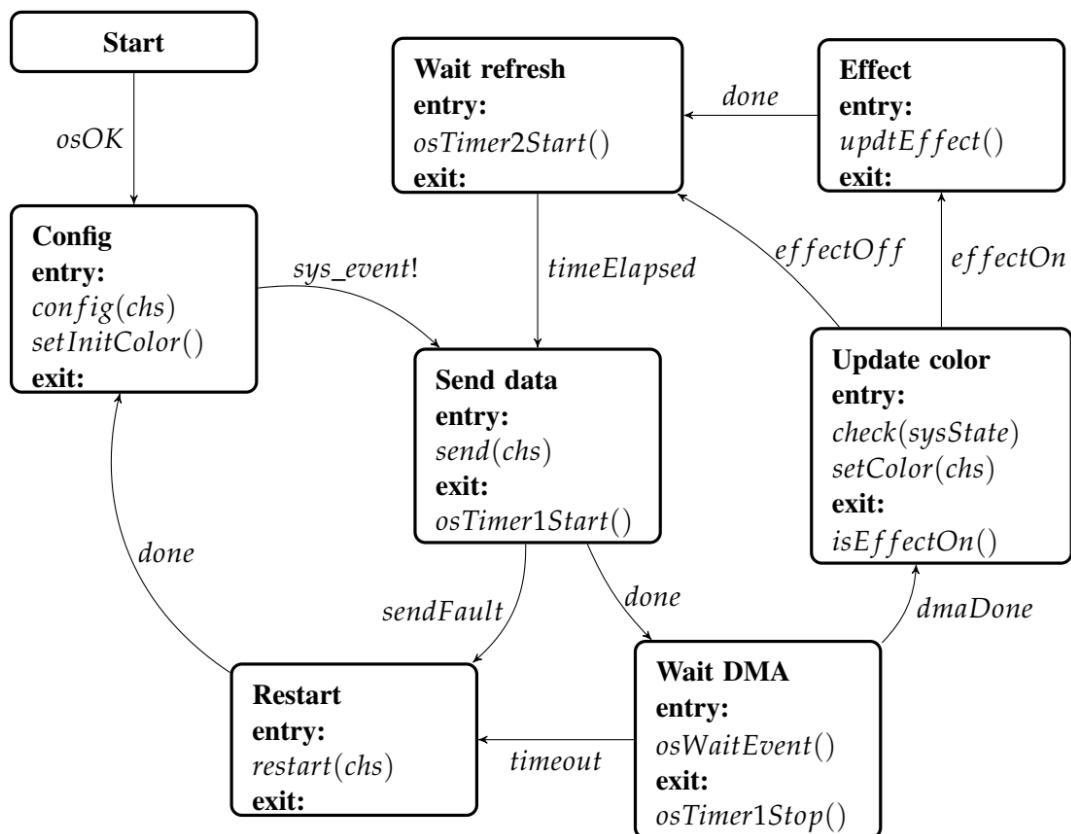
Figure B.1: DSM for LED control functionality.
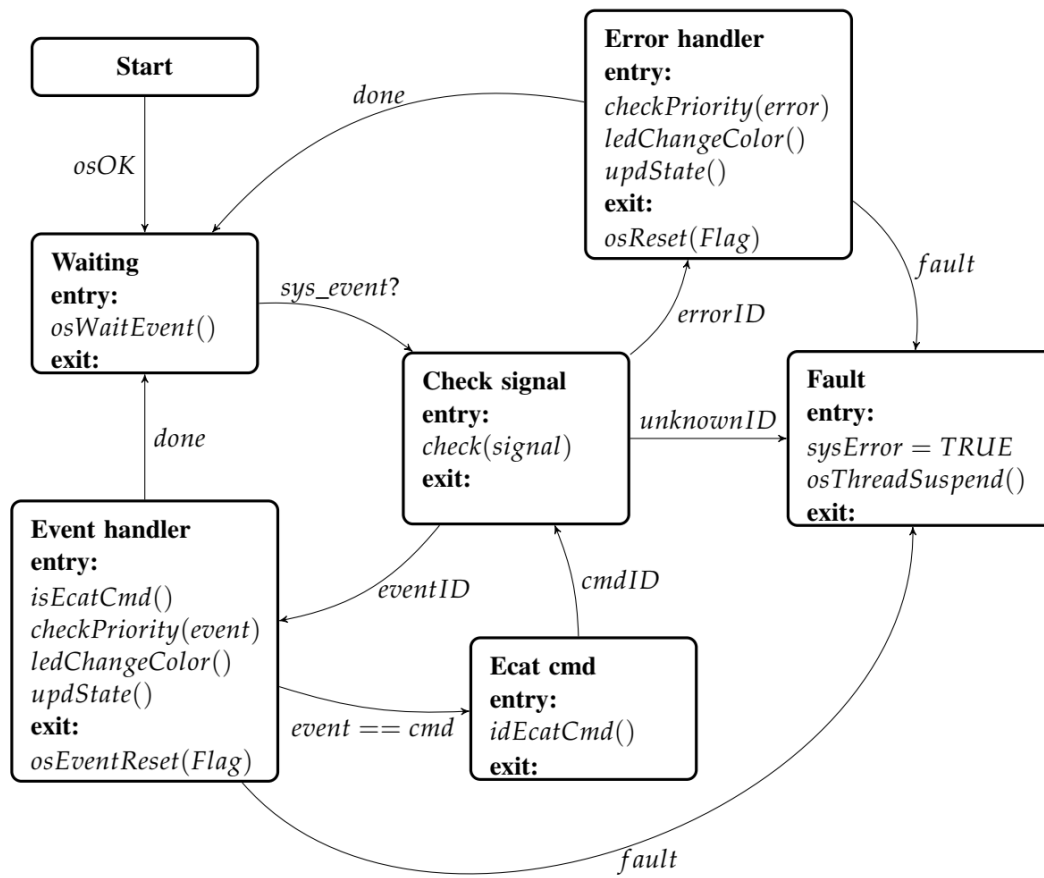
Figure B.2: DSM for Event handler functionality.
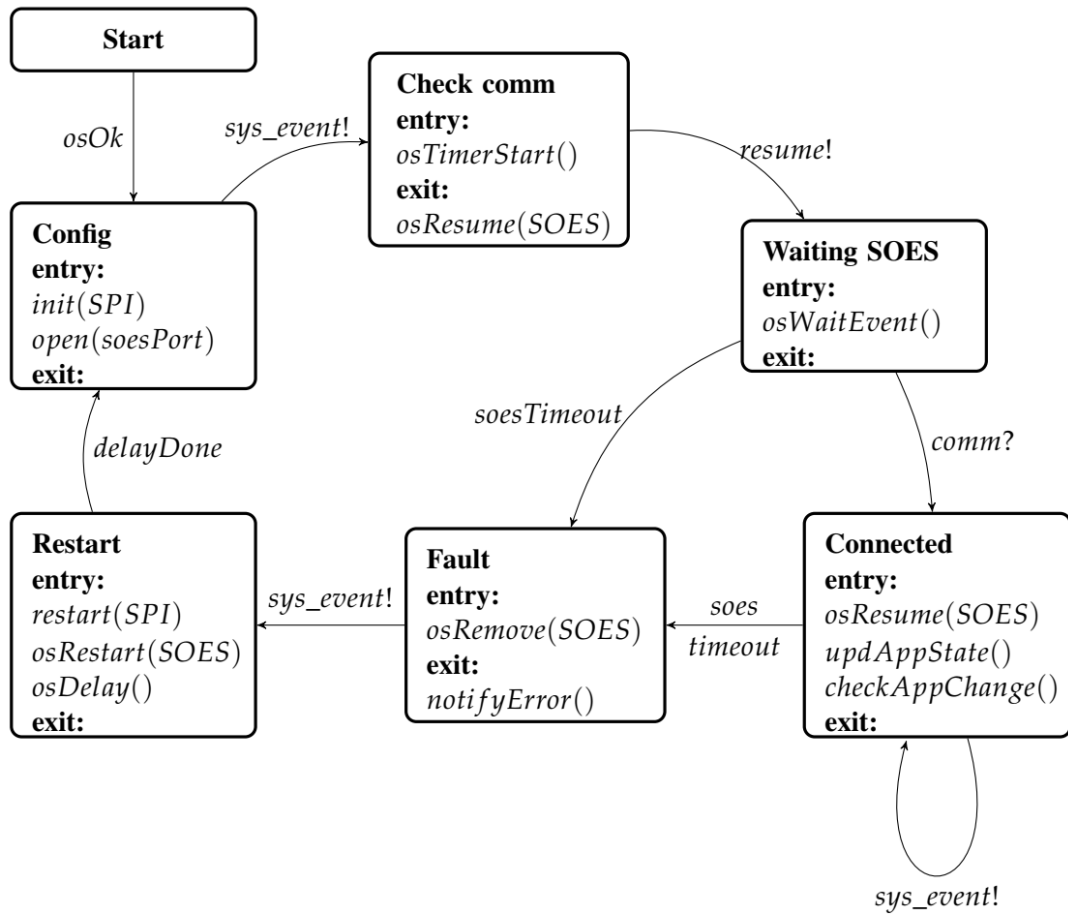
(a) Synchronization state machine.
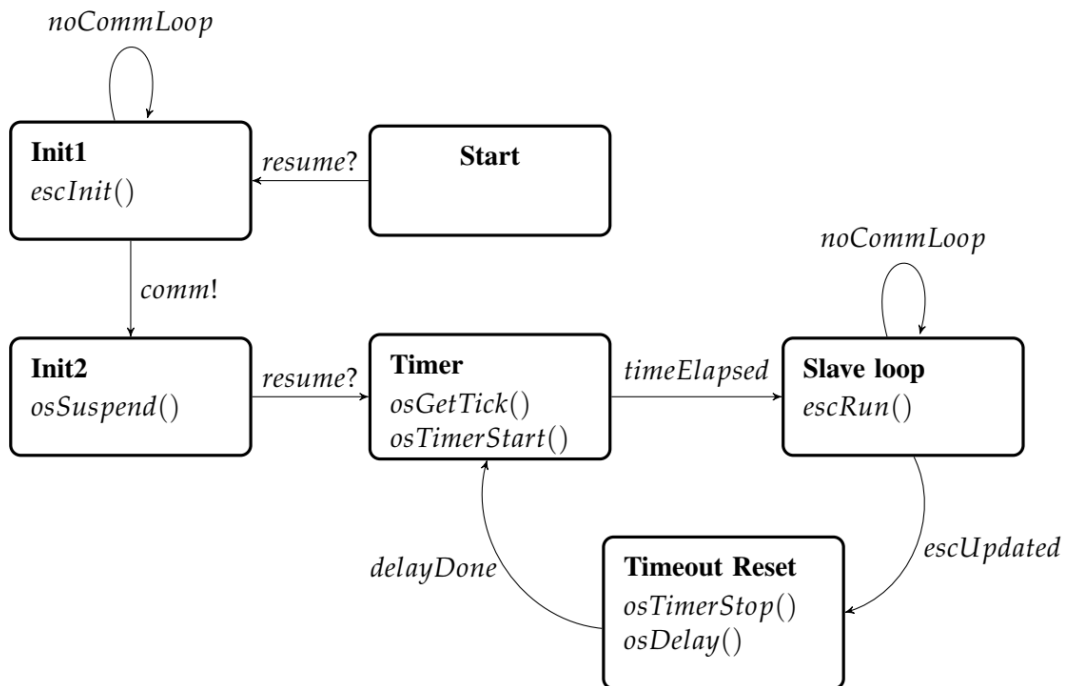
(b) SOES application state machine.

Figure B.3: DSM for EtherCAT functionality.
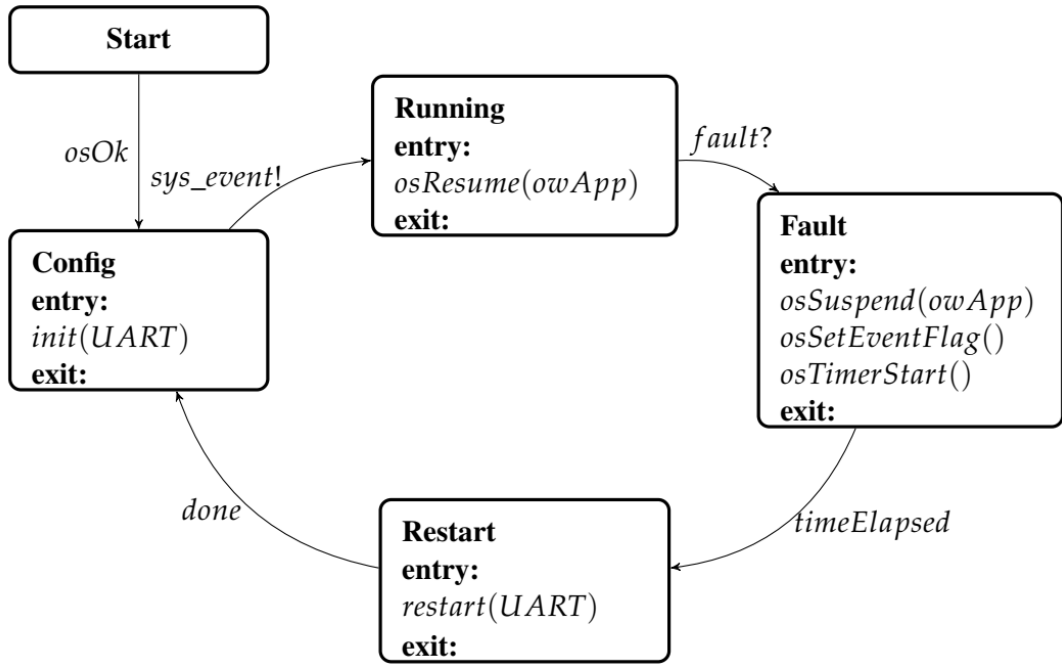
(a) Synchronization state machine.
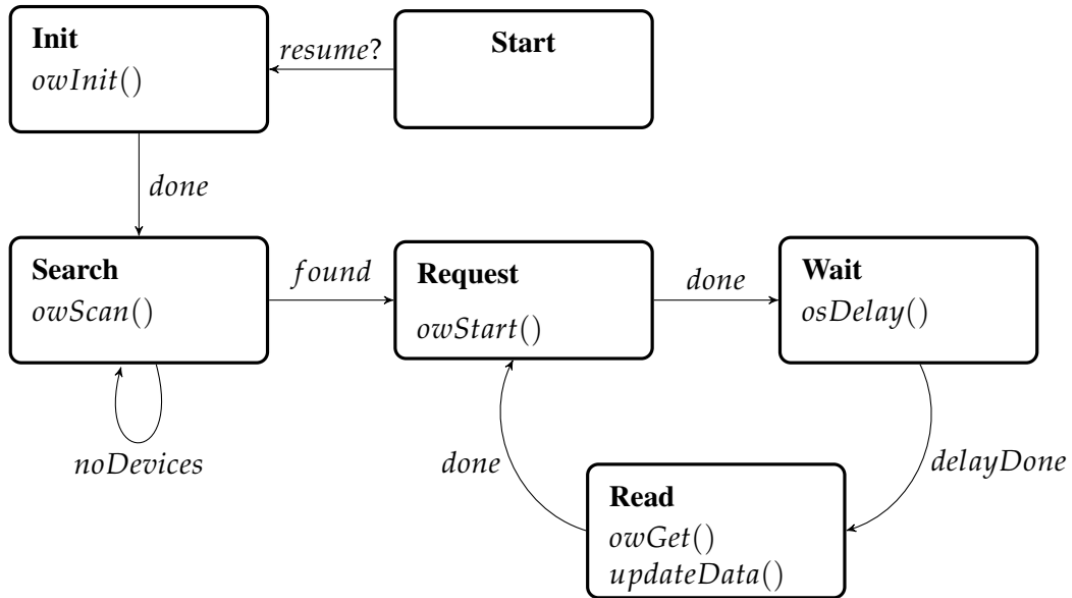
(b) One-wire application state machine.

Figure B.4: DSM for Temperature functionality.

PCB drawings and layout

Adaptors to LAN9252_EVB_SPI

CONNECTORS

7710164 J7 — EVL_ECAT_NSS — 1 2 3 — SCS
7710164 J8 — EVL_ECAT_SCK — 1 2 3 — SCK
7710164 J9 — EVL_ECAT_MOSI — 1 2 3 — SI
7710164 J10 — EVL_ECAT_MISO — 1 2 3 — SO
7710164 J11 — 1 2 3 — GND
7710164 J12 — 1 2 3 — GND
7710164 J13 — 3V3 — 1 2 3 — 3V3

9094398 J4 JTAG
3V3
JTAG_T_VCC_3V3
GND
JTAG_T_JRCLK
JTAG_T_VCP_RX
1 2 3 4 5 6 7 8 9 10 11 12 13 14
JTAG_T_JTMS
JTAG_T_JCLK
JTAG_T_JTDO
JTAG_T_JTDI
JTAG_T_NRST
JTAG_T_VCP_TX

406635 J6 SPI CN
SPI2_SCK — 1
SPI2_NSS — 2
SPI2_MOSI — 3
4 — SPI2_MISO
5 — 1_WIRE_MASTER

PWR

VIN
7527855 D1
6201574 IC1
IN NR/FB
GND OUT
EN
3V3
C9 1 uF
C10 .1 uF
6911133
C11 10nf
6480957
C12 2644595
C13 .1 uF
6480957
6911133

VIN
7527855 D2
6869379
LD1117S50TR U1
VIN VOUT
VOUT
GND
VDD_5V
C1 Tan 10uF
1360159
C2 10 uF 16 V
7190680

VDD_5V
R6 1K
6789875
LD3
6920907

3V3
1349147
C8 4.7 uF 16V

3V3
C4 .1 uF 6480957
C5 .1 uF 6480957
C6 .1 uF 6480957
C7 .1 uF 6480957

VDD_5V 6704714 J1
LED_Ring — 3 2 1

VDD 6704714 J2
LED_Ring_2 — 3 2 1

6704698 J5 PW CN
VIN — 2 1

TODO LIST for second version (2020.08.03)

- Adapt an STM32F4xxx MCU of smaller package (QFN possible) to asses the following peripherals:
--- 2 PWM outputs with independent DMA access
--- 1 UART output with independent DMA access to run 1-Wire Master
--- 1 SPI Master Port (4 wires) for general communication with another sensor
--- 1 SPI Master Port specific for communication with LAN9252
-------- If possible SQI compatible (6 wire)
--- 1 I2C Port
--- SWD/JTAG compatible interface available (10-14 Pin)
--- At least 2 GPIOs available (status led and reset button)

Microcontroller should comply with a similar capabilities of STM32F446ZE + EEPROM*
*To avoid simulation by software

NOTES
The 3V3 Volts pin will be cut out and replaced for a regular male pin
QUAD SPI not considered for this first version

What type of connector? MOLEX53392-0871
XTAL needed or working with internal? NO, works with HSI

Consider PICO CLASP and MicroFit connectors + a stacked approach to shrink the size, similar to ELMO.

Title	PCBAB0001v01A: Power/Conn	Han's Robot Germany GmbH Barmbeker Straße 9a 22303 Hamburg Germany
Size: A4	Number: 1	Revision: 3
Date: 9/17/2020	Time: 11:16:22 PM	Sheet 1 of 2
File: D:\PCB\Projektarbeit\PCB_AxisCommHub\PowerSource.SchDoc		

HAN'S ROBOT

MCU

1962046

U2C
3V3

Pin	Signal
17	VDD
30	VDD
39	VDD
52	VDD
62	VDD
72	VDD
84	VDD
108	VDD
121	VDD
131	VDD
144	VDD
33	VDDA
95	VDDUSB
6	VBAT
32	VREF+

RST 25
BOOT0 138 — BOOT0
PDR ON 143 — 3V3
VCAP 1 71 — VCAP1
VCAP 2 106 — VCAP2

VSS 16, 38, 51, 61, 83, 94, 107, 120, 130
VSSA 31

JTAG_T_NRST

STM32F446ZET6

GND

SW1 Reset 7182455
C3 .1 uF 6480957

U2B

Pin	Signal		Pin	Signal
141	PE0		56	PG0
142	PE1		57	PG1
1	PE2		87	PG2
2	PE3		88	PG3
3	PE4		89	PG4
4	PE5		90	PG5
5	PE6		91	PG6
58	PE7		92	PG7
59	PE8		93	PG8
60	PE9		124	PG9
63	PE10		125	PG10
64	PE11		126	PG11
65	PE12		127	PG12
66	PE13		128	PG13
67	PE14		129	PG14
68	PE15		132	PG15
10	PF0		23	PH0
11	PF1		24	PH1
12	PF2			
13	PF3			
14	PF4			
15	PF5			
18	PF6			
19	PF7			
20	PF8			
21	PF9			
22	PF10			
49	PF11			
50	PF12			
53	PF13			
54	PF14			
55	PF15			

EVL_ECAT_SCK
EVL_ECAT_NSS
EVL_ECAT_MISO
EVL_ECAT_MOSI

STM32F446ZET6

U2A

Pin	Signal		Pin	Signal
34	PA0		26	PC0
35	PA1		27	PC1
37	PA2		28	PC2
40	PA3		29	PC3
41	PA4		44	PC4
42	PA5		45	PC5
43	PA6		96	PC6
100	PA7		97	PC7
101	PA8		98	PC8
102	PA9		99	PC9
103	PA10		111	PC10
104	PA11		112	PC11
105	PA12		113	PC12
109	PA13		7	PC13
110	PA14		8	PC14
	PA15		9	PC15
46	PB0		114	PD0
47	PB1		115	PD1
48	PB2		116	PD2
133	PB3		117	PD3
134	PB4		118	PD4
135	PB5		119	PD5
136	PB6		122	PD6
137	PB7		123	PD7
139	PB8		77	PD8
140	PB9		78	PD9
69	PB10		79	PD10
70	PB11		80	PD11
73	PB12		81	PD12
74	PB13		82	PD13
75	PB14		85	PD14
76	PB15		86	PD15

Led_Ring_Sp_1
Led_Ring_Sp_2
SPI2_MISO
SPI2_MOSI
Led_Ring_1
SPI2_SCK
USER_BUTTON
SPI2_NSS
1_WIRE_MASTER
JTAG_T_VCP_TX — USART3_TX
JTAG_T_VCP_RX — USART3_RX
JTAG_T_JTMS
JTAG_T_JCLK
JTAG_T_JTDI
Led_Ring_3
Led_Ring_4
JTAG_T_JTDO
Led_Ring_2

STM32F446ZET6

VCAP1 — C14 2.2uf 7883048P — GND
VCAP2 — C15 2.2uf 7883048P — GND
BOOT0 — R7 10k 472799 — GND

R2 680R 6790623
R1 680R 6790623
LD2 6920916
LD1 6920953

NOTES for signal outputs
-LED Ring 1 >> PC6 = TIM8_CH1 (USING), TIM3_CH1
-LED Ring 2 >> PB5 = TIM3_CH2 (USING)

All the given pins can be adapted to any other STM32 MCU, mandatory is that complies with the notes about peripherals; see POWERSOURCE diagram.

NOTES from Nucleo
VCAP1/2 connected as in Nucleo
VBAT,VREF,VDDA connected to VDD without C nor Inductor
PDR ON- Always high to enable the power supply supervisor
MCO /PH0- not used since there is no synchronization of clocks with other microcontrollers

Title: *PCBAB0001v01A: MCU*
Size: A4 Number: 1 Revision: 3
Date: 9/17/2020 Time: 11:16:22 PM Sheet 2 of 2
File: D:\PCB\Projektarbeit\PCB_AxisCommHub\MCU.SchDoc

Han's Robot Germany GmbH
Barmbeker Straße 9a
22303
Hamburg
Germany

HAN'S ROBOT

C PCB DRAWINGS AND LAYOUT

ESI file definition

The following code does not include the lines for the object dictionary as those are rather too extensive for the document. Nevertheless, the reader is invited to pay attention to the SM descriptors and the EEPROM configuration word that are those that need to be adjusted accordingly to the used hardware and the Software Stack. For even more information regarding the XML specifications according to EtherCAT protocol, resource [Bec20a] can be reviewed.

```xml
<?xml version="1.0" encoding="utf-8"?>
<EtherCATInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ↙
    xsi:noNamespaceSchemaLocation="EtherCATInfo.xsd" Version="1.2">
  <Vendor>
    <Id>#x00485247</Id>
    <Name>Hans Robot Germany GmbH</Name>
    <!--Here comes a 16X14 image of the company-->
  </Vendor>
  <Descriptions>
    <Groups>
      <Group>
        <Type>COMMBOARD</Type>
        <Name>AXIS_COMM</Name>
        <!--Here comes a 16X14 image of the device-->
      </Group>
    </Groups>
    <Devices>
      <Device Physics="YY">
        <Type ProductCode="#x12783456" RevisionNo="#x00000001">axcommb</Type>
        <Name><![CDATA[AxisCommBoard slave]]></Name>
        <Info>
          <StateMachine>
            <Timeout>
              <PreopTimeout>8000</PreopTimeout>
              <!--4 Times from original value-->
              <SafeopOpTimeout>36000</SafeopOpTimeout>
              <BackToInitTimeout>20000</BackToInitTimeout>
              <BackToSafeopTimeout>800</BackToSafeopTimeout>
            </Timeout>
          </StateMachine>
          <Mailbox>
```

```
      <Timeout>
        <RequestTimeout>400</RequestTimeout>
        <!--4 Times from original value-->
        <ResponseTimeout>8000</ResponseTimeout>
      </Timeout>
    </Mailbox>
  </Info>
  <!-- Added from SSC TOOL-->
  <GroupType>COMMBOARD</GroupType>
  <Profile>
    <ProfileNo>5001</ProfileNo>
    <Dictionary>
      <DataTypes>
          <!--DataTypes are omitted-->
      </DataTypes>
      <Objects>
          <!--Objects are omitted-->
      </Objects>
    </Dictionary>
  </Profile>
  <!-- Added from SSC TOOL-->
  <Fmmu>Outputs</Fmmu>
  <Fmmu>Inputs</Fmmu>
  <!--Is this Fmmu needed?-->
  <Fmmu>MBoxState</Fmmu>
  <!-- ADDRESSES to be checked. So far the addresses match with ↙
      linux_lan9252 example-->
  <Sm ControlByte="#x26" DefaultSize="128" Enable="1" ↙
      StartAddress="#x1000">MBoxOut</Sm>
  <Sm ControlByte="#x22" DefaultSize="128" Enable="1" ↙
      StartAddress="#x1080">MBoxIn</Sm>
  <!--SMs addresses adjusted to from SSC file to SOES-->
  <Sm DefaultSize="8" StartAddress="#x1100" ControlByte="#x64" ↙
      Enable="1">Outputs</Sm>
  <Sm DefaultSize="36" StartAddress="#x1400" ControlByte="#x20" ↙
      Enable="1">Inputs</Sm>
  <RxPdo Fixed="true" Mandatory="true" Sm="2">
    <Index>#x1600</Index>
    <Name>MasterCMDs process data mapping</Name>
    <!--Entries are omitted-->
  </RxPdo>
  <!-- Added from SSC TOOL-->
  <TxPdo Fixed="true" Mandatory="true" Sm="3">
    <Index>#x1A00</Index>
    <Name>Input mapping 0</Name>
    <!--Entries are omitted-->
  </TxPdo>
  <!-- Added from SSC TOOL-->
  <Mailbox DataLinkLayer="true">
    <!-- From SSC TOOL: PDOAssign false, pdoConfig false, ↙
        CompleteAccess true,segmentsdo true-->
    <CoE SdoInfo="true" CompleteAccess="false" PdoConfig="false" ↙
        SegmentedSdo="true" PdoAssign="false" PdoUpload="true"></CoE>
```
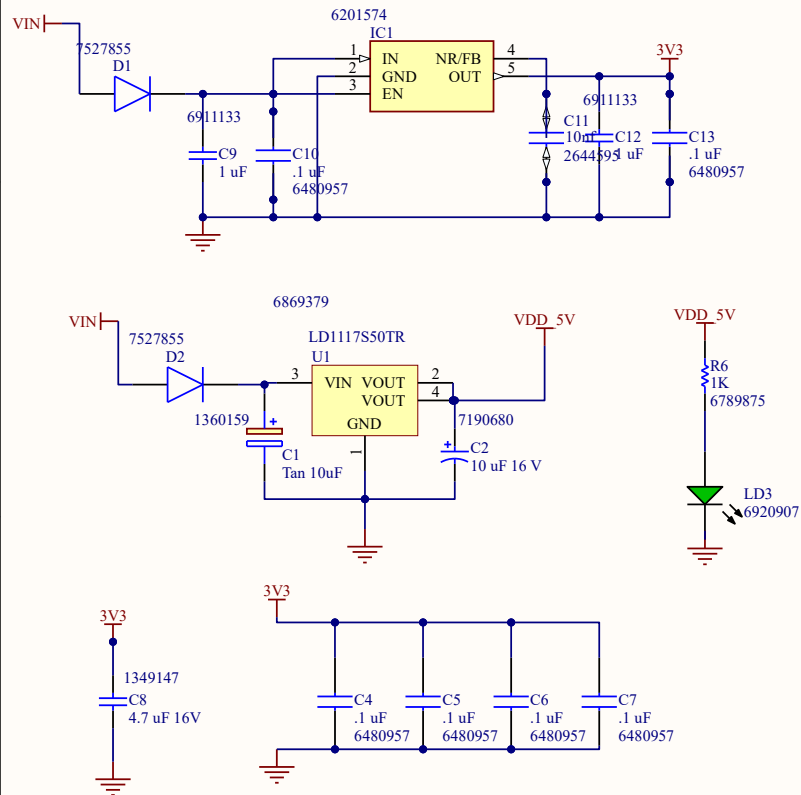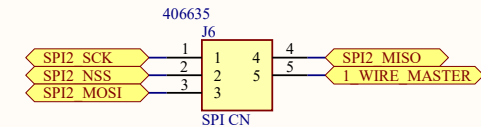
```xml
        </Mailbox>
        <Dc>
          <OpMode>
            <Name>DcOff</Name>
            <Desc>DC unused</Desc>
            <AssignActivate>#x0000</AssignActivate>
          </OpMode>
        </Dc>
        <!--This Eeproms config word matches current App -->
        <Eeprom>
          <ByteSize>512</ByteSize>
          <ConfigData>800200cc8813ff00CACA0080</ConfigData>
          <!--Bootstrap is disabled since FoE is not present-->
          <!--<BootStrap>0010800080108000</BootStrap>-->
        </Eeprom>
      </Device>
    </Devices>
  </Descriptions>
</EtherCATInfo>
```

Listing D.1: Part of the EtherCAT Slave Information file.

D ESI FILE DEFINITION

Source codes

This appendix enlists the fundamental source codes within this implementation. The order is as follows:

- Configuration header file for the main features where the full list of errors and events can be seen in E.1.

- Code for the creation and configuration of threads plus some auxiliary functions in E.2.

- Main code for the Event Handler in E.3.

- Main code for the ECAT DSM in E.4.

- Main code for the SOES DSM in E.5.

- Main code for the LED DSM in E.6.

- Code for the Temperature Application in E.7.

```
/*
 * AxisCommHub_definitions.h
 *
 *  Created on: Jun 11, 2020
 *      Author: Carlos Reyes
 */

#ifndef AXISCOMMHUB_DEFINITIONS_H_
#define AXISCOMMHUB_DEFINITIONS_H_


// General include files

#include "LAN9252_spi.h"
#include "main.h"

// Definition of peripherals
#define   NUM_OF_SENSORS        15u
```

```
#define MAX_OF_LEDRINGS       4     // This values should be modified in the ↙
    LED library
#define NUM_OF_LEDRINGS       2
#define NUM_OF_LEDS_PER_RING  5     // If the number is not the same for all ↙
    LED Rings, then change directly within the LED library

// Generic definitions
#define  TRUE              1
#define  FALSE             0
#define FAILED            -1

// Timing for SOES
#define SOES_REFRESH_CYCLE   20u       //in Systicks

//    Mind the following structure for event flags (See #define ↙
    MAX_BITS_EVENT_GROUPS     24U)
// | 8 reserved bits | 14 bits (16383d) error space | 10 bit individual event ↙
    flags|

// Declaration of event flags (10 available)
#define SYS_EVENT        (1<<0)
#define LED_EVENT        (1<<1)
#define TSENS_EVENT       (1<<2)
#define ECAT_EVENT        (1<<3)
#define  TASKM_EVENT       (1<<4)


// Offsets
#define ERR_OFFSET           1000u
#define EV_OFFSET         2000u
#define  SHIFT_OFFSET        10u
// Declaration of errors
#define  ERR_SYS_NONE        0
#define  ERR_SYS_UNKNOWN        101


#define  ERR_TEMP_SENS_INIT       1101u
#define ERR_TEMP_SENS_LOST      1102u
#define ERR_TEMP_SENS_TIMEOUT   1103u
#define ERR_TEMP_SENS_OVERHEAT    1104u
#define  ERR_TEMP_DSM_FAULT       1105u

#define ERR_LED_INIT        1201u
#define ERR_LED_TIMEOUT        1202u
#define ERR_LED_SEND        1203u
#define  ERR_LED_OSTIM        1204u
#define  ERR_LED_DSM_FAULT     1205u

#define  ERR_ECAT_INIT         1301u
#define ERR_ECAT_COMM_LOST      1302u
#define ERR_ECAT_TIMEOUT        1303u
#define ERR_ECAT_DSM_FAULT      1304u
```

```c
#define ERR_ECAT_CMD_FAULT      1305u
#define ERR_ECAT_CMD_SOFTFAULT   1306u

// Definition of internal events
#define EV_TEMP_DSM_INIT      2101

#define EV_LED_DSM_INIT       2201
#define EV_LED_UPTD           2211

#define EV_ECAT_ESC_INIT       2301
#define EV_ECAT_APP_OP         2311
#define EV_ECAT_APP_INIT       2312
#define EV_ECAT_APP_NOOP       2313
#define EV_ECAT_DSM_INIT       2321
#define EV_SOES_RESPAWNED      2331
#define EV_ECAT_CMD_ACK        2351
#define EV_ECAT_CMD_LED_TOGGLE    2352


//   Mind the following structure for status variable (according to size of ↙
    errors and events (16383d))
// | 14 bits last event space | 14 bits last error space | 4 bit General ↙
    state |
#define  STATUS_OFFSET_FOR_ERR   4u
#define  STATUS_OFFSET_FOR_EV 18u

#define STATUS_DATA_MASK      0x3FFF
#define STATUS_SHORT_MASK     0x0F

#define  STATUS_INIT          1u
#define  STATUS_STARTED       2u
#define  STATUS_NO_ERRORS     4u
#define STATUS_SOFT_ERRORS    5u


// Definition of specific timeouts
// ECAT/SOES
#define  ESC_INIT_TIMEOUT  5u
#define  ESC_REFRESH_TIMEOUT  10000u

// Definitions of ETHERCAT STATE MACHINE
// See esc.h for ALstatus Reg
// e.g. ESCop

// Auxiliar definitions for HAL adaptation

#define CHANNEL_FOR_LED1       TIM_CHANNEL_1
#define CHANNEL_FOR_LED2       TIM_CHANNEL_2
#define CHANNEL_ACTIVE_FOR_LED1  HAL_TIM_ACTIVE_CHANNEL_1
#define CHANNEL_ACTIVE_FOR_LED2 HAL_TIM_ACTIVE_CHANNEL_2


// Overall test definitions
```

```
#define ECAT_UPDT_PERIOD_TEST_IN_MS        10u   //1>> RT 10-15 >> Industrial ↙
    RT  100>> Non-critical variables monitoring
#define  ECAT_CONNECTION_CHECK_PERIOD_IN_MS  10u
#define  ECAT_CHECK_PERIOD_FACTOR          ↙
    ECAT_CONNECTION_CHECK_PERIOD_IN_MS/ECAT_UPDT_PERIOD_TEST_IN_MS
#define COMM_TESTING_TIMES              100u


// Taken from linux definitions and needed by SOES
#define BIT(nr) (1UL << (nr))

#endif /* AXISCOMMHUB_DEFINITIONS_H_ */
```

Listing E.1: Main definition and configuration header file.

```
 /*
 * SMs.c
 *
 *  Created on: Jun 3, 2020
 *      Author: Carlos Reyes
 */

#include "main.h"
#include "AxisCommHub_definitions.h"
#include "SMs.h"
#include "smEcat.h"

extern osTimerId_t timerEcatSM,timerEcatSOES;


/*--------------------------------------------TASKS for ↙
    SMs------------------------------------------------------------*/
// Task Handlers declared within the SMs.h


uint32_t tempSensTBuffer[ 192 ];
StaticTask_t tempSensTControlBlock;
const osThreadAttr_t tempSensT_attributes = {
        .name = "tempSensT",
        .stack_mem = &tempSensTBuffer[0],
        .stack_size = sizeof(tempSensTBuffer),
        .cb_mem = &tempSensTControlBlock,
        .cb_size = sizeof(tempSensTControlBlock),
        .priority = (osPriority_t) osPriorityNormal,
};


uint32_t ledRingsTBuffer[ 192 ];
StaticTask_t ledRingsTControlBlock;
const osThreadAttr_t ledRingsT_attributes = {
        .name = "ledRingsT",
        .stack_mem = &ledRingsTBuffer[0],
```

```c
        .stack_size = sizeof(ledRingsTBuffer),
        .cb_mem = &ledRingsTControlBlock,
        .cb_size = sizeof(ledRingsTControlBlock),
        .priority = (osPriority_t) osPriorityNormal,
};


uint32_t ecatSMTBuffer[ 192 ];
StaticTask_t ecatSMTControlBlock;
const osThreadAttr_t ecatSMT_attributes = {
        .name = "ecatTSM",
        .stack_mem = &ecatSMTBuffer[0],
        .stack_size = sizeof(ecatSMTBuffer),
        .cb_mem = &ecatSMTControlBlock,
        .cb_size = sizeof(ecatSMTControlBlock),
        .priority = (osPriority_t) osPriorityNormal,
};


uint32_t eventHTBuffer[ 128 ];
StaticTask_t eventHTControlBlock;
const osThreadAttr_t eventHT_attributes = {
        .name = "eventHT",
        .stack_mem = &eventHTBuffer[0],
        .stack_size = sizeof(eventHTBuffer),
        .cb_mem = &eventHTControlBlock,
        .cb_size = sizeof(eventHTControlBlock),
        .priority = (osPriority_t) osPriorityAboveNormal,
};

/*----------------------------------------------AUXILIAR
    TASKS-------------------------------------------------------------*/

/*------------------------ECAT------------------------------------*/
uint32_t ecatTestTBuffer[ 192 ];
StaticTask_t ecatTestTControlBlock;
const osThreadAttr_t ecatTestT_Attributes = {
        .name = "ecatTestT",
        .stack_mem = &ecatTestTBuffer[0],
        .stack_size = sizeof(ecatTestTBuffer),
        .cb_mem = &ecatTestTControlBlock,
        .cb_size = sizeof(ecatTestTControlBlock),
        .priority = (osPriority_t) osPriorityHigh3,
};

uint32_t ecatSOESTBuffer[1088];
StaticTask_t ecatSOESTControlBlock;
const osThreadAttr_t ecatSOEST_Attrbuttes = {
        .name = "ecatSOEST",
        .stack_mem = ecatSOESTBuffer,
        .stack_size = sizeof(ecatSOESTBuffer),
        .cb_mem = &ecatSOESTControlBlock,
        .cb_size = sizeof(ecatSOESTControlBlock),
```

```
        .priority = (osPriority_t) osPriorityAboveNormal,
};


/*----------------------System Monitor----------------------------*/

uint32_t taskManagerTBuffer[ 192 ];
StaticTask_t taskManagerTControlBlock;
const osThreadAttr_t taskManagerT_Attributes = {
        .name = "taskManagerT",
        .stack_mem = &taskManagerTBuffer[0],
        .stack_size = sizeof(taskManagerTBuffer),
        .cb_mem = &taskManagerTControlBlock,
        .cb_size = sizeof(taskManagerTControlBlock),
        .priority = (osPriority_t) osPriorityHigh,
};



/********************************************* Extern Variables from LED Rings ↙
    Multichannel ↙
    ********************************************************************/

//volatile uint8_t currentColors[MAX_OF_LEDRINGS]; //Global array for colors ↙
    to be updated, this will be changed continuously by ↙
    EventHandler/Notification //CHCKME this is shared memory
extern volatile uint8_t dmaLed1_rcvd, dmaLed2_rcvd,refreshTimeoutLed;



/********************************************* Variables to debug ↙
    ***************************************************************/
osStatus_t static ecatStatus,uartPrintStatus;
uint32_t *heapObserver0,*heapObserver1,*heapObserver2;

/***************************************** Var task manager ↙
    ***********************************************************/
static osThreadState_t status_ecatTestT, status_ecatT, ↙
    status_evHT,status_uartPT,status_tSensT,status_ledsT, ↙
    status_taskMT,status_ecatSOEST;
osTimerId_t timerTsens; //IMPRVME   This may be a local variable to save memory
static uint8_t timedoutTsens;



/*------------------------Here start the definitions of the functions needed ↙
    by SM --------------*/
/*--------------------------------- Task Manager functions ↙
    ------------------------------------*/
/* *
 * @brief   This function will update the status for each task and terminate ↙
    a thread if needed
 * */

void taskManger(void * argument) {
```

```
        osStatus_t status;
        while (1) {
            if (restartTaskManFlag) {
                restartTaskManFlag = FALSE;
                osDelay(100);
                status = osThreadSuspend(ecatSOESTHandler);
                status = osThreadTerminate(ecatSOESTHandler);
                ecatSOESTHandler = osThreadNew(soes, NULL, &ecatSOEST_Attrbuttes);
                status = osThreadSuspend(ecatSOESTHandler);
                osEventFlagsSet(evt_sysSignals, TASKM_EVENT|EV_SOES_RESPAWNED);
            }

            status_ecatTestT = osThreadGetState(ecatTestTHandler);
            status_ecatT = osThreadGetState(ecatSMTHandle);
            status_ecatSOEST = osThreadGetState(ecatSOESTHandler);

            //osThreadYield(); //Yield to any other thread that may be ready
            //osDelay(1000);          //1ms update rate
            osEventFlagsWait(taskManSignals, TASKM_EVENT,osFlagsWaitAny , ↙
                osWaitForever);

            status_ecatSOEST = osThreadGetState(ecatSOESTHandler);
            status_evHT = osThreadGetState(eventHTHandle);
            status_uartPT = osThreadGetState(uartPrintTHandler);
            status_tSensT = osThreadGetState(tempSensTHandle);
            status_ledsT = osThreadGetState(ledRingsTHandle);
            status_taskMT = osThreadGetState(taskManagerTHandler);
        }

        //osThreadTerminate(taskManagerTHandler);   //If ever jumps out the loop

}

/* *
 * @brief   This function adds the threads to be executed to the OS and the ↙
     general signals
 * */
void addThreads(void) {
    evt_sysSignals = osEventFlagsNew(NULL);
    if (evt_sysSignals == NULL){
        //Handle error
        __NOP();
    }
    taskManSignals = osEventFlagsNew(NULL);
    if (taskManSignals == NULL){
        //Handle error
        __NOP();
    }

    heapObserver0 = evt_sysSignals;
    heapObserver1 = taskManSignals;

    //   Initializing main threads
```

```
        tempSensTHandle = osThreadNew(tempSens_SM, NULL, &tempSensT_attributes);
        ledRingsTHandle = osThreadNew(ledRings_SM, NULL, &ledRingsT_attributes);
        ecatSMTHandle = osThreadNew(ecat_SM, NULL, &ecatSMT_attributes);
        eventHTHandle = osThreadNew(eventH_SM, NULL, &eventHT_attributes);

        //  Auxiliar tasks

        ecatSOESTHandler = osThreadNew(soes, NULL, &ecatSOEST_Attrbuttes);
        ecatStatus = osThreadSuspend(ecatSOESTHandler);
        taskManagerTHandler = osThreadNew(taskManger, NULL, ↙
            &taskManagerT_Attributes);

        sysState = STATUS_INIT;
        //Debug tasks
        //eventTesterTHandler = ↙
            osThreadNew(eventTesterTask,NULL,&eventTesterT_Attributes);    ↙
            //Pending This task could start before the system is ready
    }
```

Listing E.2: Part of the code to declare threads and other auxiliar functions.

```
    /*
     * smEvH.c
     *
     *  Created on: Jun 26, 2020
     *      Author: Carlos Reyes
     */



    #include "SMs.h"
    #include "smEvH.h"



    //External variables



    /*
     * @brief Sate Machine for error/event handler
     *
     */

    void eventH_SM (void * argument) {

        uint32_t status,flags;
        uint16_t event_data;

        while(1) {      //Infinite loop enforced by task execution

            switch (evH_step) {
            /*----------------------------------------------------------------*/
            case  evh_init:
                evH_initFlag = TRUE;
```

```
                evH_step = evH_waiting;
                break;
        /*-------------------------------------------------------------------*/
        case  evH_waiting:
            //   entry:
            status = osEventFlagsWait(evt_sysSignals, SYS_EVENT, ↙
                osFlagsWaitAny, osWaitForever);
            //   exit:
            evH_step = evH_check;
            break;
        /*-------------------------------------------------------------------*/
        case  evH_check:
            //   entry:
            status = osEventFlagsGet(evt_sysSignals);   //TODO check whether ↙
                the flags are cleared
            event_data = (status>>SHIFT_OFFSET);
            if (event_data>EV_OFFSET) {
                evH_step = evH_notifHandling;
            }
            else if (event_data>ERR_OFFSET) {
                evH_step = evH_errHandling;
            }
            else {
                __NOP();
                evH_step = evH_error;
            }

            //   exit:
            break;
        /*-------------------------------------------------------------------*/
        case evH_errHandling:
            // entry:
            if (event_data == ERR_ECAT_DSM_FAULT ||
                    ERR_ECAT_CMD_FAULT || ERR_LED_DSM_FAULT ||
                    ERR_TEMP_DSM_FAULT || ERR_TEMP_SENS_OVERHEAT ||
                    ERR_ECAT_COMM_LOST
                    ) {
                // PENDING Add an specific action depending on the error
                errorFlag = TRUE;
                normalFlag = FALSE;
            }   //
            else if (event_data == ERR_TEMP_SENS_LOST ||
                    ERR_ECAT_CMD_SOFTFAULT) {
                // PENDING Add an specific action depending on the warning
                warningFlag = TRUE;
                normalFlag = FALSE;
            }
            else {
                // do nothing
                warningFlag = TRUE;
                __NOP();
                evH_step = evH_error;
                break;
```

```
            }

            sysState &= ~(STATUS_DATA_MASK<<STATUS_OFFSET_FOR_ERR);
            sysState |= ↙
                ((event_data&STATUS_DATA_MASK)<<STATUS_OFFSET_FOR_ERR);

            //exit
            flags = osEventFlagsGet(evt_sysSignals);   // SYS_EVENT flag ↙
                is cleared while returning from eventsFlagWait functions
            osEventFlagsClear(evt_sysSignals,flags);   // Due to the ↙
                priorities, it is not possible that another task creates ↙
                events while this SM is running
            evH_step = evH_waiting;
            break;
    /*----------------------------------------------------------------*/
        case evH_notifHandling:
            //  entry:
            if (event_data == EV_TEMP_DSM_INIT) {
                // If needed add an specific action depending on the ↙
                    notification
                temp_initFlag = TRUE;
            }   //
            else if (event_data == EV_LED_DSM_INIT) {
                // If needed add an specific action depending on the ↙
                    notification
                led_initFlag = TRUE;
            }
            else if (event_data == EV_ECAT_DSM_INIT) {
                // If needed add an specific action depending on the ↙
                    notification
                ecat_initFlag = TRUE;
            }
            else if (event_data == EV_ECAT_CMD_ACK) {
                // Add an specific action depending on the ECAT cmd
                sysState = 0xFF;
                //evH_step = evH_ecatCMD;
                //break;
            }
            else if (event_data == EV_ECAT_APP_OP) {
                // Add an specific action depending on the warning
                if ((sysState&STATUS_SHORT_MASK) == STATUS_STARTED ) { ↙
                    //&& ((sysState>>STATUS_OFFSET_FOR_ERR) & ↙
                    STATUS_DATA_MASK) == ERR_SYS_NONE)
                    warningFlag = FALSE;
                    normalFlag = TRUE;
                    if(((sysState>>STATUS_OFFSET_FOR_ERR) & ↙
                        STATUS_DATA_MASK) == ERR_ECAT_COMM_LOST) {
                        errorFlag = FALSE;
                    }
                }

            }
            else if (event_data == EV_ECAT_APP_INIT) {
```

```
                    // If needed add an specific action depending on the
                        notification
                    warningFlag = TRUE;
                    normalFlag = FALSE;
                }
                else if (event_data > EV_ECAT_CMD_ACK) {
                    // PENDING This could be either an error or a warning or
                        a shortcut to CMD step
                    //evH_step = evH_ecatCMD;
                    __NOP();
                }
                else {
                    // do nothing
                    __NOP();

                    evH_step = evH_error;
                    break;
                }
                if ((sysState&STATUS_SHORT_MASK) == STATUS_INIT &&
                    temp_initFlag && led_initFlag && ecat_initFlag ) {
                    status = (sysState>>STATUS_OFFSET_FOR_ERR);  // status
                        used temporary
                    sysState = STATUS_STARTED|(status<<STATUS_OFFSET_FOR_ERR);
                }

                sysState &= ~(STATUS_DATA_MASK<<STATUS_OFFSET_FOR_EV); //
                    Clearing the previous Event
                sysState |=
                    ((event_data&STATUS_DATA_MASK)<<STATUS_OFFSET_FOR_EV);

                //exit

                flags = osEventFlagsGet(evt_sysSignals);   // SYS_EVENT flag
                    is cleared while returning from eventsFlagWait functions
                osEventFlagsClear(evt_sysSignals,flags);   // Due to the
                    priorities, it is not possible that another task creates
                    events while this SM is running
                evH_step = evH_waiting;
                break;
        /*------------------------------------------------------------------*/
            case evH_ecatCMD:
                //  entry:
                //  PENDING An specific ecat command handler need to be
                    defined
                //eventHandled = TRUE; //Not needed so far since Flag is
                    being cleared
                __NOP();

                //exit
                status = SYS_EVENT |(event_data<<ERR_OFFSET);
                osEventFlagsClear(evt_sysSignals, status);
                evH_step = evH_waiting;
                break;
```

73

```
            /*-------------------------------------------------------------------*/
                case evH_error:
                    //  entry:
                    // This DSM terminates since an error in the event handler ⤸
                        is critical and should be debugged by programmer.
                    errorFlag = TRUE;
                    sysState = ERR_SYS_UNKNOWN;


                    //  exit
                    osThreadSuspend(eventHTHandle);
                    break;
            /*-------------------------------------------------------------------*/
                default:
                    __NOP();
                }
        }

        //osThreadTerminate(eventHTHandle);


    }
```

🔲 **Listing E.3:** Main source code for Event Handler DSM.

```
/*
 * smEcat.c
 *
 *  Created on: Jun 26, 2020
 *      Author: Carlos Reyes
 */

#include "SMs.h"
#include "smEcat.h"
#include "LAN9252_spi.h"
#include "esc.h"
#include "esc_hw.h"



/*-------------------Variable used specially in this ⤸
    SM-------------------------------------------------*/

volatile uint8_t timedoutEcat,restartEcatFlag;
static uint8_t escAPPok;


/*--------------------------External ⤸
    variables-------------------------------------*/
extern TIM_HandleTypeDef htim5;     //From main.c
extern int lan9252; //From lan9252_spi.c
extern volatile uint8_t ecatDMArcvd;   //Defined in LAN9252 library

// External variables for synchronizing with soes SM
extern volatile uint8_t soesTimeoutFlag;
```

```c
osTimerId_t timerEcatSOES; // << CHCKME This is used by SOES library
extern _ESCvar ESCvar;     // << Instance of the ESC that are declared within ⤸
    the sampleApp.c
void APP_safeoutput (); //CHCKME
extern _MBXcontrol MBXcontrol[];
extern uint8_t MBX[];
extern _SMmap SMmap2[];
extern _SMmap SMmap3[];

/*--------------------------smEcat ⤸
    functions----------------------------------------*/
/*
 * @brief Sate Machine for overall task of eCAT interface
 *
 */

void ecat_SM (void * argument) {

   //TEMP for TESTING
   uint16_t ESC_status;
   //FINISHES
   uint8_t error = 0;
   uint8_t firstExec = 1;
   uint32_t rcvdData;

   osStatus_t timerStatus;
   osTimerId_t timerEcatSM,timerEcatSM2;//timerEcatSOES;
   uint32_t timerDelay;
   timerEcatSOES = osTimerNew(timeoutSMCallback_ecat, osTimerOnce, NULL, NULL);
   //timerEcatSM = osTimerNew(timeoutSMCallback_ecat, osTimerOnce, NULL, NULL);
   //timerEcatSM2 = osTimerNew(timeoutSMCallback_ecat, osTimerOnce, NULL, NULL);


   if (timerEcatSM == NULL) {
      __NOP(); //Handle the problem of creating the timer
   }
   if (timerEcatSOES == NULL) {
      __NOP(); //Handle the problem of creating the timer
   }
   while(1) {     //Infinite loop enforced by task execution

      switch (ecat_step) {
      /*--------------------------------------------------*/
         case  ec_config:
            // action
            if(   ecat_SPIConfig(&hspi4) == FAILED) error++;

            //exit
            if (error) {
               notifyError(ERR_ECAT_INIT);
               error = 0;
```

```
                ecat_step = ec_fault;
            }   //TODO this should be sort of a signal, this should not stop
                 the execution of this SM
        else {
            lan9252 = open ("LOCAL_SPI", O_RDWR, 0);
            ecat_step = ec_checkConnection;
        }

        break;
/*--------------------------------------------------------*/
    case  ec_checkConnection:
        // action
        timerDelay = 40u;
        timerStatus = osTimerStart(timerEcatSM, timerDelay);  //Timeout for
             SOES
        if (timerStatus != osOK) {
            notifyError(ERR_LED_OSTIM); // CHCKME This is a internal OS error.
        }


        osThreadResume(ecatSOESTHandler);   //>> SOES SM starts with higher
             priority
        osEventFlagsWait(evt_sysSignals,ECAT_EVENT, osFlagsWaitAny,
             osWaitForever);

        // exit
        if (restartEcatFlag) {
            notifyError(ERR_ECAT_TIMEOUT);
            restartEcatFlag = FALSE;
            ecat_step = ec_fault;
        }
        else {
            if (osTimerIsRunning(timerEcatSOES)) { //PENDING This OSTimer
                 could overflow even when there is no timeout due to other
                 threads allocated by the OS
                if (osTimerStop(timerEcatSOES) != osOK) {
                    notifyError(ERR_ECAT_OSTIM);
                }
            }
            ecat_step = ec_connected;
        }
            break;
/*--------------------------------------------------------*/
    case  ec_waitDMA: // This state is used only if communication is test
         before soes app has started
        osThreadYield();
        osEventFlagsWait(evt_sysSignals, ECAT_EVENT,osFlagsWaitAny,
             osWaitForever);

        //exit
        if(ecatDMArcvd) {    //This DMA rcvd can be the full buffer finished
             transmiting interruption
            ecatDMArcvd = FALSE;
```

```c
            if(ecatVerifyResp(TEST_BYTE_OFFSET) != FAILED) {
                notifyEvent(EV_ECAT_APP_READY);
                ecat_step = ec_idle;
            }  //TODO this should be improved to use a shared buffer with the
                data comming from SPI or something similar
            else {
                notifyError(EV_ECAT_APP_NOK);
                ecat_step = ec_fault;
            }
            break;
        }  //TODO DMAReceived should be changed by interruption

        if(timedoutEcat) {
            notifyError(ERR_ECAT_TIMEOUT);
            timedoutEcat = FALSE;
            ecat_step = ec_fault;
        }  //The timeout callback function modifies this error flag
        break;
/*--------------------------------------------------------*/
    case  ec_connected:
        // entry
        if (firstExec) {
            firstExec = FALSE;
            osThreadResume(ecatSOESTHandler);
        }

        // action
        if (ESCvar.ALstatus == ESC_APP_OK && !escAPPok) {
            escAPPok = TRUE;
            osEventFlagsSet(evt_sysSignals, ECAT_EVENT);
        }
        else if((ESCvar.ALstatus & ESCop)&&!escAPPok){
            escAPPok = TRUE;
            notifyEvent((uint8_t)EV_ECAT_APP_OP);
        }
        else if((ESCvar.ALstatus & ESCinit)&&!escAPPok){
            notifyEvent((uint8_t)EV_ECAT_APP_NOK);
        }

        osDelay(100u); // This could be a definition

        // exit
        if (restartEcatFlag) {
            restartEcatFlag = FALSE;
            notifyError(ERR_ECAT_COMM_LOST);
            ecat_step = ec_fault;
        }

        break;
/*--------------------------------------------------------*/
    case  ec_sleep:
        __NOP();
        osThreadSuspend(ecatSMTHandle);
```

```
            break;
      /*-------------------------------------------------------*/
         case  ec_fault:
            //entry

            //action
            escAPPok = FALSE;
            firstExec = FALSE;
            //Task manager should have restarted the SOES Thread
            //osEventFlagsWait(evt_sysSignals,TASKM_EVENT|EV_SOES_RESPAWNED, ⤸
                osFlagsWaitAny, osWaitForever);
            //exit
            ecat_step = ec_restart;
            break;
      /*-------------------------------------------------------*/
         case  ec_restart:
            //action

            ecat_deinit(&hspi4); // CHCKME whether error prompts due to shared ⤸
                resource
            //updateTaskManFlag = TRUE;
            //osEventFlagsSet(taskManSignals, TASKM_EVENT); //<<Adds SOES Thread ⤸
                again through a higher priority system task
            //HAL_StatusTypeDef halstatus = HAL_TIM_Base_Stop_IT(&htim5);
            osDelay(3000);     //Waits to restart the communication, meanwhile ⤸
                another task is assessed

            //exit
            ecat_step = ec_config;
            break;
         default:
            __NOP();
         }
   }

   //osThreadTerminate(ecatSMTHandle);

}

/*---------------------------------------Temporary functions(on ⤸
    develop)-------------------------------------------------*/


/* *
 * @brief   This is the timeout callback function for  ECAT
 * */

void timeoutSMCallback_ecat(void * argument) {
   //do something
   uint32_t status;
   HAL_StatusTypeDef halstatus;
```

```
    //status = osThreadSuspend(ecatSOESTHandler);   //<< Cannot be called within
        ISR
    //suspendTaskManFlag = TRUE;
    //status = osEventFlagsSet(taskManSignals, TASKM_EVENT);
    //restartEcatFlag = TRUE;
    halstatus = HAL_TIM_Base_Stop_IT(&htim5);
// status = osEventFlagsSet(evt_sysSignals, SYS_EVENT);
}
void timeoutSOESCallback_ecat(void * argument) {
    //do something
    //osThreadSuspend(ecatSOESTHandler);
    restartEcatFlag = TRUE;
    //osEventFlagsSet(taskManSignals, TASKM_EVENT);
    __NOP();
}


/* *
 * @brief   This is the timeout callback function specially for SOES. The timers
     are oneshot, no need for stop them.
 *              This way the queues are not overflown.
 * */
void timeoutSOESCallback(void * argument) {
    uint32_t status,test;
    test = *(uint32_t *)argument;
    if(test == 1) {
        __NOP(); //Timeout in init
        //Notify event
    }
    else {
        __NOP(); //Timeout while communicating
        //Notify event
    }
    soesTimeoutFlag = TRUE;
    restartEcatFlag = TRUE;     //Flag for taskmanager should be before flag is set.
    //restartTaskManFlag = TRUE;
    //status = osEventFlagsSet(taskManSignals, TASKM_EVENT);

    }
```

■ **Listing E.4:** Main source code for ECAT DSM.

```
    /*
    * soesApp.c
    *
    *  Created on: Jul 16, 2020
    *      Author: Carlos Reyes
    *      Comments: Based on the rtl_slavedemo provided within the SOES Library.
    *       GNU General Public License header copied from the original file
    */


    // Comments from original file.
```

```
/*
 * Licensed under the GNU General Public License version 2 with exceptions. See
 * LICENSE file in the project root for full license information
 */

//#include <kern.h>        // << Kernel added within  the CMSIS+FreeRTOS
#include "cmsis_os.h"
#include "AxisCommHub_definitions.h"
#include "ecat_slv.h"
#include "utypes.h"
//#include "bsp.h"          // << BSAP compatibility already included in the ⤸
    main file, stm32f446ze
#include "bootstrap.h"

//include for testing
#include "smEcat.h"

// External global variables related to DATA
extern int16_t gv_temperatureData[NUM_OF_SENSORS];    // Declared in SMs.c

// Variables needed for synchronization with SMs
extern osThreadId_t ecatSOESTHandler;
extern osTimerId_t timerEcatSOES;
osTimerId_t timerSOES;
extern volatile osEventFlagsId_t evt_sysSignals,taskManSignals;
extern uint32_t *heapObserver0,*heapObserver1,*heapObserver2;

// Variables needed mainly for this SOES SM
enum enum_soesStates {s_start, s_init1, s_init2, s_timerset, s_slaveloop, ⤸
    s_sleep, s_nostep, s_error}soes_step;
volatile uint8_t soesTimeoutFlag;

/* Application variables */
_Rbuffer    Rb;
_Wbuffer    Wb;
_Cbuffer    Cb;

 uint16_t masterCommand,masterTest0,masterTest1,masterTest2;

/*-----Test variables--------------------------------------*/
uint8_t testInputButton;
uint8_t testOutputLed;

/*-----App functions--------------------------------------*/

void cb_get_inputs (void)
{
    Rb.status += 0xFA;  // These variables will be updated by other SMs
    Rb.event += 0xFA;
    Rb.error += 0xFA;
    for (uint8_t i = 0; i < NUM_OF_SENSORS;i++) {
        Rb.temp[i] = gv_temperatureData[i]; //
```

```
        }
}


void cb_set_outputs (void)
{
    //   Outputs from the master
    masterCommand = Wb.command;      // In the future this will be a shared ⤸
        memory
    masterTest0 = Wb.testVal0;
    masterTest1 = Wb.testVal1;
    masterTest2 = Wb.testVal2;

}

void soes (void * arg)
{
    uint32_t time2soes = 0;
    osStatus_t timerStatus;
    uint32_t argument;

    /* Setup config hooks */
    static esc_cfg_t config =
    {
       //.user_arg = "/spi0/et1100",
       .user_arg = "LOCAL_SPI",
       .use_interrupt = 0,
       .set_defaults_hook = NULL,
       .watchdog_cnt = 1000,
       .pre_state_change_hook = NULL,
       .post_state_change_hook = post_state_change_hook,
       .application_hook = NULL,
       .safeoutput_override = NULL,
       .pre_object_download_hook = NULL,
       .post_object_download_hook = NULL,
       .rxpdo_override = NULL,
       .txpdo_override = NULL,
       .esc_hw_interrupt_enable = NULL,
       .esc_hw_interrupt_disable = NULL,
       .esc_hw_eep_handler = NULL
    };

    // This is the soes sm


    soes_step = s_start;

    while(1) {
        switch (soes_step) {
        /*--------------------------------------------------------*/
        //   Dummy state
        case s_start:
            //  entry:
```

```
            __NOP();
            //  exit:
            soes_step = s_init1;
            break;
    /*---------------------------------------------------------*/
    case  s_init1:
        //  entry:

        if (timerSOES != NULL) {
            // Timer not null might mean that it came from an strange state
            __NOP(); //Handle error
            soes_step = s_error;
            break;
        }
        //  Timer for the init state sm, needs to be null at the beginning
        argument = 1u;
        timerSOES = osTimerNew(timeoutSOESCallback, osTimerOnce, ↙
            &argument, NULL);
        if (timerSOES == NULL) { //Normal check-up of timer after creation
            __NOP(); //Handle error
            soes_step = s_error;
            break;
        }

        timerStatus = osTimerStart(timerSOES, 1000u);
        if (timerStatus != osOK) {
            __NOP();    //Handle error
            soes_step = s_error;
            break;
        }

        ecat_slv_init (&config);

        //  exit:
        if(osTimerIsRunning(timerSOES)) {
            timerStatus = osTimerStop(timerSOES);
            timerStatus = osTimerDelete(timerSOES);
            if (timerStatus != osOK) {
                __NOP();   //Handle error
                soes_step = s_error;
                break;
            }
        }
        if (soesTimeoutFlag) {   // soes loop left by timeout
            // Handle error
            soes_step = s_error;
            break;
        }
        soes_step = s_init2;
        break;
    /*---------------------------------------------------------*/
    case  s_init2:
        //  entry:
```

```c
            osEventFlagsSet(evt_sysSignals, ECAT_EVENT|EV_ECAT_ESC_INIT);    ✓
                //TODO << Check with heap observer that two flags are set
            osThreadSuspend(ecatSOESTHandler);    // << Resumed by Ecat SM in ✓
                State: Connected. This could be an event
            //  exit:
            argument = 2u;
            timerSOES = osTimerNew(timeoutSOESCallback, osTimerOnce, ✓
                &argument, NULL);
            if (timerSOES == NULL) {
                __NOP(); //Handle error
                soes_step = s_error;
                break;
            }
            //  Starting soes app timing
            time2soes = osKernelGetTickCount();  //PENDING This variable could ✓
                be used for improved refresh cycle control
            soes_step = s_timerset;
            break;
        /*--------------------------------------------------------*/
        case  s_timerset:
            //  entry:
            timerStatus = osTimerStart(timerSOES, 1000u);
            if(timerStatus != osOK) {
                __NOP(); //Handle error
                soes_step = s_error;
                break;
            }
            heapObserver1 = timerSOES;

            //  exit:
            soes_step = s_slaveloop;
            break;
        /*--------------------------------------------------------*/
        case  s_slaveloop:
            //  entry:
            ecat_slv();
            //  exit:
            if(osTimerIsRunning(timerSOES)) {
                timerStatus = osTimerStop(timerSOES);
                if (timerStatus != osOK) {
                    __NOP();    //Handle error
                    soes_step = s_error;
                    break;
                }
            }
            if (soesTimeoutFlag) {   // soes loop left by timeout
                // Handle error
                soes_step = s_error;
                break;
            }
            soes_step = s_sleep;
            break;
        /*--------------------------------------------------------*/
```

```
        case  s_sleep:
            //  entry:
            osDelay(SOES_REFRESH_CYCLE);
            // A better refresh cycle control could be achieved by using ⟋
                osDelayUntil();
            //  exit:
            if (soesTimeoutFlag) {   // soes loop left by timeout
                // Handle error
                soes_step = s_error;
                break;
            }
            soes_step = s_timerset;
            break;
        /*------------------------------------------------------*/
        case  s_error:
            __NOP();  // Handle the error
            timerStatus = osTimerDelete(timerSOES);
            if (timerStatus != osOK) {
                __NOP(); //Handle error
            }
            //osDelay(100); //TEST
            osThreadSuspend(ecatSOESTHandler); // this should wait for event ⟋
                handler or something to restart
            break;
        /*------------------------------------------------------*/
        default:
            soes_step = s_error;
            //soesTimeoutFlag = FALSE;
        } // End switch
    }  // End while
}
```

■ Listing E.5: Main source code for SOES APP DSM.

```
/*
* smLed.c
*
*  Created on: Jun 25, 2020
*      Author: Carlos Reyes
*/

#include "SMs.h"
#include "smLed.h"
//#include "WS2812_Lib_MultiChannel.h"

osTimerId_t refreshLed,timeoutLed;  //Pending: this could be local or static
static volatile uint8_t boolTimeoutLed,boolRefreshTimeoutLed;  //PEnding is ⟋
    this necessary?
// Debug variables
volatile uint32_t currentFlags1,currentFlags2;

//External variables
```

```
extern TIM_HandleTypeDef *ledCH1,*ledCH2,*ledCH3,*ledCH4;   // Declared in
    WS2812 Libraries


/*
 * @brief Sate Machine for overall task of LED RINGS controlled by PWM
 *
 */

void ledRings_SM (void * argument) {
    uint8_t chsetupOK[NUM_OF_LEDRINGS];
    uint8_t error = 0;
    uint32_t temp32,eventStatus;
    osStatus_t timerStatus;


    timeoutLed = osTimerNew(timeoutCallback_led, osTimerOnce, NULL, NULL);
    refreshLed = osTimerNew(refreshCallback_led, osTimerOnce, NULL, NULL);
    if (timeoutLed == NULL) {
        __NOP();  //Debug the error.
    }

    while(1) {

        switch (led_step) {
        /*--------------------------------------------------------*/
            case  L_config:     //Initializes and links the handlers with
                WS2812 library
                //   action
                if (NUM_OF_LEDRINGS > 0) {
                    if(ledDMA_configCh(1,&htim8) != FAILED)
                        chsetupOK[0] = TRUE;
                    else {
                        chsetupOK[0] = FALSE;
                        error++;
                    }
                }
                if (NUM_OF_LEDRINGS > 1) {
                    if(ledDMA_configCh(2,&htim3) != FAILED)
                        chsetupOK[1] = TRUE;
                    else {
                        chsetupOK[1] = FALSE;
                        error++;
                    }
                }
                //if (NUM_OF_LEDRINGS > 2)
                //if (NUM_OF_LEDRINGS > 3)


                //   exit
                if (error) {
                    notifyError(ERR_LED_INIT);  //Pending This should notify
                        over ECAT but not stop the overall SM
                    led_step = L_restart;
```

```
            }
            else {
                error = 0;
                //Set the Effects
                setColorState(color_preop);
                temp32 = SYS_EVENT|(EV_LED_DSM_INIT<<SHIFT_OFFSET);
                osEventFlagsSet(evt_sysSignals, temp32);   // System ↙
                    notification
                led_step = L_send;
            }

            break;
/*-------------------------------------------------------------*/
    case L_send:
        //   action
        for (uint8_t i = 1; i <= NUM_OF_LEDRINGS; i++) {
            if (ledDMA_send(i) == FAILED)
                notifyError(ERR_LED_SEND);

        }
        timerStatus = osTimerStart(timeoutLed, (uint32_t) 1000U);  ↙
            //Timeout for DMA
        if (timerStatus != osOK) {
            notifyError(ERR_LED_OSTIM); // This is a internal OS error.
        }

        //exit
        led_step = L_waitEvent;
        break;
/*-------------------------------------------------------------*/
    case  L_waitEvent:
        //   action
        osEventFlagsWait(evt_sysSignals, LED_EVENT, osFlagsWaitAny, ↙
            osWaitForever);

        //   exit

        if (boolTimeoutLed) {
            if (osTimerIsRunning(timeoutLed)){
                if (osTimerStop(timeoutLed) != osOK) {
                    __NOP();//Handle internal OS  error
                    notifyError(ERR_LED_OSTIM);
                }
            }

            boolTimeoutLed = FALSE;
            notifyError(ERR_LED_TIMEOUT);
            led_step = L_restart;
            break;
        }
        else if(dmaLed1_rcvd && dmaLed2_rcvd) { //  SAFE: Only ↙
            updates a color state if no timeout
            if (osTimerIsRunning(timeoutLed)) {
```

```c
                        if (osTimerStop(timeoutLed) != osOK) {
                            __NOP();//Handle internal OS error
                            notifyError(ERR_LED_OSTIM);
                        }
                    }

                    dmaLed1_rcvd = 0;
                    dmaLed2_rcvd = 0;

                    led_step = L_updateColorState;
                    break;
                }

                break;
        /*-----------------------------------------------------------*/
            case  L_updateColorState:
                //   action
                if (errorFlag) {
                    setColorState(color_error);
                }
                else if (initFlag) {
                    __NOP();  // Keeps the default color
                }
                else if (ecatCMDFlag) {
                    setColorState(color_custom);
                }
                else if (warningFlag) {
                    setColorState(color_warning);
                }
                else if (normalFlag) {
                    setColorState(color_normal);
                }

                //   exit
                //notifyEvent(LED_UPDATED);
                led_step = EFFECTS_ACTIVATED ? L_updateEffect : l_waitRefresh;

                break;
        /*-----------------------------------------------------------*/
            case  L_updateEffect:
                //   action
//          if (led_effectRateUpdt()) {   This function checks whether the
    current effect needs to be updated
//              __NOP(); //PENDING Effects are a future future
//          }

                //   exit
                led_step = l_waitRefresh;
                break;
        /*-----------------------------------------------------------*/
            case  l_waitRefresh:
                // action
```

```c
            if(osTimerStart(refreshLed, (uint32_t)PWM_REFRESH_PERIOD)!= ↙
                osOK) {
                __NOP(); //Handle the OS TIMER starting error.
                notifyError(ERR_LED_OSTIM);
                led_step = L_restart;
                break;
            }
            //
            eventStatus = osEventFlagsGet(evt_sysSignals);
            osEventFlagsWait(evt_sysSignals, LED_EVENT, osFlagsWaitAny, ↙
                osWaitForever);

            //exit

            //Refreshing time is already elapsed
            if (boolRefreshTimeoutLed) {
                if(osTimerIsRunning(refreshLed)){
                    if(osTimerStop(refreshLed)!= osOK) {
                        __NOP();   //Only for error debugging
                        notifyError(ERR_LED_OSTIM);
                        led_step = L_restart;
                    }
                }

                boolRefreshTimeoutLed = FALSE;
                led_step = L_send;
            }

            break;
    /*---------------------------------------------------------*/
        case  L_restart:     //After timeout or error

            if (osTimerIsRunning(timeoutLed))
                timerStatus = osTimerStop(timeoutLed);

            if (timerStatus != osOK) {
                __NOP(); //PENDING Handle the deletion error
            }

            for (uint8_t i = 1; i<=NUM_OF_LEDRINGS; i++) {
                ledDMA_restartCH(i);
            }

            //exit
            led_step = L_config;

            break;
    /*---------------------------------------------------------*/
        default:
            __NOP();
        }
    }
```

```
        //osThreadTerminate(ledRingsTHandle); //If at any moment the cp reaches ↙
            out of the while loop


    }
```

```
/*
 * owApp.c
 *
 *  Created on: Aug 19, 2020
 *      Author: Carlos Reyes
 *      Based on owApp.c from
 *          Author:         Tilen MAJERLE <tilen@majerle.eu>
 *          Version:        v3.0.0
 */

#include "AxisCommHub_definitions.h"
#include "cmsis_os.h"
#include "lwow.h"
#include "devices/lwow_device_ds18x20.h"
#include "scan_devices.h"
//#include "stdio.h"


//Creating a new one-wire instance

extern const lwow_ll_drv_t lwow_ll_drv_stm32_hal;
lwow_t ow_inst;
lwow_rom_t rom_ids[20];    //ROM IDs are stored here
size_t rom_found;

//Definition from MAIN
extern UART_HandleTypeDef huart3;

//Definition from SM
extern int16_t gv_temperatureData[NUM_OF_SENSORS];

//Functions definition
static void owApp(void* arg);

const osThreadAttr_t oneWireTask_attr = {
        .priority = osPriorityAboveNormal1,
        .stack_size = 512
};


/*
 *    @brief Simple function to spawn the application thread
 *
 */
void initOwApp(void) {
    ow_inst.arg = &huart3;
```

```
    osThreadNew(owApp, NULL, &oneWireTask_attr);
}

/**
 * @brief          Application thread
 * @param[in]      arg: Thread argument
 */
static void owApp(void* arg) {
    float avg_temp;
    size_t avg_temp_count;

    /* Initialize 1-Wire library and set user argument to NULL */
    lwow_init(&ow_inst, &lwow_ll_drv_stm32_hal, &huart3);

    /* Get onewire devices connected on 1-wire port */
    do {
        if (scan_onewire_devices(&ow_inst, rom_ids, LWOW_ARRAYSIZE(rom_ids), ↙
            &rom_found) == lwowOK) {
            printf("Devices scanned, found %d devices!\r\n", (int)rom_found);
        } else {
            printf("Device scan error\r\n");
        }
        if (rom_found == 0) {
            osDelay(1000);
        }
    } while (rom_found == 0);

    if (rom_found > 0) {
        /* Infinite loop */
        while (1) {
            printf("Start temperature conversion\r\n");
            lwow_ds18x20_start(&ow_inst, NULL);      /* Start conversion on ↙
                all devices, use protected API */
            osDelay(1000);                          /* Release thread for 1 ↙
                second */

            /* Read temperature on all devices */
            avg_temp = 0;
            avg_temp_count = 0;
            for (size_t i = 0; i < rom_found; i++) {
                if (lwow_ds18x20_is_b(&ow_inst, &rom_ids[i])) {
                    float temp;
                    uint8_t resolution = ↙
                        lwow_ds18x20_get_resolution(&ow_inst, &rom_ids[i]);
                    if (lwow_ds18x20_read(&ow_inst, &rom_ids[i], &temp)) {
                        printf("Sensor %02u temperature is %d.%d degrees (%u ↙
                            bits resolution)\r\n",
                            (unsigned)i, (int)temp, (int)((temp * 1000.0f) - ↙
                                (((int)temp) * 1000)), (unsigned)resolution);
                        gv_temperatureData[i] = (uint16_t) temp;
                        avg_temp += temp;
                        avg_temp_count++;
                    } else {
```

```
                        printf("Could not read temperature on sensor %u\r\n", ↙
                            (unsigned)i);
                }
            }
        }
        if (avg_temp_count > 0) {
            avg_temp = avg_temp / avg_temp_count;
        }
        printf("Average temperature: %d.%d degrees\r\n", (int)avg_temp, ↙
            (int)((avg_temp * 100.0f) - ((int)avg_temp) * 100));
    }
}
printf("Terminating application thread\r\n");
osThreadExit();
}
```

🔲 **Listing E.7:** Main code for Temperature App.