

Research Project and Seminar

Information and Communication Systems

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

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

Acknowledgment

This is the place to thank all the people involved with your thesis / project. Examples would be your family, friends, and of course your supervisor. The acknowledgement will not have any influence on your grade; however, we think it is good style to have an acknowledgement in your thesis.

Abstract

The abstract of your thesis goes here. There may be formal requirements on it that can be found in the corresponding examination guidelines (Prüfungsordnung). If there are none, ask your supervisor. As a rule of thumb, the abstract should be concise and focused. It is not a shortened introduction to your work. We also suggest that—if an abstract is not required—only write one if it is really well done.

Table of Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | The need of RT within industrial environments | 1 |
| 1.2 | Industrial standards and the TSN initiative | 4 |
| 2 | State of the art | 7 |
| 2.1 | Current applications | 7 |
| 2.2 | An overview about the RT capable SW in robotics | 8 |
| 2.3 | Approaching openness within the RT protocols | 10 |
| 2.3.1 | EtherCAT protocol | 12 |
| 3 | Solution proposal | 15 |
| 3.1 | Technical specifications | 15 |
| 3.2 | Available Hardware | 15 |
| 3.3 | PCB proposal | 18 |
| 3.4 | Software structure | 18 |
| 4 | Implementation | 19 |
| 4.1 | LED Control | 19 |
| 4.2 | Temperature acquisition | 20 |
| 4.3 | EtherCAT Slave communication: SOES adaptation | 21 |
| 4.3.1 | EtherCAT data consistency and constraints for design | 21 |
| 4.3.2 | SOES | 23 |
| 4.3.3 | EtherCAT Slave Controller (ESC): LAN9252 | 25 |
| 4.3.4 | Development | 26 |
| 4.4 | Device State Machines (DSMs) | 26 |
| 4.4.1 | Scheduling | 27 |
| 4.5 | PCB | 28 |
| 5 | Results | 33 |
| 5.1 | PCB, SPI and LED* | 33 |
| 5.2 | DSMs and RTOS scheduling | 35 |
| 5.3 | SOES | 37 |
| 6 | Conclusions | 39 |
| | Bibliography | 41 |
| A | PCB drawings and layout | 45 |
| B | Source codes | 49 |

TABLE OF CONTENTS

Introduction

**Redaction style at 95/100

This document describes the different stages through the development of an embedded communication hub for sensor data acquisition in a robotic system, within this document this prototype will be referred as *Axis Communication Hub* or ACB. During this 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. 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. Finally in this chapter, a brief comparison of the openness of these protocols and how this is related to the development of devices is presented. 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. This has focus on EtherCAT devices as it is within the scope of this Research Project and shows advantages that will be detailed as the reader reads through this document. Afterwards, the third chapter deals with the goal of the Research Project and its proposed solution. a summary of technical specifications, the hardware available, software structure and an overall prioritization of the goals is also included. Later on, during the fourth chapter, the main points related to the implementation is presented. The overall results are discussed in chapter five, where the reader can find comments about the implementation and test challenges. As part of the conclusions chapter, a proposed list of points for further development is discussed. Finally, extra information focused on the technical details of the implementation can be found within the appendixes.

1.1 The need of RT within industrial environments

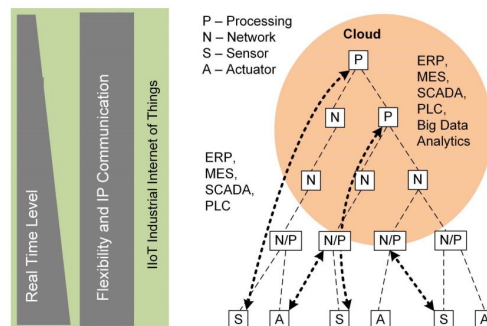
During the last years an increase in the usage of the Ethernet-based fieldbuses within industry has been recorded. This, with no surprises, shows the expected adaptation of the industrial

1 INTRODUCTION

automation to the IT infrastructure, which is fundamental for the *Industrie 4.0* paradigm and its consequent huge amount of data to be monitored, analyzed and controlled, dealing at the same time with different time constraints and interconnectivity among the different layers of an industrial system and their devices. Having in mind the current *automation pyramid* with direct access from the top to the bottom[SKJ18], see figure 1.1, it is understandable that several technologies providing this access have been meeting each other coming either from the top or the down levels, at the point that they offer similar features regarding data access and security. Each of them with their own development history, alliances and, therefore, standards. 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 fieldbus side -lowest level-. All of them have developed in an individual way as response of market, 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 where information, technical as well, and development tools have become even more available and open to the end-user. Leading now then to a situation where the private initiatives are not any longer the full owners of the technology development.

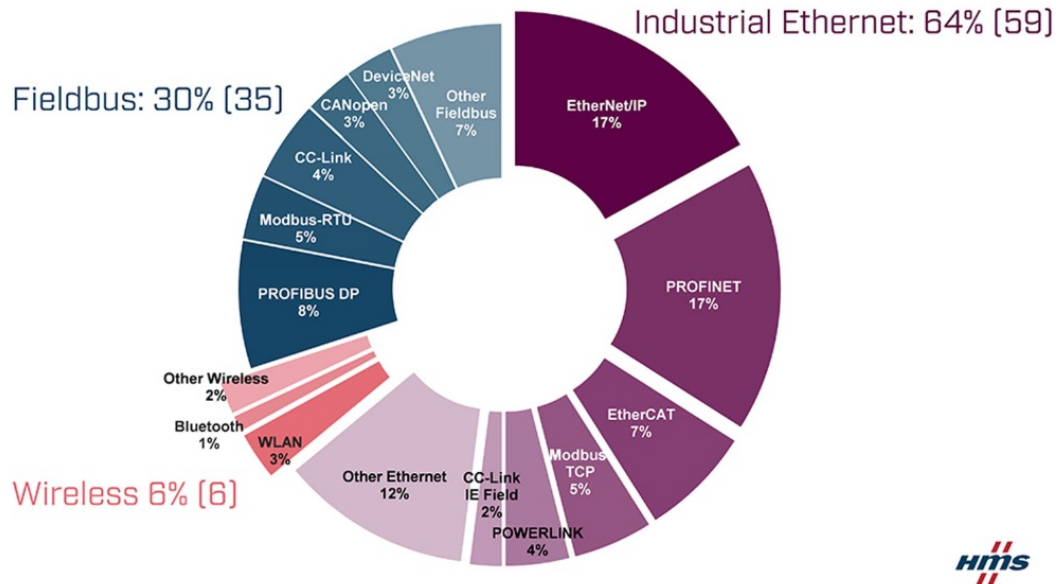
Another line of work, closely related to interoperability, is the Real Time (RT) applications in their both versions with *hard* and *soft* requirements. Nowadays, there is an increasing number of applications in robotics that demand control loops and device chains that demand hard real time performance. Although, this requirements are more typical at the device level, such as, robots, cncs, etc. They all now face the IIoT requirements; 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 these sort of features are increasingly being demanded as well at upper levels IT levels.

The current automation industry has a record of many competitors and closetechnologies, as natural consequence for specific processes requirements (depending on the industry), but also



■ **Figure 1.1:** *Industrie 4.0* architecture. Industrial Internet of Things.[SKJ18]

1.1 THE NEED OF RT WITHIN INDUSTRIAL ENVIRONMENTS



■ **Figure 1.2:** Industrial network market shares 2020 according to HMS Networks.[Car20]

as a response of market strategies. Nevertheless, the search for standardization can be tracked back to the 80s, as the fieldbuses were standardized by the International Electrotechnical Commission (IEC). Continuing after the 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%; as the commonly called fieldbuses decreased in the same period from 42% to only 30%; finally the industrial wireless remained around the 6%, see figure 1.2. [Car20]

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 -as they have modified Media Access Control (MAC) Layers-. More details about this differences will be addressed in the following chapters.

As history shows, vendor protected technology have 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 -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 chapter 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. If the reader has already good knowledge of these standards, this section can be easily skipped and continue with the next chapter. First of all, due to the historical and technological process of innovation within the information and communication systems, several parties have been related and, at some extent 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 fieldbuses, 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 100Mbps. 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 and 400 Gbps.[ISO17] 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 fieldbus standard published in 1999, where 8 Types of fieldbuses 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 fieldbus war.[Fel02] Its most updated version in 2019 counts with 26 Types of protocols and grouped them as fieldbus Communication Profile Families (CPF).

IEC 61784-Part 2:2008 It is an extension for the CPs capable of RT that are based on the IEEE 8802-3 standard (commonly named Ethernet). Commercial names included are the next: EtherCAT, Profinet, Ethernet/IP, Ethernet Powerlink, and Modbus TCP.[VZS19] The SERCOS CPF is highlighted, since its third version is, altogether with the EtherCAT 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.[Sta00][Sta20] This is a

1.2 INDUSTRIAL STANDARDS AND THE TSN INITIATIVE

very interesting Hard Real Time capable protocol that might need further development which is not included in 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. [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 on-going project and still around 40% of its standards are in draft or preparation phase.[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.[II20] This is an on-going 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 fieldbuses.

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.[IEC20] Quoting: *OPC UA is a client-server communication protocol for industrial use cases without hard realtime requirements. The new PubSub extension of OPC UA adds the possibility of many-to-many communication based on the Publish / Subscribe paradigm. In conjunction with the upcoming Time-Sensitive Networking (TSN) extensions of Ethernet, OPC UA PubSub aims to also cover time-deterministic connectivity.*[PERK18]

1 INTRODUCTION

State of the art

**Redaction style at 95/100

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 capable 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 the 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. Finally, an overview of the EtherCAT protocol is included.

2.1 Current applications

As rapidly mentioned in 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 Sercos 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 a 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 Real Time control loop for Servo Motors, which run CAN devices over EtherCAT (CoE service). The implementation of the Master device ran on different open source RT OS based on Linux, namely, Xenomai and Linux with the *RT_PREEMPT*. It was concluded that both of the approaches were capable to achieve update periods of $1ms$, and Jitter around 1.15 microseconds. Moreover, Xenomai could average execution times around 100

microseconds[DC17]. It is worth to mention that the Master device executed on top of both kernels, the IgH EtherCAT Master, the latter mentioned in 2.3.

The characterization and optimization of performance of different RTE profiles within TSN is the current topic as the TSN standards and the RTE commissions are still working together, as briefly mentioned in 1.2. In [MGSR20] are presented simulations of TSN topologies with EtherCAT and SERCOS data frames, where the QoS is addressed and evaluated through the usage of SDN (Software-defined Network) for TSN switches. The approach of this project is to test different scheduling features given fixed Cycle Times of the Data frames, which were proposed to be similar to the current 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 is still open.

Besides robotics, a recent industrial application concerning distributed control and monitoring based on EtherCAT open protocol can be seen in this article [Nor20].

Addressing now the usage of open source tools, 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 hydraulical 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 Hz. 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[Han20].

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 the previous section was being written, several resources and examples showed the current usage of RT-oriented 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

2.2 AN OVERVIEW ABOUT THE RT CAPABLE SW IN ROBOTICS

applications, it was worth adding this section to summarize the RT Software for development in robotics.

As mentioned in [DC17] and [MHF⁺20], the usage of middlewares within the field of robotics is growing and it relies on robot software that exists between the application and a RT capable OS. In [JYJP20] a list of requirements is provided to address these middlewares and how to consider them Real Time Robot Software Platforms. The mentioned list can be interesting to start getting familiar with the capabilities and features of the so-called Robot Software. The list of requirements is as follows:

- R1: Support data exchange
- R2: Real time support (strict period execution and sporadic performance support)
- R3: Supports thread and process types for user defined programs
- R4: Easy configuration of applications (robot control SW, PLC SW, vision inspection SW, non-real-time SW, etc.)
- R5: Support multiple periods.
- R6: Threads or processes running in the same period are classified by priority.
- R7: Check and handle the event through the event handler.

Common names for different projects aiming to create this development frameworks are the following: Common Object Request Broker Architecture (CORBA), Real-Time CORBA (RT-CORBA), Data Distribution Service (DDS), OPC Unified Architecture (OPC-UA), Robot Operating System (ROS), Open Platform for Robotic Services (OPRoS) OpenRTM (open robotics technology middleware), open robot control software (OROCOS), XbotCore, and real-time middleware for industrial automation devices (RTMIA). Further comments and comparison 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[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 hardt RT requirements. Orocos has further improved its compability, 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 roughly 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 has been already mentioned in 2.1, which have reached control loops down to 1khz for 33 axes [MLH⁺17].

RTMIA RTMIA middleware + Linux or Xenomai but used open PLC running parallely[?] « This needs further reading [?]

The previous information was presented only to draw an idea for the non-familiar reader about the applications and, since this topic is in on going development and, furthermore, many other plataforms are addressing similar challanges are arising; 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

2.3 APPROACHING OPENNESS WITHIN THE RT PROTOCOLS

specification is of huge importance, a set of enhancements for the *open62541* project were developed by *Fraunhofer IOSB* and a serie of patches for the Linux kernel has been released to make it a RT compatible.[[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*. [[RT-20](#)]

Sercos Stacks Sercos III technology is able to be operated in a common TSN-based network [[Ser18](#)], since its development group is working closing together with the TSN group. 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 later even allows the host to use any standard Ethernet controller.[[Ser20](#)] It is important to mention that there are testing tools to certificate those devices and achive a Safety Integrity Level 3 (SIL-3).

EtherNet/IP Stacks There are several commercial stacks that comply with the Open DeviceNet 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. Examples are provided for integration only in Linux and Windows in [[OpE20](#)]; nevertheless, a variation for embedded systems has been presented for an STM32 microcontroller. 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.[[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 indsutrial 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 [-2.1-](#); it makes sense to present an introduction to the protocol itself. This way it is easier to go sensibly to the implementation of what is one of the basic chain-elemnts in what could become a very complex application: an EtherCAT Slave device with open source elements.

2.3.1 EherCAT protocol

As mentioned in the introduction, EtherCAT is an industrial Communication Profile developed by Beckhoff 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 field of smart cities[?], in a certain degree this approach eliminates the need for many expensive "black boxes"[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.

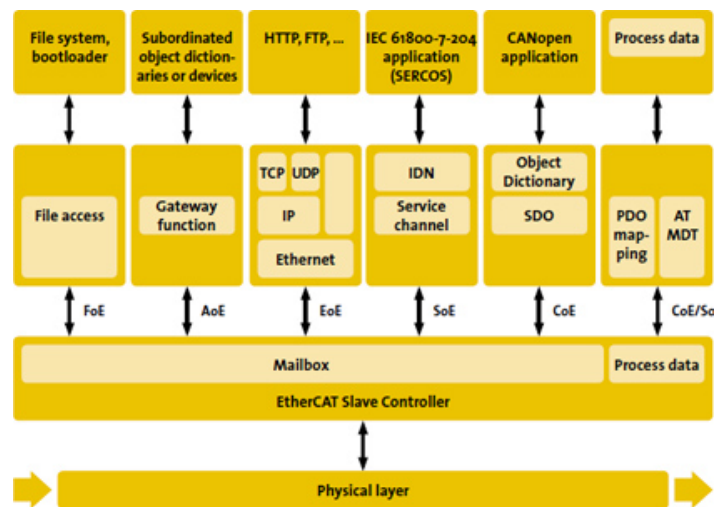
The OSADL emphasized in 2008 [Bor08], for example, a vision for leading the integration of opensource in the industry by using the Linux Kernel as a certified Industrial RT (IRT) operative system for industrial embedded applications. Back in that day, Beckhoff was involved in that discussion representing the contrary model. Nonetheless, in the last months the same company has apparently retaken the opensource initiative by the introduction of the FreeBSD compatible version of the TwinCAT Runtime[Bec20b].

In comparison with other RTE profiles, EtherCAT has shown a higher performance, more flexible topology and lower costs than other ethernet fieldbus technologies. This protocol applies a master-slave mode, in which the master device uses standard 100BASE-TX ethernet adapter and the ESC (EtherCAT Slave Controller), that implements a EtherCAT IP (intellectual property) core within an ASIC or a FPGA to process the frames. As the working cycle starts, the EtherCAT 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 EtherCAT is fixed to 100 Mbit/s with full duplex communication. The EtherCAT network is able to connect maximally 65535 devices via switch and media converter. The EtherCAT system can update 1000 I/Os in just 30 microseconds or exchange 1486 byte contents in 300 microsecond. [Bec17][XJY11]

Important to highlight is that other CPs are also integrated as services inside the protocol, as previously mentioned for the SERCOS specification in 2.1. Other examples of these integrated CPs are File over EtherCAT (FoE) or Ethernet over EtherCAT (EoE), which make possible to support a wide variety of devices and application layers in the same network[Bec20a]. A complete list of the communication profiles that are on hand through the protocol's mailboxing is given below, as to the overall layered integration can be seen in figure 2.1.

2.3 APPROACHING OPENNESS WITHIN THE RT PROTOCOLS

- 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 2.1:** Different communication profiles can coexist in the same system.

An EtherCAT device with switchport 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 EtherCAT 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.[Bec20a]

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. Consequent certification process should adhere to Beckhoff's specifications.[Bec18]

Having presented this brief summary of the EtherCAT technology, the reader may continue to the following chapters. More detailed information of the protocol itself that was needed to understand the function of the SOES library can be read in the section 4.3.

2 STATE OF THE ART

Solution proposal

****Redaction style at 90/100**

This chapter is intended to give the reader a summary of the proposed solution and its structure, as well as the technical specifications that were given and taken into account. 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 Technical specifications

In table 3.1 the requirements for this prototype 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 sections.

Regarding the safety feature 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.

****Add table in solution proposal with the speeds**

3.2 Available Hardware

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 considered since the overall characteristics are somewhat similar and generic -regarding peripherals like serial interfaces or dma, processing power and memory, for instance-. MCUs from Infineon [Inf18][Gro04] and Texas Instruments [Son20] were good possible candidates; however, the basic familiarity with the STM32CUBE IDE and the related ST technology was a crucial factor, since the learning curve is not negligible when

3 SOLUTION PROPOSAL

| Feature | Requirement | Implementation | Remark |
|-----------------------|---|---|---|
| Upper layer interface | Ethernet/EtherCAT compatibility. Non-safety relevant. Services and synchronization: tbd | 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: tbd | Attachable PCB for LAN9252-EVB-SPI. Size: 55mmx38mm | ✓ Second layout with both chips included possible in the short term. |
| Safety | n.a. | Non Safety-Critical for this prototype | – SFoE 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 | ✓ |
| Speed/position | Possible interface of BISS-C type | Not required for this prototype | – |
| Refresh data cycle | tbd | No hard RT deadlines. Deterministic refresh cycle of ~ 10ms by RTOS. Timeout faults handling. | ⊕ |
| Data structure | tbd | Functional and parametrization data structure as Object Dictionary. Standard ESI file. | ✓ |
| FW programming | n.a. | CMSIS - FreeRTOS for thread, event and time management. | ⊕ |

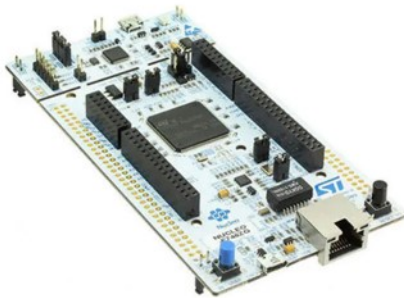
■ **Table 3.1:** Technical specifications

| 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 TQFPPEP packaging |

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

it comes to develop any firmware at a fair level, even more when it deals with other to-learn* technologies, for instance, development with RTOS, modification of open libraries, EtherCAT protocol and the Network Controller chip.

Regarding the Network Controller, the LAN9252 belongs to a set of ASICs that are verified and certified by Beckhoff GmbH. For a further reference for other alternatives visit [Gmb20]. 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 main characteristics of the above mentioned hardware.



(a) NUCLEO-STM32F446ZE



(b) LAN9252-EVB-SPI

■ **Figure 3.1:** Evaluation boards for prototyping.

3.3 PCB proposal

As it can be seen in the Fig. 3.1(b), the evaluation board counts with on-board male pins, this was taken as an advantage and the PCB to be designed consisted on a plugable 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 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.

3.4 Software structure

**Redaction style at 80/100

The final proposal for the structure of the embedded system can be seen in fig. 3.2. The functional blocks represented in grey 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. Another thing to keep in mind is the abbreviation DSM (Device's State Machine) which is used in this document as substitute for State Machine, such that it is not confused with Synchronization Manager, defined as well by Beckhoff as SM.

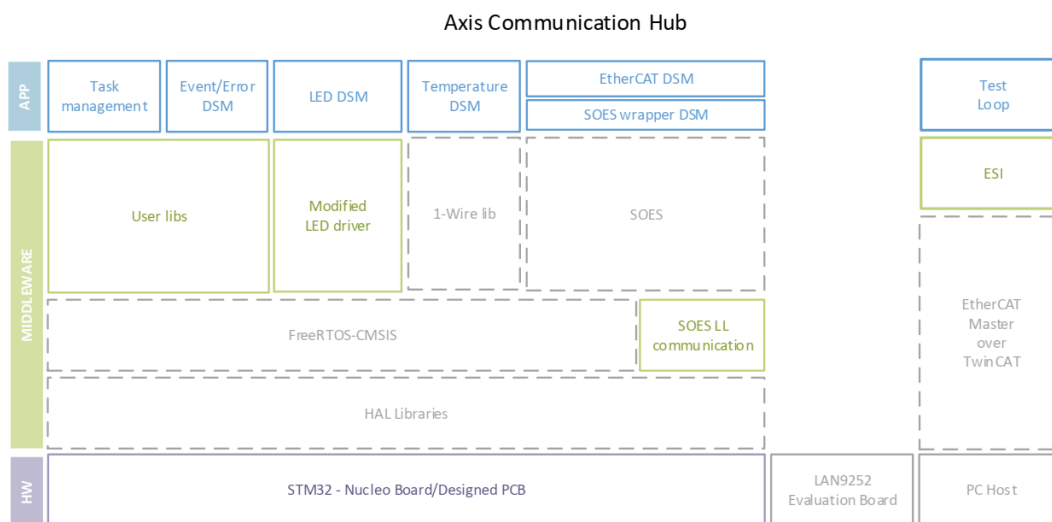


Figure 3.2: Layered structure of the proposed functional blocks.

Implementation

*Redaction style at 70/100 The following chapter will document the different functional modules that were implemented according to the proposal. Most of them interact with each other through the SMs, which has been added as well as a module for its understanding.

4.1 LED Control

*Redaction style at 30/100

The robot counts" with at least two LED rings that notify to the user the overall state of each access, this is considered only a visual aid which means that is not critical for the system. Currently there is a control for them using Arduino* devices. One of the goals of this prototype is to integrate the control of the LED rings within a single board; therefore, different approaches were taken into account depending on the stages and experience in developing on STM32 chips and its IDEs. The following stages were carried out:

Here comes one picture of the LED ring and a very brief description of the LED and its interface, is it interface or protocol?-> interface

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

Writing code for one LED control Using the peripherals of the MCU, simple routines were written to set different basic colors in RGB. The peripherals used were a PWM channel and one timer to keep control of the timing.

Exploration of libraries Once the basic communication was understood, it was clear that the usage of libraries would be more practical since the control demanded configurable number of LEDs and updating all of them at once, even with the possibility of including effects. From this exploration around STM32, the codes were mainly divided in two types: libraries for 8/16 bits processors that whose main task was controlling the LEDs, and a pair of libraries that used DMA Module within 32-bit processor. The latter was

4 IMPLEMENTATION

divided into two different approaches, namely rewriting an unique buffer for only one PWM channel and another highly focused on performance by dividing a circular buffer and interrupting the main processor only to update parts of it as the data was sent.

First library selection Due to the unexperience working with DMA modules and as the LED control was not of a high priority as the EtherCAT implementation, it was decided to run first one of the basic libraries to achieve a multi LED control and look for its adequating to the Axis Communication Board*.

Trial of adequation of the library into the SM task *here comes the explanation of why that library did not match with a multitask approach*

Second library selection As the first library was able to control a set of 20 LEDs, some drawbacks were found in the approach, since the processor was focused only on polling Timers states. Event though, this first trial helped improving the overall understanding of the host MCU and the LEDs, it was not suitable to multi-tasking. Moreover, usage of DMA became clearer and it was decided to improve the approach by selecting a 32-bit processor based library. The Library* was then included and following the guidelines of the GNU* license it was modified to use two independant PWM generators -easily extendable* to four-. Furthermore, the main logic* of the library was divided such that its execution becomes non-sequential but triggered by events. The new interactions with other parts of the code can be seen in following subsections for SM*s.

4.2 Temperature acquisition

*Redaction style at 30/100

First readouts Within the project's schedule this was the first set of test, so through them the IDE stm32cubei was learnt, along with the general configuration of the MCU. First trials* included set up of internal clocks, interruptions, GPIO basic usage, basic configuration of freeRTOS in STM32 *here comes a reference to a future subsection where the structure and relation of freeRTOS and CMSIS is explained*; as well as the introduction to the one-wire protocol. The first approach, similar to the one of the LED, was to understand the sensor and its protocol. Therefore, self written functions were tested based on peripherals, namely two timers, through which the timing of the data streams are controled as wel as the duration of ones and zeros. This approach worked as intended but it was knew from the beginning that does not match the multi-tasking and that further was needed. This implementation was able only to access in a generic way to the temperature conversion, and further functions were needed to access the sensor's ROM needed e.g. for identification.

Readouts as a task/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 used 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 or blinking LEDs. *here could come a reference to the captures of signals and PWM being generated and interrupted*. 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 @9600 and @11200 bps suits 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 within the smt32cube development environment.

4.3 EtherCAT Slave communication: SOES adaptation

*Redaction style at 85/100

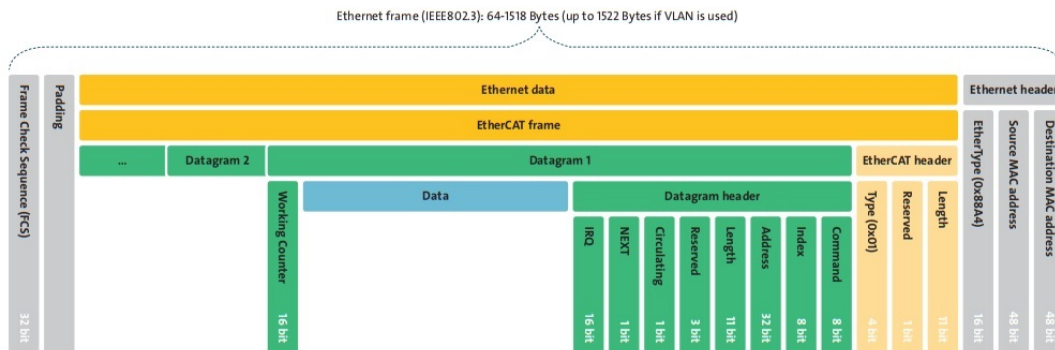
This functional module is the one with highest priority, therefore most of the effort given was focused not only on the library itself but the protocol and the hardware commissioning. Hence, this section was extended and divided as follows: first, more technical details are presented regarding the EtherCAT specification, such that the SOES features are better understood. As to the next subsection, some constraints for the prototype are commented and the available hardware is explained*. Once summarized, 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 itself. 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 from the integration of the SOES library. It is presented however before, such that it is understandable what features became priorities and which others are now presented as proposals.

4 IMPLEMENTATION

The reader may recall the set of Communication Profiles that are available within the EtherCAT fieldbus, see 2.3.1. From them, 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 bootload; whereas the latter would make the ACB* accessible for any IT tool based on TCP/IP.



■ **Figure 4.1:** EtherCAT Datagram within the Ethernet Frame. Source: ETG.1000.4 - EtherCAT frame structure.

The data frame and the Synchronization Managers

*Redaction style at 85/100

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 Clocs (DC) are need to be synchronized. [?][?]

There are three synchronization modes:

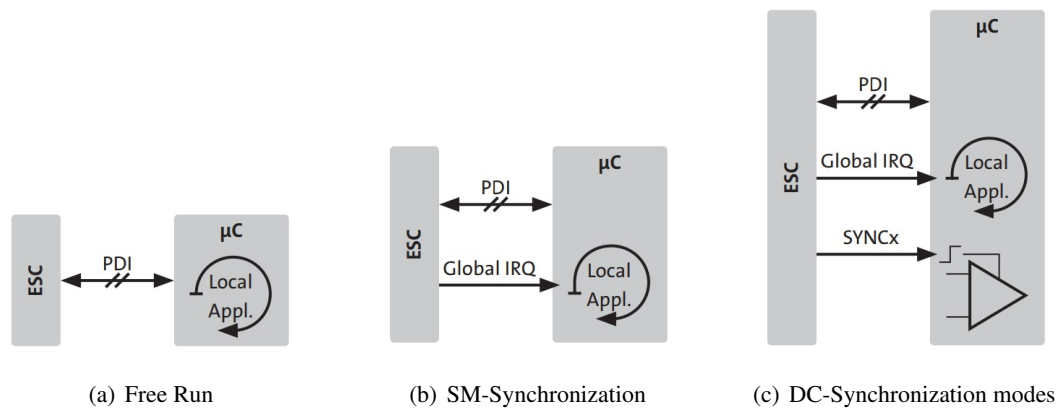
Free Run Application is triggered by local clock and runs independently from EtherCAT cycle.

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.

4.3 ETHERCAT SLAVE COMMUNICATION: SOES ADAPTATION

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

The scope of this prototype covers the Freerun and SM-Synchronous mode, as they are the basic ones for communication Master-Slave. A graphical representation of them is shown in figure 4.2.



■ **Figure 4.2:** Synchronization modes defined in ETG.1000. Source [?]

SMXs (Synchronization Managers 1,2,3 ...) coordinate access to the ESC memory from both sides, EtherCAT and Host MCU (PDI). In case of process data communication it ensures that process data can always be written to the memory by EtherCAT and can always be read by 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. [?] The mapping of the process data objects within the Ethernet Frame can be seen in figure 4.1 and 4.3. The correct setup of the SMXs 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.

4.3.2 SOES

*Redaction style at 90/100

As briefly commented in section 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

4 IMPLEMENTATION

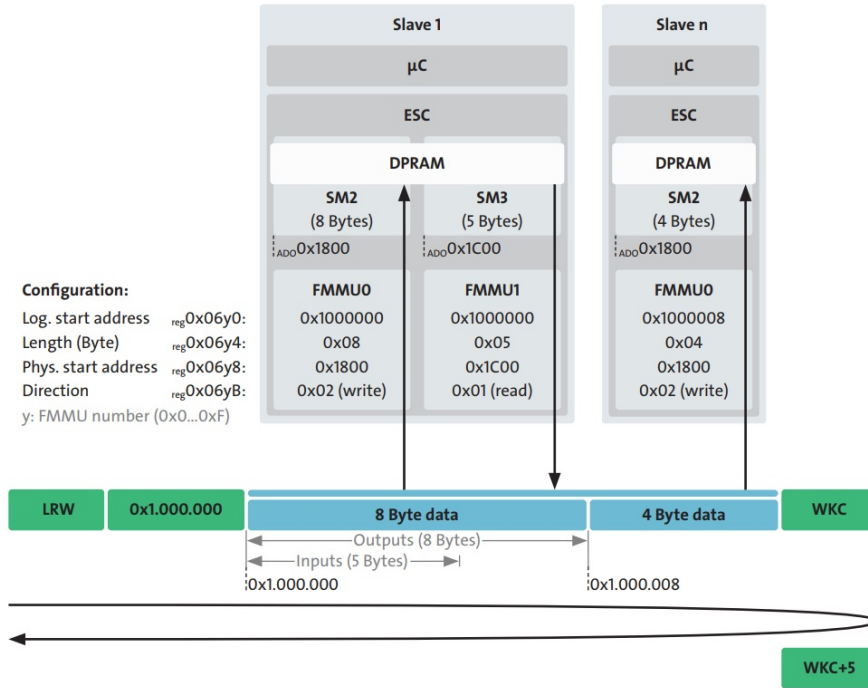


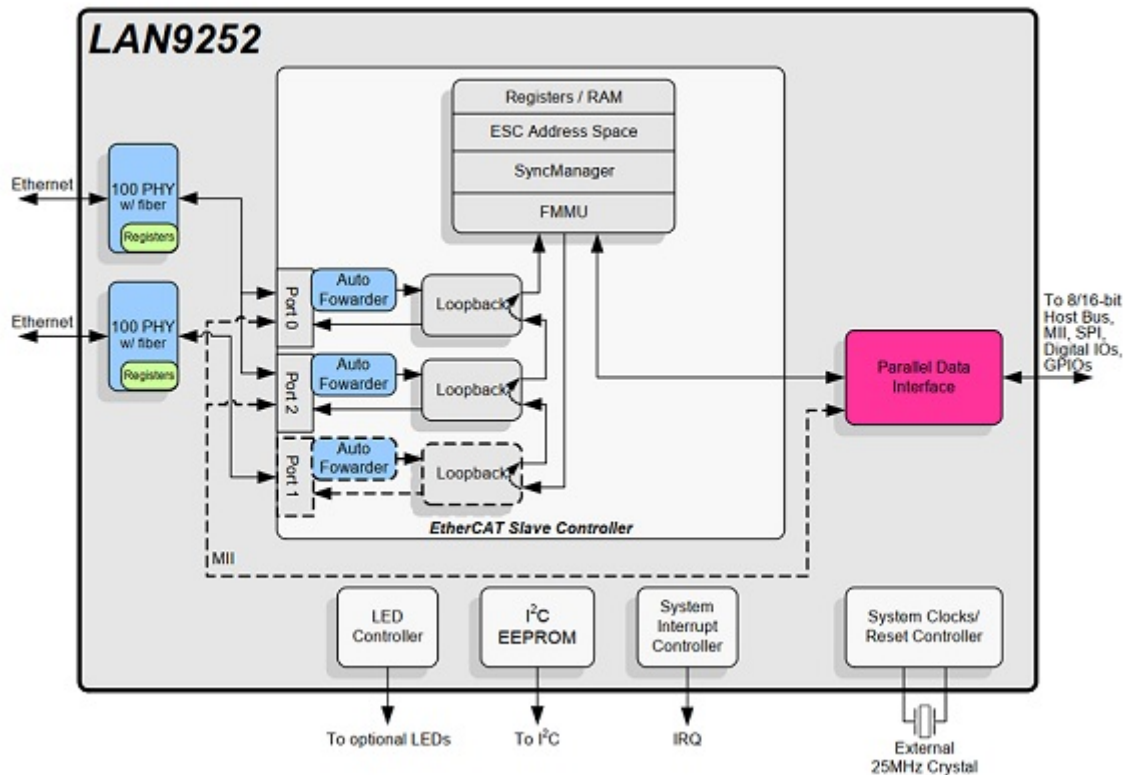
Figure 4.3: Depending on the different states of the Slave, there will be different data frames being exchanged with the Master. The above one corresponds to the Process Data Object which is updated continuously by the SM2/3 during Operation State. Reference: ETG.1000.6-SDO

| 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 > 64KB) |

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

Project Research was to achieve the EtherCAT Slave functionality in the prototype without those tools, as the protocols are open. In 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 ETHERCAT SLAVE COMMUNICATION: SOES ADAPTATION



■ **Figure 4.4:** Internal structure of the LAN9252, highlighting the PDI which was selected for this application to be SPI

4.3.3 EtherCAT Slave Controller (ESC): LAN9252

As part of the available hardware introduced in 3, the LAN9252-EVB-SPI is an evaluation kit for the ASIC LAN9252 manufactured by Microchip. This IC is an EtherCAT Slave Controller with 4K bytes 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 4 SyncManagers to allow the exchange of data between the EtherCAT master and the local application.[?]As briefly summarized in 4.3.1, each SMX 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 4.4.

4 IMPLEMENTATION

4.3.4 Development

*Redaction style at 70/100 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 the low-level functions of the library THIS

First tests with most basic read functions over spi THIS

Selecting the features to be implemented on MCU Host according to specification and complexity
THIS

Second tests with read/write functions for directly addressed registers THIS

Third tests with the EtherCAT Master At this stage a compliant EtherCAT Master was configured through a PC running *TwinCAT 3*. 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 (*ELMO Controller*) and an in-house multi-protocol end effector tool. For those different data structures were declared and very simplistic update loops were programmed within the XAE environment using *SText***** programming language.

Creation of an ESI file and flashing **Mention that the existing information is either for the Beckhoff's ET1100 or PIC32 (in the case of the LAN9252 set of APIs).

Object dictionary **Mention that according to the standard *mention the standard for ESI files** and object dictionary was created matching to the one contained in the ESI file, but mapped according to the few documentation available of SOES.

Fourth tests: running the flashed device Longer tests and configuration loop due to the deepening on EtherCAT protocol. Refer to the the SMs characteristics**** Constant comparisons between the data read by the Master *this could have another image from the mapping access through XAE* and the data received by the MCU host.

4.4 Device State Machines (DSMs)

*Redaction style at 85/100

In order to have a deterministic behaviour 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 implementation follows a case comparator approach, since it was simple, yet effective and flexible enough, to work during the prototype. This

4.4 DEVICE STATE MACHINES (DSMs)

characteristics was 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:

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 ??.

Temperature Initializes and runs the temperature related functions. It relies primarily on the open-source library. See in ??.

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 develop advanced, since the native infinite loop the SOES library is based on had to be adapted. The two involved DSMs can be seen in 4.5.

LED Initializes and updates periodically the RGB value of the LED Rings. See in ??.

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 dependant 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 action*. This flexibility was opted due to the inherent interconnection of the DSMs as they are not fully independant. To meet a suitable representation, the previopusly said is integrated with the approach of UPPAAL software to model timed automatas in , and the reader is invited to look it up in [?]. As a summary, since the modelling of automatas for real time systems demands a synchronization feature, this is represented and attached to the edges between locations (location in the sense that a state is a location constrained to valuated time and other variables) with the symbols ? and !, the one* acting as a wait action and the latter as a notification. Those, clearly can be understood as signals between different threads.

4.4.1 Scheduling

*Redaction style at 85/100

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

4 IMPLEMENTATION

| Thread | Release period (ms) |
|-----------------|---------------------|
| SOES APP SDM | 5 to 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

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 [?] and [?]. The time constraints are defined as *desired*, since the system is non Safety-Critical -recall the safety specification in 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 the results section 5.2.

In 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 annex header [?], and the individual functionality can be reviewed in the previous DSM section, 4.4.

A final list of priorities and threads is presented in the table 5.2 within the section of Results ??.

4.5 PCB

*Redaction style at 90/100

In this section straightforward information regarding the the manufacturing of the PCB proposal is given. The schematics and PCB layout can be consulted within the appendix A. The main idea around this design was introduced in 3.3 and has the purpose of providing experience in embedded hardware design and a base work for comming 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:

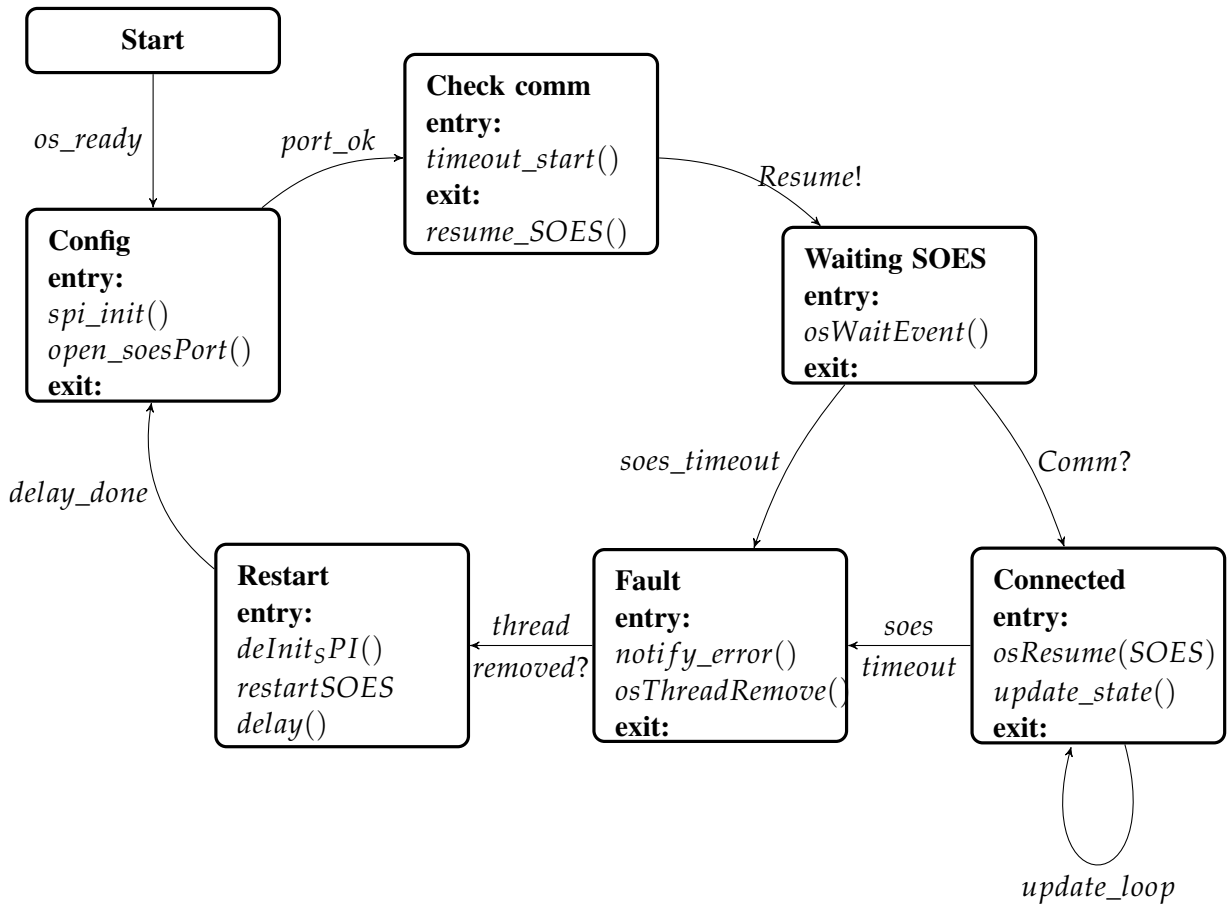
Design *Altium* PCB design software was used to prototype the PCB for this project. During the process the first approach was based on the open files for both the NUCLEO-F446ZE Development Board by ST [?] and LAN9252-EVB-SPI by Microchip[?]. By analyzing the general diagrams and selecting and adapting 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 and 3.3 V. To meet the routing needs a four layered PCB was selected. The final 3D model can be seen in figure 4.6

Manufacturing Due to practical reasons, the board manufacturing process was in charge of PCBWay, a PCB manufacturing company. In respect to soldering, it was made in-house.

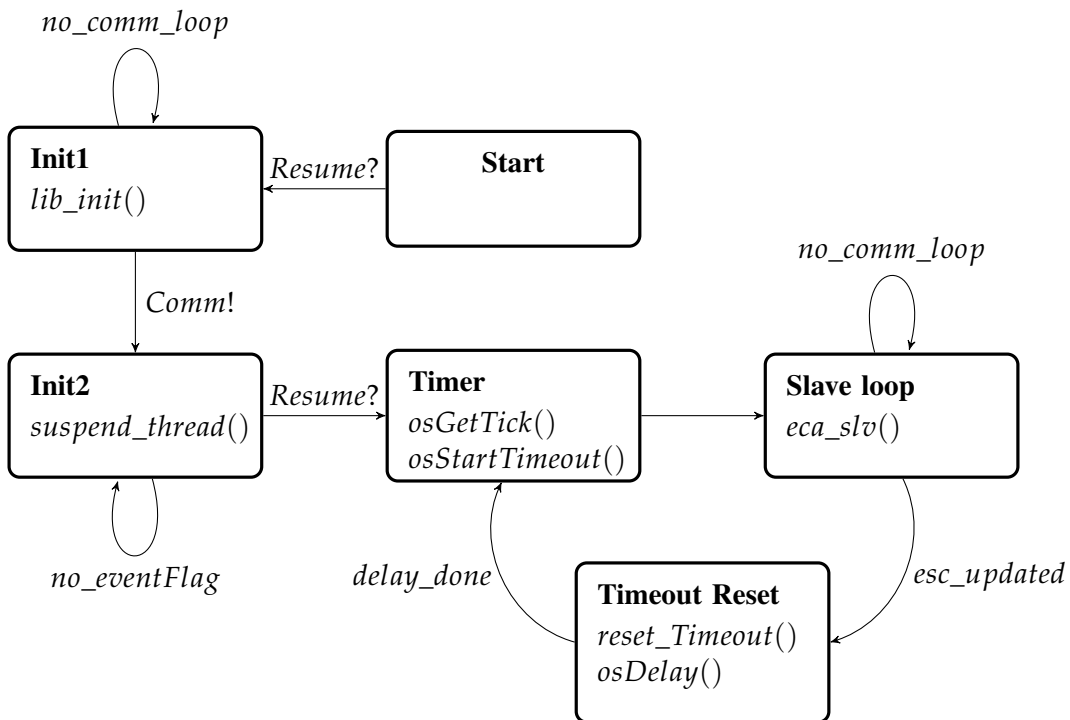
Testing I The overall integrity and functioning of the Power-on and SWD-Programming of the STM32 MCU via SWD/JTAG connector on-board was firstly tested with good results.

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 AXB 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.

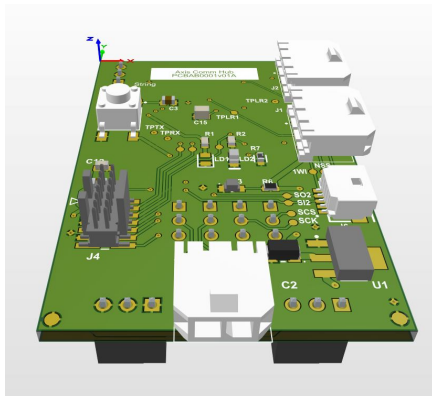
Testing III This phase is an ongoing task, since depending on the development of all the features, communication speeds and hardware configurations, different scenarios are continuously appearing. A more deeper analysis will be taken into account for next versions.



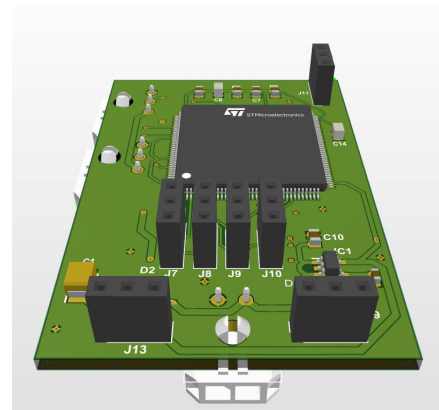
(a) Synchronization state machine



(b) SOES application state machine



(a) ACB 3D view top



(b) ACB 3D view bottom

Figure 4.6: 3D model generated by Altium for the final design.

4 IMPLEMENTATION

Results

****Redaction style at 85/100**

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 fotos and screenshots will be presented. Some challenges, their solving* and further improvements are also presented along the chapter.

5.1 PCB, SPI and LED*

In the figure ?? the final PCB prototype can be seen. It is worth mentioning that one issue that emerged was related to the physical requirements by the temperature sensors. According to the library integration process commented in 4.2, during the first month a rather simplistic approach was coded to have access to the temperature values with provided example sensors using an external power source, this contrasted with the final sensors that 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 independant pins were explicitly needed, namely RX and TX. That 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 marely for testing and will be corrected in any further development out of the scope of this Research Project.

*Include pcb foto

Another issue that is good to remark, is the noisy communication that emerged at SPI higher speeds. Generally, the debugging of SPI signals was very frequently, since for any new modification of the logic lead to the possibility of a fault, and those needed to be traced back to their origin, being sometimes the Host MCU or the LAN9252 chip. Nevertheless, to hook up

5 RESULTS

| Config | GPIO setup | Comm | Visibility |
|---------|--------------------------------------|------------|------------|
| 1 | PU: SS NPU/NPD: SCK//MOSI/MISO | @10Mbps ⊖ | ⊕ |
| 2 | PU: SS NPU/NPD: SCK//MOSI/MISO | @20Mbps ⊖ | ○ |
| 3 | PU: SS NPU/NPD: SCK//MOSI/MISO | @40Mbps ⊖ | ⊖ |
| 4 | NPU/NPD: SCK/SS/MOSI/MISO | @2.5Mbps ⊕ | ○ |
| 5 | NPU/NPD: SCK/SS/MOSI/MISO | @20Mbps ○ | ⊖ |
| 6 | NPU/NPD: SCK/SS/MOSI/MISO | @40Mbps ○ | ⊖ |
| Current | NPU/NPD: SCK/SS/MOSI/MISO | @10Mbps ⊕ | ⊖ |

Table 5.1: Electrical configuration of the SPI GPIO ports affected the reliable communication if adjusted to be visualized through a logic analyzer or oscilloscope. Regarding speeds, above 10Mbps the software delay counteracts the bus datarate.

to the SPI bus introduced another fault source that was not identified until the communication speeds and their quality were generally characterized. In order to keep continue with more reliable test environment, it was decided to decrease the SPI data rate without affecting the considerations previously mentioned in table [?]. 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 all the related functions, namely between 2.7us to 2.9us; which starts to be significant as the interface configuration goes up. The observations regarding this issue are presented in table 5.1 and ascreenshots in 5.1.

This overall problematic is rather common within hardware design, therefore, longer times are needed to fulfill the requirements of an optimized hardware design when it comes to sensitive data signals. The mentioned working line itself is very broad and an introduction to the topic can be seen in [?],this momentary scapes from the scope of this Research Project.

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

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 lays on 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 2 DMA, each with 8 streams and each stream in turn with 8 channels[?],

the mentioned option did not represented a problem. Nonetheless, it could be argued that only one PWM generator could control up to 4 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 ready 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 efficece 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 needs expertise with DMA peripheral and formal evaluation of the benchmarks related to the utilization of the processor and memory footprintl 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 [?] or transfering of memory chunks at way faster speeds. The latter could be though interesting, since the LAN9252 counts with SQI interface.

*Include foto of the LED Rings

5.2 DSMs and RTOS scheduling

**Redaction style at 80/100

As introduced during the implementation section 4.4.1, this part will comment a bit further the results of the attempts for using an RTOS to schedule and synchronize the execution of the DSMs, such that a deterministic behaviour could be reached. The same manner as the previous section, some issues and comments about it are going to be presented along the way.

Firstly, in the table 5.2 the current list of threads is presented. Important to point out is the necessity to define priorities carefully besides the previously given *desired* release times. In the same table is included an OS-defined thread that is being crucial for current SW structure. Some further information regarding this specifications are commented next.

**The quality of the redaction style here is 80/100

Regarding the *User Task Manager* it was implemented since during the synchronization of ECAT and SOES DSMs -recall the synchronization signals 4.4- forcing threads to suspend was meaningful, every time the SOES went into infinite loops due to the way it is natively implemented. This is, polling continuously ESC state registers instead of using and IRQ pin -recall SM-Synchronous in 4.3.1-. This means that the SOES APP cannot return from its own called subfunctions if a physical connection problem appear, such as power-off of ESC itself, cable disconnection, power source drop, etc. Given this scenario, clearly a solution with timeout control was tried out but the low level timers (HW peripherals) induced problems as it is of not-good practice -or at least demands deeper understanding of HW and the RTOS-

5 RESULTS

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

■ **Table 5.2:** 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*

to mix hardware interrupts when running OS. Therefore, it was switched to use callback functions from osTimers (Timers implemented by the OS). This timers are really managed by the so-called osTimer Daemon Task, which is fully configurable but a few characteristics need to be taken into account as the Daemon, as the name suggests, runs as Task hidden for the user and if this Task is not correctly set, callbacks can be lost as the Damon tries to interrupt a higher priority task. Some of the parameters to configure are callback stack depth, queue length and priority, more information regarding this can be found in [?].

Once timers are set can interrupt threads depending on their priorities, if one task is within an infinite loop and keeping its own state machine and consequently others is considered a fault. The osTimer should then create 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 -even software interruptions-. Furthermore, the suspension or termination of threads need to be handle with care, since if a thread is suspended or terminated right after a higher priority task preempts the callback function -if the Event SDM were of higher priority than SOES APP SDM-, as soon as the higher priority task finishes, the callback function would be executed -without problem since the osTimerDaemonTask is of higher priority too- but as the OS would try to go back to the thread were it was originally called, and this has been put out of the execution queue, this would result in os hard fault.

Just as the previous example, hard faults -not surprisingly- are to be avoided, but to do this demands rather *good strategy*. Moreover, this situation could repeat in other conditions, even due to implementation of the OS. For example, there are even osFunctions that explicitly said to be executable within ISR, might have no effect, as was the case for Event Flags -considering of course their respective volatile definitions-; or waiting states in a DSM can be result in hard fault since the execution of an unique Task that end up calling apparently endless a osDelay function is avoided by the OS itself.

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 of the 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.

As to another topic, since it could be quite demanding to know the precise execution time of each Task it is not possible to think about optimizing the utilization or the OS configuration in a correct manner -if even required-. For instance, calculation of the Utilization factor that is helpful for scheduling, could be also needed in more practical design cases, e.g., while considering heat sinks for processors within enclosed devices.

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 and through 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 (**include example of one). Omitting the rather small HAL overhead added with the GPIO control, an estimation could be achieved.

Live thread tracing A trace debugging like SEGGER SystemView[?]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 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

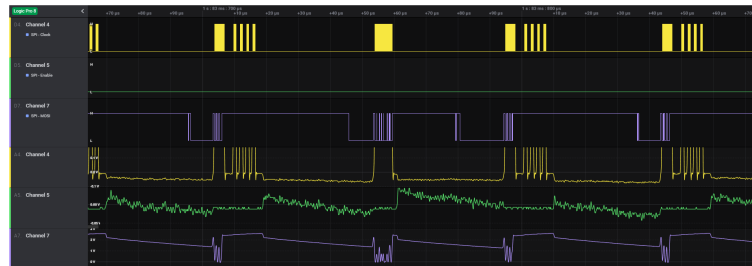
**This is a draft

The results presented in this section implies the correct functionality of the temperature functional block.

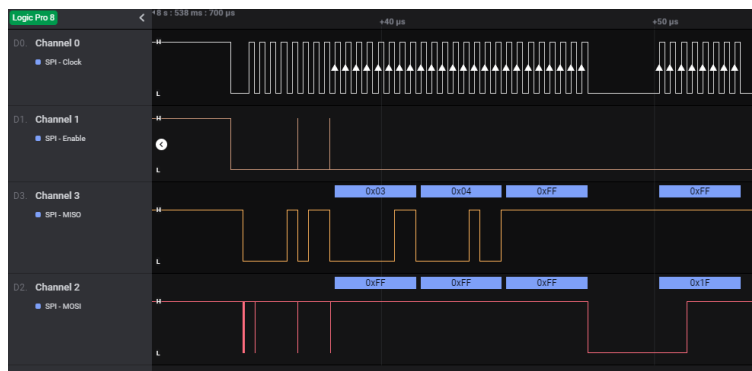
*Include Wireshark traceInsert captures of the EtherCAT frames/datagrams for initialization or a change of state INIT-OP

**Insert a capture of XAE showing the EtherCAT Slave within the EtherCAT tree. Moreover, the data types declared in SText to update in free-run mode the values sent by the Slave device.

5 RESULTS



(a) Non Pulled-up signal



(b) Spikes at higher datarates

■ **Figure 5.1:** Issues with SPI bus. In 5.1(a) the green signal corresponds to SS pin and demands a pull-up resistor to proper visualization but LAN9252 demands open-drain pin. Where as in 5.1(b) Ch1 and Ch2 show spikes that emerged more frequently after 5 Mbps, leading to a incorrect recognition of the CLK signal.

Conclusions

6 CONCLUSIONS

Bibliography

- [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.
- [Bec17] 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. EtherCAT - the Ethernet Fieldbus. <https://www.ethercat.org/en/technology.html>, January 2020. Last visited: 16/09/2020.
- [Bec20b] 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.
- [Bor08] Detlef 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] Raimarius Delgado and Byoungwook 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.
- [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.
- [Gmb20] Beckhoff Automation GmbH. EtherCAT Slave Controller Overview. <https://www.ethercat.org/en/downloads.html>, January 2020. 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.
- [Han20] Han's Robot Germany. Homepage. <http://hansrobot.de/home>, September 2020. Last visited: 16/09/2020.
- [HWHP] W. Hagemeister, Weber, R. Hacker, and F. Pose.

BIBLIOGRAPHY

- [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.
- [Inf18] Infineon Technologies AG. XMC4700 / XMC4800 Datasheet. <https://www.infineon.com/cms/en/product/>, September 2018. Last visited: 17/09/2020.
- [ISO17] IEEE ISO, IEC. 8802-3:2017 Part 3: Standard for Ethernet. <https://www.iso.org/standard/72048.html>, 2017. Last visited: 11/09/2020.
- [JDH02] B. C. Johnson, D. G. 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.
- [MGSR20] M. A. 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.
- [MLH⁺17] L. Muratore, A. Laurenzi, E. M. Hoffman, A. Rocchi, D. G. Caldwell, and N. G. 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. G. 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.

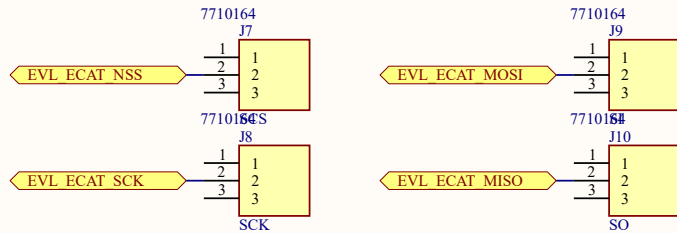
- [Ser18] Sercos. Ethernet TSN heralds a new era of industrial communication. <https://www.sercos.org/technology/sercos-in-connection-with-tsn-opc-ua/sercos-in-connection-with-tsn/>, January 2018. Last visited: 11/09/2020.
- [Ser20] Sercos. Driver Software. <https://www.sercos.org/technology/implementation/driver-software>, January 2020. Last visited: 11/09/2020.
- [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] Maneesh Soni. EtherCAT on Sitara Processors. Technical report, Arm Microprocessor Group Texas Instruments, April 2020. Last visited: 17/09/2020.
- [Sta00] DN Staff. SERCOS picks up the pace. <https://www.designnews.com/sercos-picks-pace>, August 2000. Last visited: 11/09/2020.
- [Sta20] SERCOS Staff. SERCOS performance. <https://www.sercos.org/technology/advantages-of-sercos/performance/>, January 2020. Last visited: 11/09/2020.
- [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.
- [XJY11] Hu Xing, Huan Jia, and Liu Yanqianga. Motion control system using sercos over ethercat. *Procedia Engineering*, 24:749–753, 12 2011.

BIBLIOGRAPHY

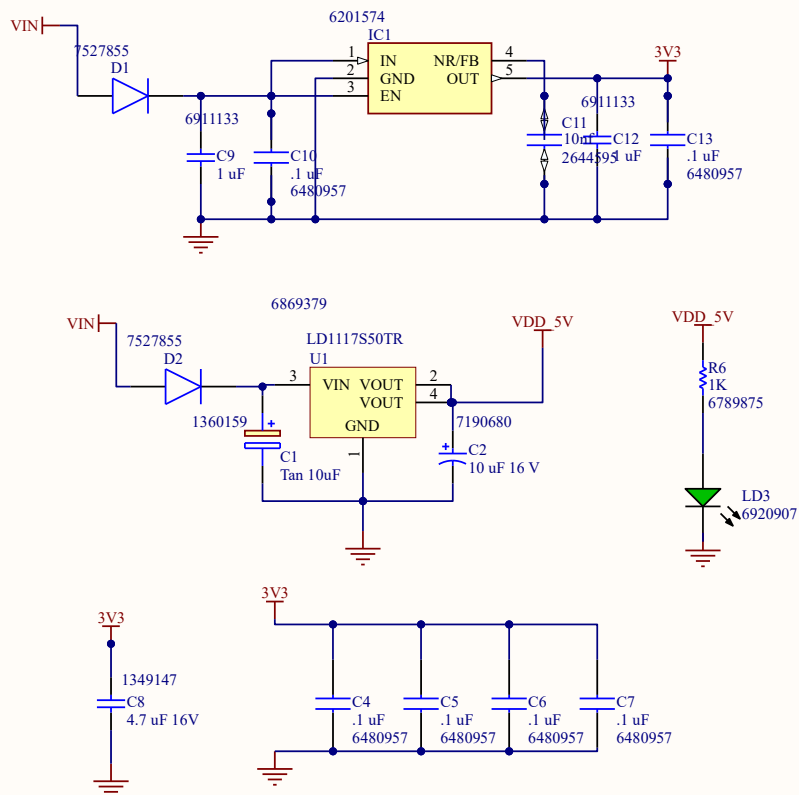
PCB drawings and layout

****here comes the electrical diagrams and the pcb layout exported by Altium****

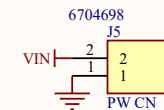
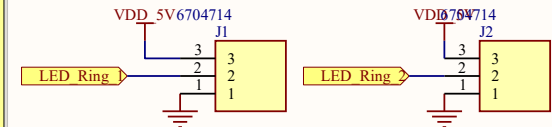
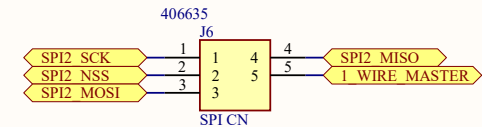
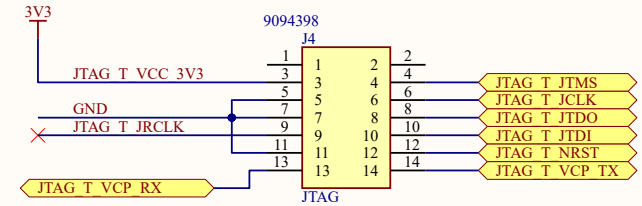
Adaptors to LAN9252_EVB_SPI



PWR



CONNECTORS



TODO LIST for second version (2020.08.03)

- Adapt an STM32F4xxx MCU of smaller package (QFN possible) to assess 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**

Size: A4

Number:1

Revision:3

Date: 9/17/2020

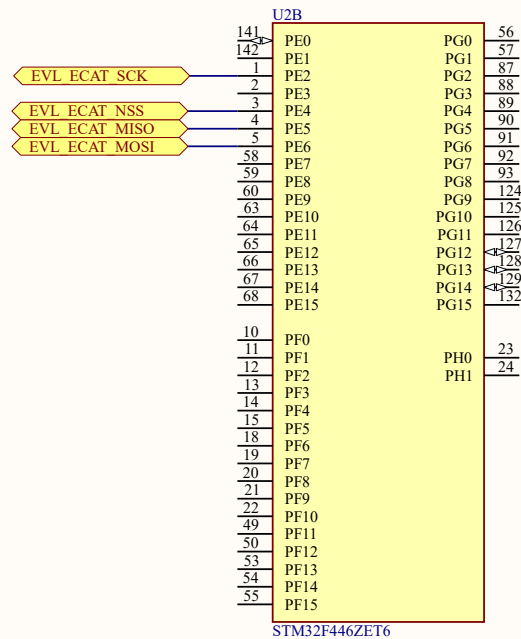
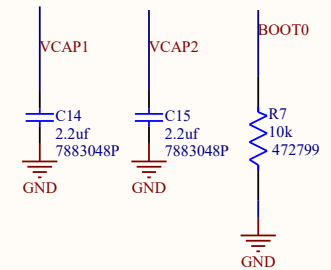
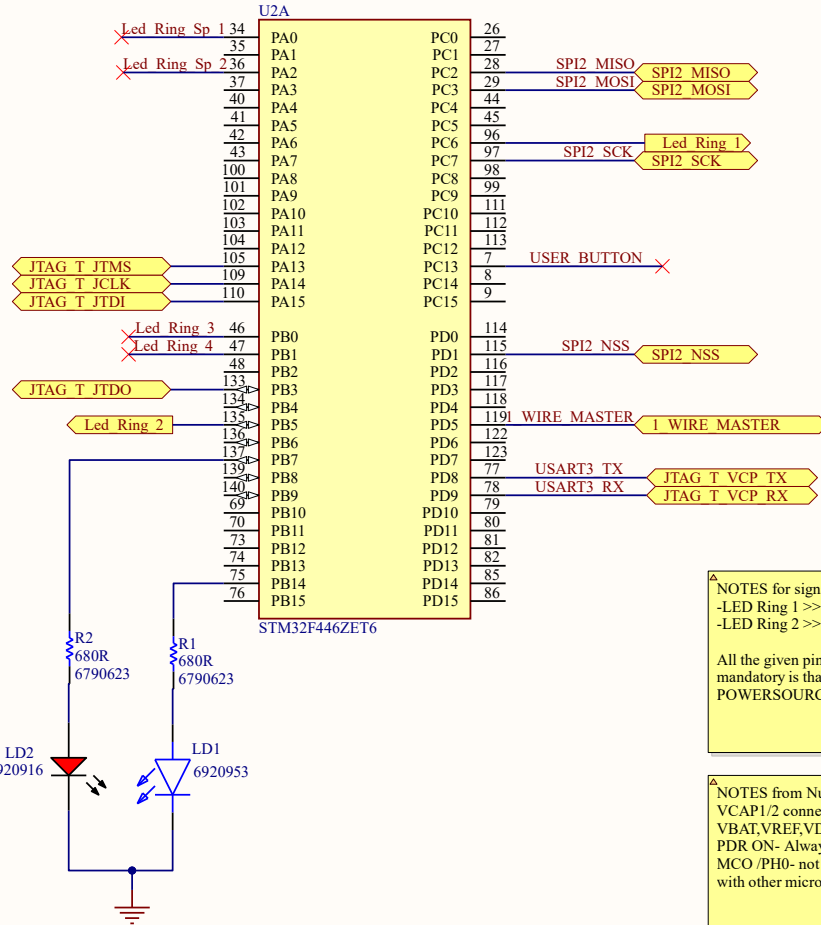
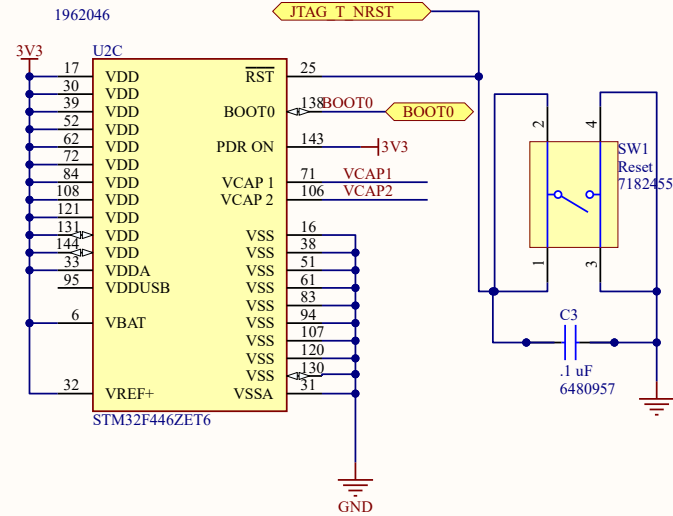
Time: 11:16:22 PM Sheet 1 of 2

File: D:\PCB\Projektarbeit\PCB_AxisCommHub\PowerSource.SchDoc

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




MCU



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 | | | <i>Han's Robot Germany GmbH</i> <i>Barmbeker Straße 9a</i> <i>22303</i> <i>Hamburg</i> <i>Germany</i> |  |
| 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 | | | | |

A PCB DRAWINGS AND LAYOUT

Source codes

****Redaction @ 0/100**

```

    /*
    * smEcat.c
    *
    * Created on: Jun 26, 2020
    * Author: CarlosReyes
    */

#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; //CHCKME These might have been ✓
    substituted by soesTimeoutFlag
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;

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[];

```

B SOURCE CODES

```
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 ✓
            transmitting interruption
            ecatDMArcvd = FALSE;
            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 coming from SPI or something similar
            else {
                notifyError(EV_ECAT_APP_NOK);
            }
        }
    }
}

```

B SOURCE CODES

```
        ecat_step = ec_fault;
    }
    break;
} //TODO DMAReceived should be changed by interruption

if(timeoutEcat) {
    notifyError(ERR_ECAT_TIMEOUT);
    timeoutEcat = 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
```

```

        escAPok = 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);
}

```

B SOURCE CODES

```
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 overflowed.
 */
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 B.1:** Part of the source code for ECAT DSM

```
/*
 * soesApp.c
 *
 * Created on: Jul 16, 2020
 * Author: CarlosReyes
 * 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)
{

```

B SOURCE CODES

```
// 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;

}

/** Optional: Hook called after state change for application specific
 * actions for specific state changes.
 */
void post_state_change_hook (uint8_t * as, uint8_t * an)
{

    /* Add specific step change hooks here */
    if ((*as == BOOT_TO_INIT) && (*an == ESCinit))
    {
        boot_inithook ();
    }
    else if ((*as == INIT_TO_BOOT) && (*an & ESCerror ) == 0)
    {
        init_boothook ();
    }
}

void post_object_download_hook (uint16_t index, uint8_t subindex,
                               uint16_t flags)
{
    switch(index)
    {
    case 0x7100:
    {
        switch (subindex)
        {
        case 0x01:
        {
            //encoder_scale_mirror = encoder_scale;    //Pending The 0x7100 ✓
            address object could be used afterwards
            break;
        }
        }
        break;
    }
    case 0x8001:
    {
        switch (subindex)
        {
        case 0x01:
        {
            Cb.reset_counter = 0;
            break;
        }
        }
    }
}
```



```

        }
        break;
    }
}

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) {

```

B SOURCE CODES

```
        // 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:
    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;

```

B SOURCE CODES

```
        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 B.2:** Source code for SOES APP DSM