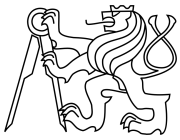
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	IO-Link Device for testing of IO-Link Masters
Student:	Bc. Ondřej Volf
Supervisor:	Ing. Miloš Fenyk
Study Programme:	Informatics
Study Branch:	Design and Programming of Embedded Systems
Department:	Department of Digital Design
Validity:	Until the end of summer semester 2017/18

Instructions

Prepare IO-Link device module allowing its configuration in full range of IO-Link communication parameters, simulation of process data and diagnosis generation regarding IO-Link Specification.
Write requirement specification for the IO-Link Device module.
Prepare design of the module in the program Enterprise Architect based on collected and elicited requirements.
Microcontroller shall be selected from the STM32FXXX product line and the design implementation shall be in the C++ programming language.
All steps shall be coordinated with the Prague IO-Link Competence Center.

References

Will be provided by the supervisor.

doc. Ing. Hana Kubátová, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 23, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design

Master's thesis

IO-Link Device for testing of IO-Link Masters

Bc. Ondřej Volf

Supervisor: Ing. Miloš Fenyk

9th May 2017

Acknowledgements

I would like to thank to all my colleagues in Siemens Development team for creating great environment. Special thanks goes to Lukas Hamacek for giving me the opportunity to work on this thesis in Siemens and Milos Fenyk for supervising this thesis and providing the best support possible.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 9th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Volf, Ondřej. *IO-Link Device for testing of IO-Link Masters.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Během vývoje IO-Link Master produktů je třeba průběžně testovat funkčnost pomocí různých IO-Link deviců. Tato práce si klade za cíl vytvořit IO-Link test device, který by proces testování značně zjednodušil a umožnil by testovat efektivněji.

Klíčová slova IO-Link, IO-Link Master, IO-Link Device, Vývoj vestavěných zařízení

Abstract

During development of IO-Link masters there is need for proper testing using various IO-Link devices. This thesis have a goal to create specific IO-Link test device, that would easen up this testing and allow ways to test more efficiently.

Keywords IO-Link, IO-Link Master, IO-Link Device, Embedded device development

Contents

Citation of this thesis viii

1 Introduction **1**

2 Basics **3**
2.1 IO-Link Technology 3
2.1.1 System topology 4
2.1.2 Physical Layer 5
2.1.3 Data Link Layer 8
2.1.4 Application Layer 13
2.1.5 IO-Link Device Description 15

3 Analysis and design **17**
3.1 Introduction to problem and motivation 17
3.2 Collecting requirements 17
3.2.1 Stakeholders 18
3.3 Categorizing requirements 21
3.3.1 Functional Requirements 22
3.3.2 Non-functional requirements 26
3.4 Planning 26
3.5 HW Selection 27
3.5.1 Micro-controller selection 27
3.5.2 PHY selection 30

4 Realisation **35**
4.1 Overview 35
4.1.1 Used components 35
4.2 Basic Design 36

4.2.1 External parts 37
4.3 BSP 38
4.3.1 BSP 39
4.3.2 MCU 40
4.3.3 PHY 44
4.4 Stack integration 44
4.5 Gateway application 46
4.5.1 Mail System 46
4.5.2 Configuration 48
4.5.3 Process Data 49
4.5.4 Digital Inputs and Digital Outputs 49
4.5.5 Events 50
4.5.6 File System 51
4.6 RTOS integration 53
4.6.1 Optional usage 53

Conclusion **55**

Bibliography **57**

A Acronyms **61**

B Contents of enclosed CD **63**

List of Figures

1.1 IO-Link Test Rack 2

2.1 IO-Link Example topology 5
2.2 IO-Link Master topology 6
2.3 Pin layout of connectors 7
2.4 Physical layer of IO-Link Master 8
2.5 Physical layer of IO-Link Device 8
2.6 SDCI message sequences 9
2.7 Overview of M-sequence types 10
2.8 Data Link Layer of IO-Link Master 11
2.9 Data Link Layer of IO-Link Device 12
2.10 Successful establishment of communication 13
2.11 Communication establishment retried after no response 13
2.12 Application Layer of IO-Link Master 14
2.13 Application Layer IO-Link Device 14

3.1 IO-Link Master test partner use case 20
3.2 IO-Link Master current load test use case 22
3.3 Basic Scheme 29

4.1 Deployment Diagram 36
4.2 Component Overview 38
4.3 Strategy Pattern 41
4.4 Strategy Pattern GPIO 42
4.5 GPIO Abstraction 43
4.6 Integration of TMG IO-Link Stack 44
4.7 Publisher-subscriber pattern 47
4.8 Mail Queue 48

4.9 File System Design 52
4.10 Test Device Board 55

Chapter 1

Introduction

With the rapid development of industrial processes in last few hundred years there are increasing demands for efficiency and flexibility of used technology. The vision of Industry 4.0, which is lately quite popular trend of automation and data exchange in manufacturing technologies, focus on usage of "smart", advanced and intelligent sensors and actuators working in intelligent networked factory.

One small part of Industry 4.0 was development of IO-Link standard (IEC 61131-9), that enables easy optimization of used processes and offers easy connection and communication with smart sensors and actuators withing a production cycle, highly contributing to the success of Industry 4.0[1].

IO-Link is technology supported and developed by world industrial leaders (e.g. Balluff, Siemens, MaximIntegrated, Panasonic, SICK and many more), which develops so called IO-Link Masters (gateway to industrial ethernet) and IO-Link devices (sensors/actuators that use IO-Link protocol). In Siemens Prague branch office of research department, there is currently ongoing development and maintance of IO-Link Master products. This development and maintance subject to many tests, that guarantee bug-free and safe result. For this testing there are many various test scenarios, using many different devices as can be seen in used test rack in figure 1.1, to properly test every funtionality of IO-Link Master. These tests are sometimes very time inefficient, difficult and expensive.

To counter these problems, new IO-Link test device was proposed at the development department. Device, that would help with automatization of testing and that would be usable for as many test scenarios as possible. Project to create such device was therefor initiated as low priority task, assigned to 3 students with supervision of senior developers. Task was

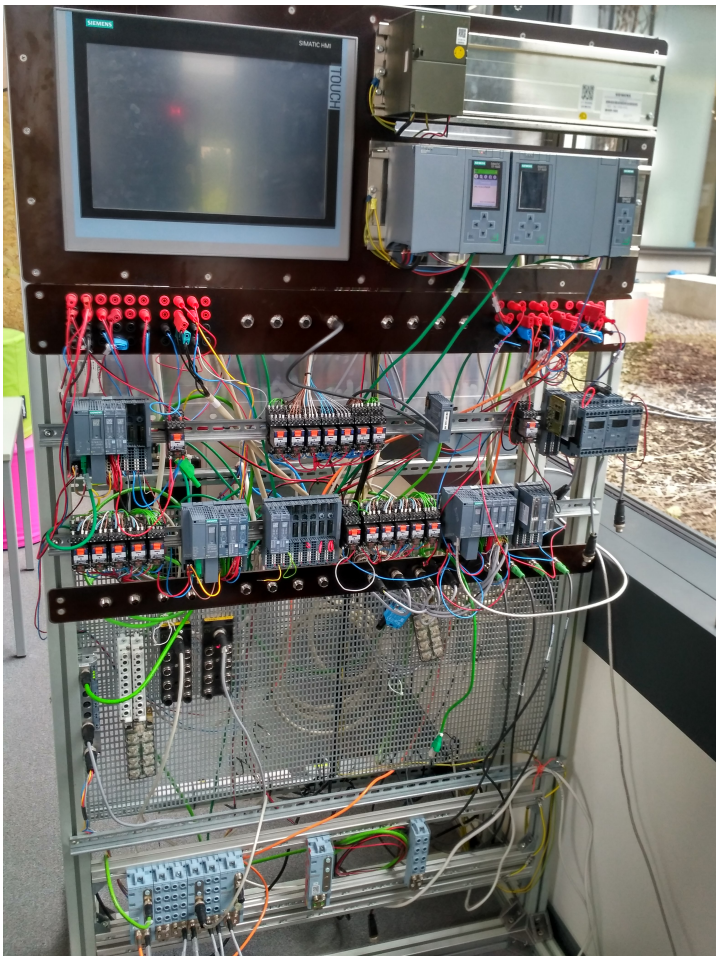


Figure 1.1: IO-Link Test Rack - Rack that is used to test Siemens IO-Link Masters. In upper part there are PLCs, IO-Link Masters and relays for automatization and in lower part there are various IO-Link Devices.

divided into 3 sections. Complete HW development, FW development and communication protocol with PC application.

This thesis is focused mainly on FW development, but will partly involve other sections as well, since they are interconnected. In this thesis there will be brief description of IO-Link protocol. Because for development of any product, there are firstly needed detailed requirements from all interested parties, the second chapter will focus on collecting and evaluating of these requirements as well as some predevelopment processes (e.g. planning and HW selection). Last part of thesis is description actual realisation of final device.

Chapter 2

Basics

2.1 IO-Link Technology

With the increasing power of microprocessors and other electronic components the industrial automation require some progress on end devices as well. This leads to development of improved sensors, actuators or integration of both, that don't use standard binary inputs/outputs, but are more sophisticated and offers way more functions than simple devices (e.g. reporting events or configuration of devices) to fit into new standard of Industry 4.0. [1]

For this reason communication standard with the trade name IO-Link has been designed and standardized under the norm IEC 61131-9 under the term Single-Drop Digital Communication Interface (SDCI) for small Sensors and Actuators [2]. This communication standard defines electrical connections, communication protocol and many constraints and requirements to create safe and reliable protocol. Communication is based mostly in standard 24V UART with baudrate up to 230,4 KBaud [3].

Main advantages of IO-Link technology are for instance [4]:

- Noise immunity - IO-Link data transfer is based on a 24V signal and is therefore extremely insensitive to external influence.
- Identification of devices - IO-Link ensures unambigous device identification. Sensors with IO-Link capability are clearly and uniquely identified by vendor and device ID.
- Diagnosis detection - With IO-Link transfer process and service data takes place simultaneously. Any events on device like wire-breaks or short-circuits are immediately detected by the IO-Link master and

reported to control unit. Diagnosis data can be accessed even during operation.

- Easy IO-Link device replacement - Configuration of device can be stored in IO-Link masters and automatically downloaded to new unconfigured device. This makes device replacement during maintanance fast and easy and prevent workers from making any mistakes during new configuration.

- Standardized connectors - Simple and standardized connectors for IO-Link reduces variety of interfaces.

- Integration to any system - IO-Link is an open standard so devices can be integrated in virtually any fieldbus or automation system (e.g. PROFIBUS, PROFINET, EtherCAT).

Together these capabilities result in overall reduced costs, increased process efficiency and improved machine availability. The benefits of IO-Link are especially significant in applications that are frequently modified and are severely impacted by extended unplanned downtime.

Based mainly on the IO-Link specification [5], this chapter gives a short insight over the basics of IO-Link technology, its topology, physical and electrical properties and data flow as well as basics of device identification. Information in this chapter won't run into details and will cover only shallow parts of this standard to understand rest of the thesis.

2.1.1 System topology

IO-Link provides up to the lowest level of an automation hierarchy. In the past sensors and actuators were connected to industrial fieldbus through simple remote I/O devices. In case of IO-Link this particular function is handled by IO-Link Master module, which provides gateway between IO-Link and used fieldbus and connects IO-Link devices via point to point communication through simple three, respectively five, non-shielded wire connection. IO-Link master is connected via chosen fieldbus or industrial ethernet to PLC. Usual topology can be seen on figure 2.1.

IO-Link Masters are essentially the heart of IO-Link communication. Without their gateway there wouldn't be any way to connect devices to fieldbus and therefore to PLC. On figure 2.2 it is shown how IO-Link Master works. IO-Link Master has usually several ports, that work completely independently and therefore every port can handle only one device without any interference from others. This also means there is no addressing

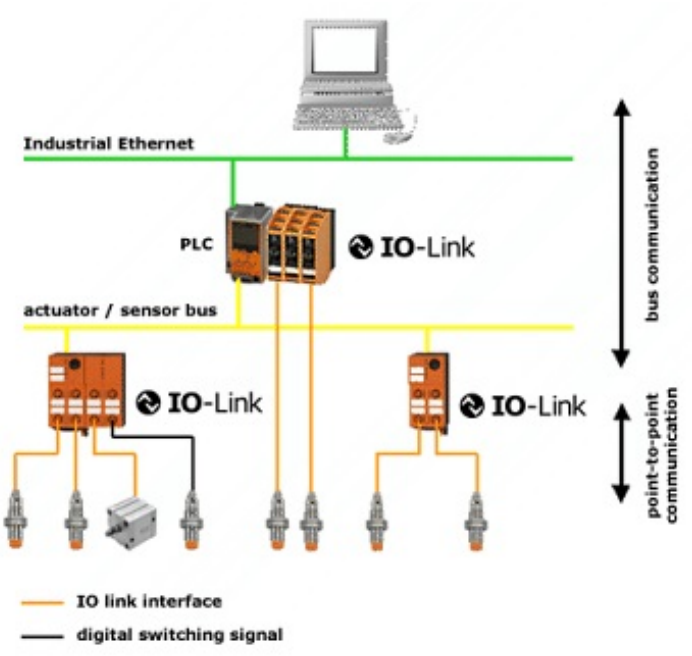


Figure 2.1: IO-Link Example topology - Orangle lines represent IO-Link communication, yellow lines are industrial fieldbus and green line is usual industrial ethernet

during communication which removes some additional header data from data packets. Physically the Master need to have as many ports/connectors as many devices user wants to connect. Ports of IO-Link Master can also be configured to SIO mode, which means standard digital input/output. This helps to integrate IO-Link to systems that are not fully IO-Link compatible.

On figure 2.2 it can be also seen that communication with Fieldbus is handled entirely by Gateway part. This enables porting IO-Link Masters virtually to any existing Fieldbuses. Either the standardized mapping in fieldbuses, e.g. for PROFIBUS, PROFINET, EtherCat, or manufacturer-specific mappings for Ethernet/IP, CANopen, Modbus, CC-Link. Only required parts are Master side standardized by norm IEC 61131-9 and Gateway side standardized by norm IEC 61158.

2.1.2 Physical Layer

IO-Link has been designed and standardized so it works with digital input and output interfaces according to IEC 61131-2 in a point-to-point connection. For this reason IO-Link uses 24V level. Moreover, the usage

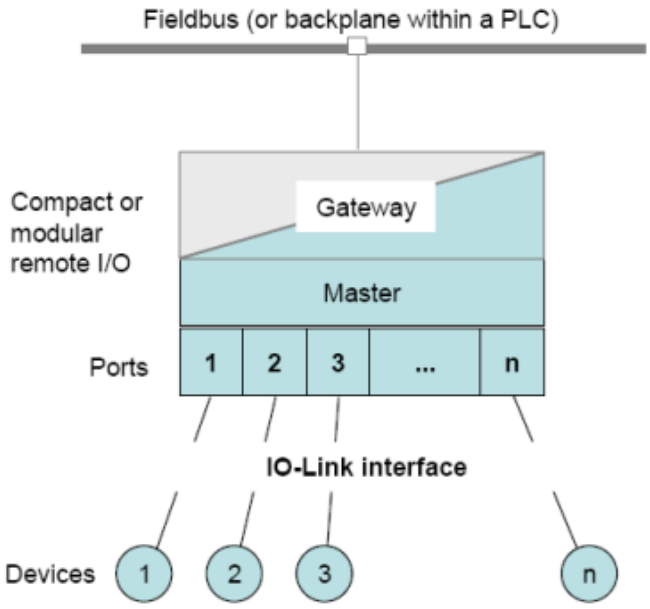


Figure 2.2: IO-Link Master topology - Topology of IO-Link master is basically a star network. Center of this star network is IO-Link master itself, which communicates separately with every IO-Link device connected. [6]

of this voltage level makes the IO-Link protocol very robust and suitable for industrial usage, which can produce strong electromagnetic interference, without need of shielding. Maximum length of cables is 20 m, maximum for overall loop resistance is 6.0 Ω and effective line capacitance has to be less than 3.0 nF to ensure reliable connection.

Connectors used for communication with sensors/actuators are 3-wire M5, M8 and most commonly used M12 port class A based on IEC 60947-5-2. The pin assignment is according to the IO-Link interface specification [5]:

- Pin 1: L1+ power supply - 24V
- Pin 3: L1- ground line - 0V
- Pin 4: C/Q switching line where C is used for SDCI communication and Q is standard input/output

These 3 Pins are standardized and are used everytime in connectors. Moreover, there is additional coding of the M12 connectors named port class B ilustrated in figure 2.3. Some device units do need additional

input/output channel on Pin 2 shown in assignment for port class A, or additional galvanically isolated supply voltage for components with higher power budget via port class B. Portclass B is needed mostly for actuators with higher power consumption, because L1+ can provide max 200mA according to specification. [5]

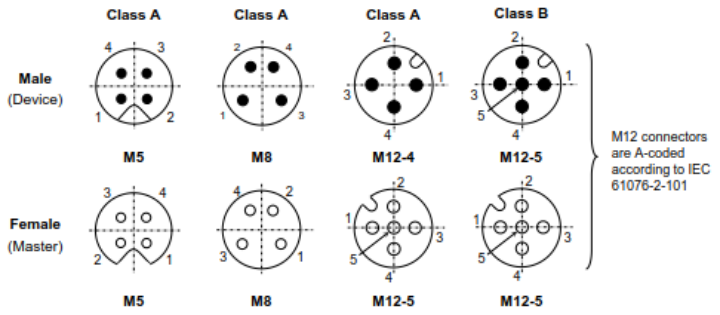


Figure 2.3: Layouts of all available connectors. In case of M12 Class B ports shall be marked to distinguish them from Class A ports, because of risks deriving from incompatibilities.

In figure 2.4 it is shown how master can change modes on its ports. During standard SDCI mode master periodically sends Wake-up pulses until proper response is received and communication is initiated. Once this happens, both master and device are set into their standard SDCI communication mode and communication is handled by their data link layers. During SIO mode, master acts like standard input/output device. In this mode there is no SDCI communication at all and communication layers are bypassed entirely to enable direct processing of input/output signals by the application layer. Port can also be deactivated in which case C/Q line shall be switched to high impedance and all communication shall be stopped.

On the device side there are same operating modes except inactive mode. In figure 2.5 we can also see, that most of the functions are in opposite direction. This is because of nature between Master and Device where Master controls communication and issues most commands. Device is by default set into SIO mode and acts as digital input, which allows detection of wake-up pulses coming from the master. Once wake-up pulse is received device goes into startup mode and initial communication is started. After successful initial communication the device switch to the IO-Link mode and standard operation starts.

Lastly whole communication is based on UART frames with inverted NRZ modulation. Therefore every logic '1' corresponds to a voltage

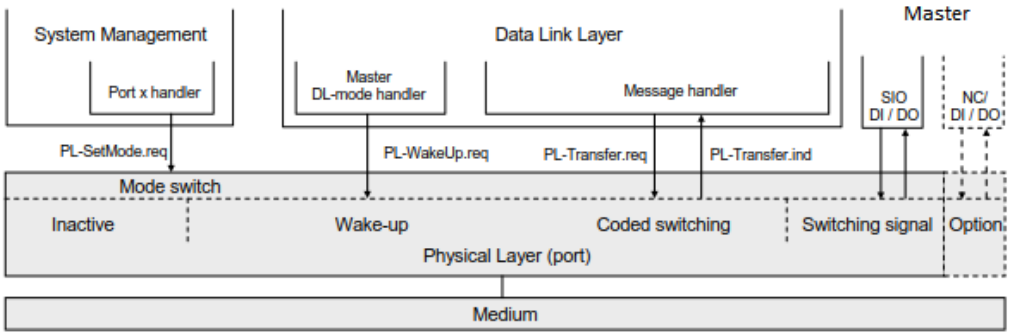


Figure 2.4: The physical layer of an IO-Link Master

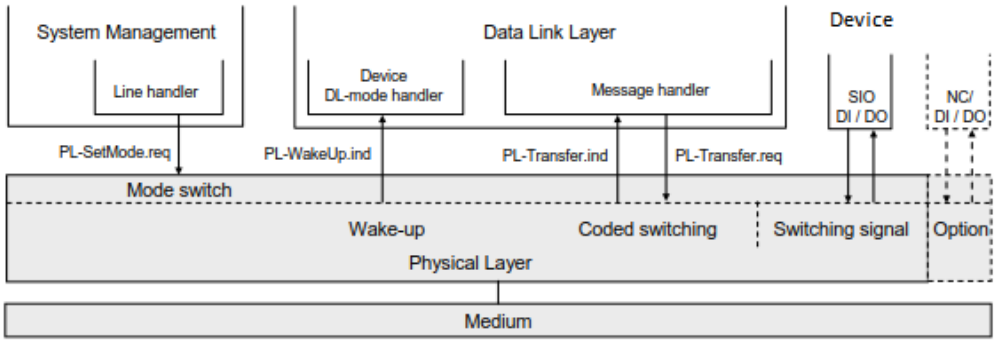


Figure 2.5: The physical layer of an IO-Link Device

difference of 0V between C/Q and L- and logic '0' corresponds to +24V difference between same lines. Communication is encoded into frames of 11 bits, with standard 1 bit length start and stop bits, 8 data bits and 1 even parity bit.

2.1.3 Data Link Layer

2.1.3.1 M-Sequence telegrams

Data link layer is basically core part of IO-Link communication, because it controls data transfer using IO-Link interface and function as interface between application layer and physical layer. Data frames are sent through physical layer in form of M-sequence (message sequence) telegrams. These telegrams consists of several UART frames, consider both request and reply part of message and differ based on type and size of data sent as can be

seen on 2.6. Every M-Sequence carry information about its data type. This helps to distinguish nature of carried data into several categories:

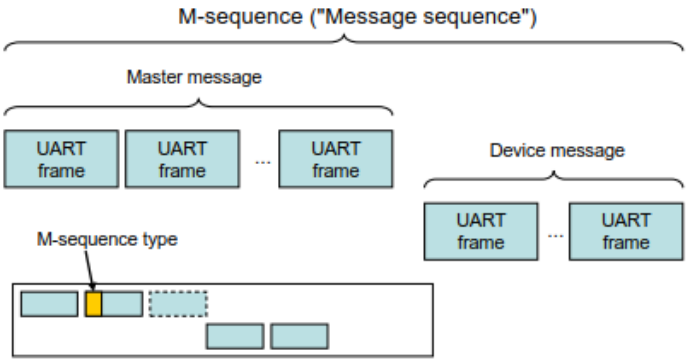


Figure 2.6: Every message starts with frames from master and ends with device reply. Type of message sequence is defined in first part.

- Process data - information that is periodically carried with high priority. Transfer begins automatically after start sequence without manual request from application. Process data involves measured values or control variables.
- On-request data - infortmation, that are sent only on application request. Specifically on request of Master unit. This involves configuration of Slave application (sensor) and data of certain events.
- Direct parameters / Diagnosis data - special on-request data, that are transferred without confirmation on reserved channel.
- Events / Service Protocol Data Units (PDU) - on-request information, that are transferred after receiving CHK/STAT flag bit in standard data channel. Unlike other On-request data transfer of events can be initiated by slave application.

All types of M-sequences are on figure 2.7. Every M-Sequence consists of mandatory "M-sequence Control" (MC) octet and another mandatory "CHECK/TYPE" (CKT) octet. These can be optionally followed by either "Process Data" (PD) and/or "On-Request Data" (OD) octets. Response is always ended by "CHECK/STAT" (CKS) octet. Detailed description of mentioned octets:

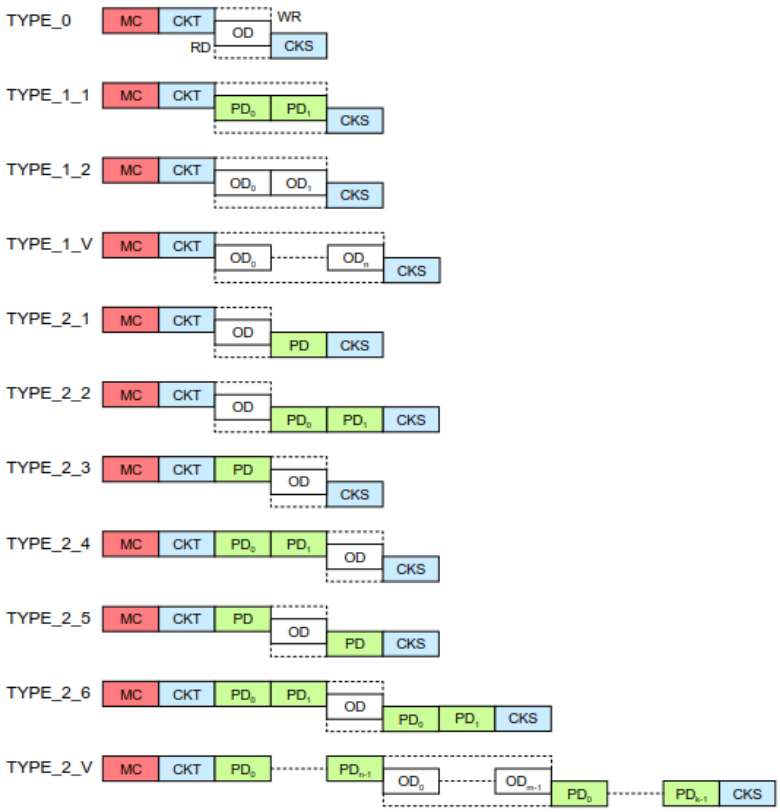


Figure 2.7: Overview of M-sequence types

- M-sequence Control - This octet initiates every Message sequence. It specifies type of operation (read/write - 1 bit), communication channel (PD, OD, Direct param, Events - 2 bits) and Address. Address consists of 5 bits which indicate the octet offset of the user data on the specified communication channel.

- CHECK/TYPE - This specifies type of M-Sequence (i.e. Type 0, Type 1 or Type 2 - 2 bits) and 6 bit checksum of data sent.

- On-Request data and Process Data - Data octets which don't involve any additional information.

- CHECK/STAT - This last octet indicates by 1 bit if any active Event is pending for Master retrieval, availability of process data by another bit and 6 bit checksum of all replied frames.

2.1.3.2 Structure of Data Link Layer

Data link layer (DL) also offers interface for application layer to exchange of Process Data (PD) and On-request Data (OD). Another set of DL services is available to system management (SM) for the retrieval of Device identification parameters and the setting of state machines withing the DL. The DL uses Physical layer services for controlling the physical layer and for exchanging UART frames. The DL takes care of the error detection of messages (whether internal or reported from the PL) and the appropriate remedial measures (e.g. retry) [5]. Structure of Master DL can be seen in figure 2.8. Device DL has quite similar structure as can be seen in 2.9. Difference is as in PL caused by Master issuing most commands and Device receiving them.

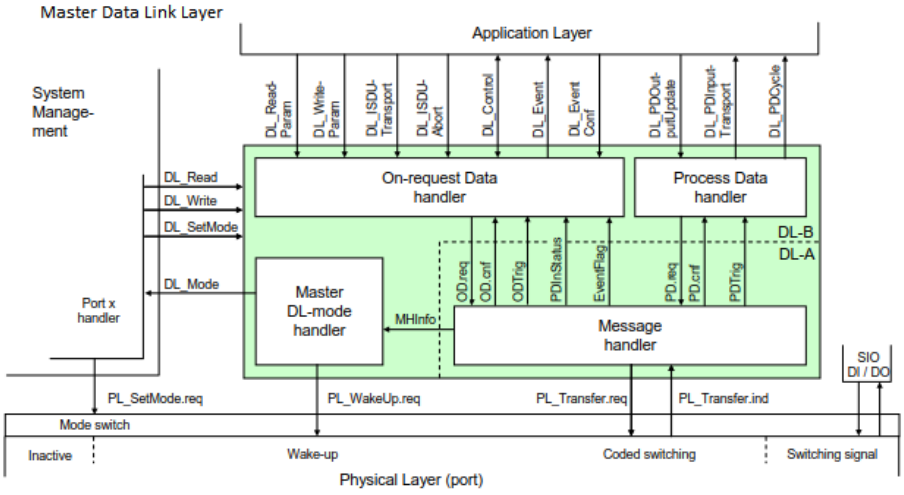


Figure 2.8: The Data Link Layer of an IO-Link Master.

The structure of DL is due to the nature of the data categoried into Process Data handlers and On-request Data handlers which are in turn using a message handler to deal with the requested transmission of messages. TThe special modes of Master ports such as wake-up, SDCI, and SIO (disabled communication) require a dedicated DL-mode handler withing the Master DL. The special wake-up signal modulation requres signal detection on the Device side and thus a DL-mode handler withing the Device DL. Each handler comprises its own state machine.

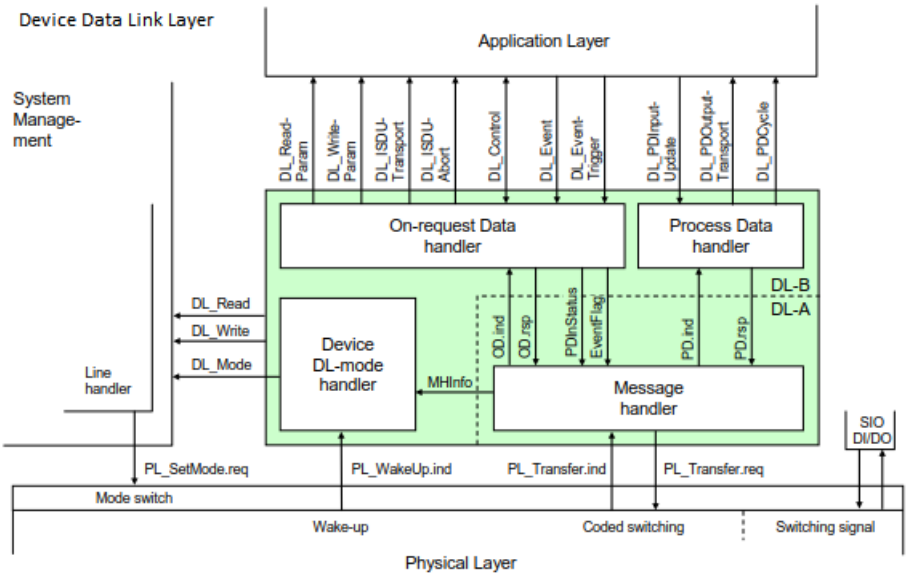


Figure 2.9: The Data Link Layer of an IO-Link Device

2.1.3.3 Establishment of communication

In previous sections there was mentioned so called wake-up request. Master attempts to establish communication by sending a wake-up pulse and then executing a Type 0 read frame. Because IO-Link supports 3 various speeds of baudrates, M-sequence is sent for every baudrate possible until response is received. This whole process can be seen in 2.10. In mentioned figure there are following states once wake-up request has been sent:

1. Master telegram in COM3 (optional)
2. Master telegram in COM2
3. Master telegram in COM1
4. Response from device in COM1

Depending on connected device, a valid reponse may even be received after COM3 or COM2 in which case another steps won't be required to run anymore. Once valid reponse is received, startup shall commence. If no response is received, whole procedure is repeated twice more. If even after that there is no response, there is fallback from C/Q to SIO mode and whole wake-up procedure continues after delay 2.11. For detailed information about establishing communication and mentioned time constraint see [7].

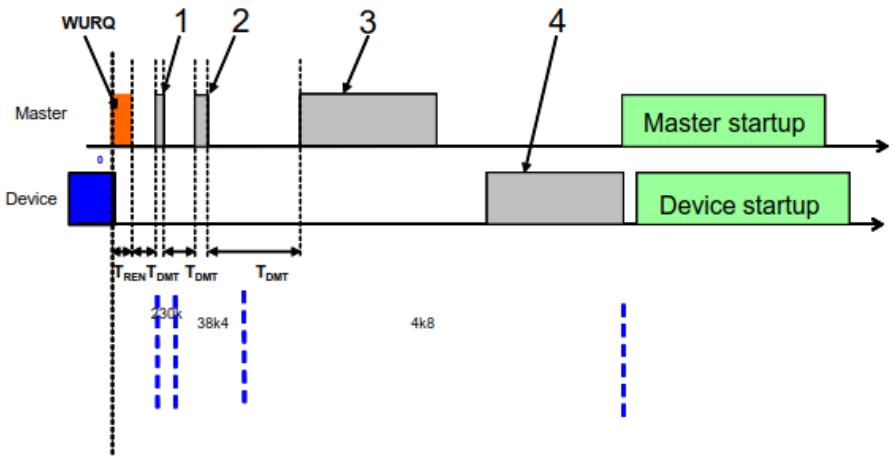


Figure 2.10: Successful establishment of communication. In this particular case response was send after COM1 request. In case of another baudrate it would be sent earlier and communication would be established faster.

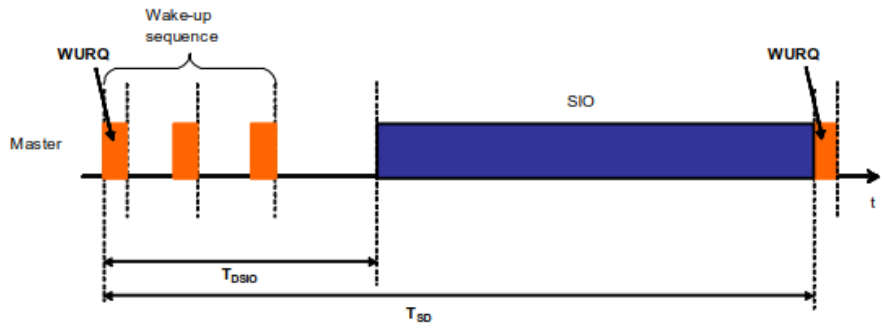


Figure 2.11: Communication establishment retried after failed wake-up sequence

2.1.4 Application Layer

Application layer (AL) of IO-Link communication provides interface between Data link layer and main application itself. As in other layers there is major difference in Master and Device AL. While master focus mainly on outgoing commands as can be seen in figure 2.12, the device focus on incoming ones 2.13.

In structure diagrams it is shown, that main application communicates only with System Management (mostly initialization) and Application layer. For this reason there is going to be brief description of mentioned services:

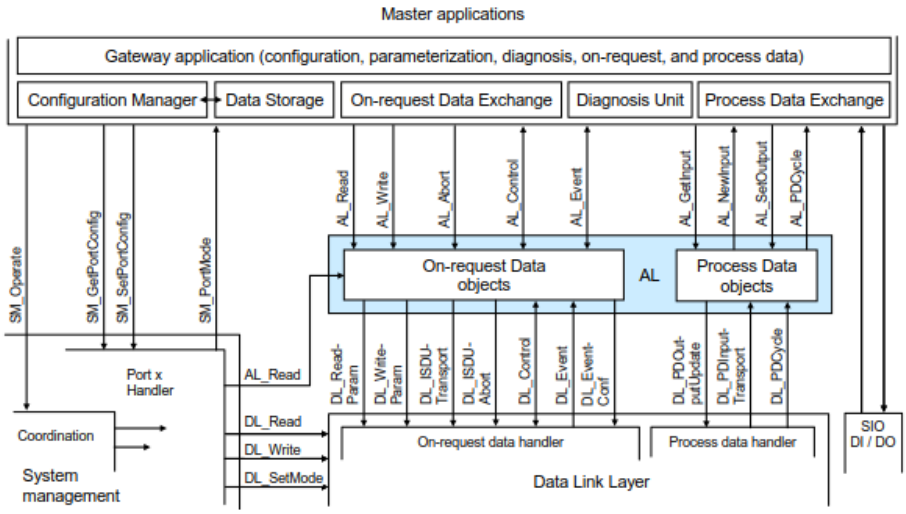


Figure 2.12: The Application layer of an IO-Link Master.

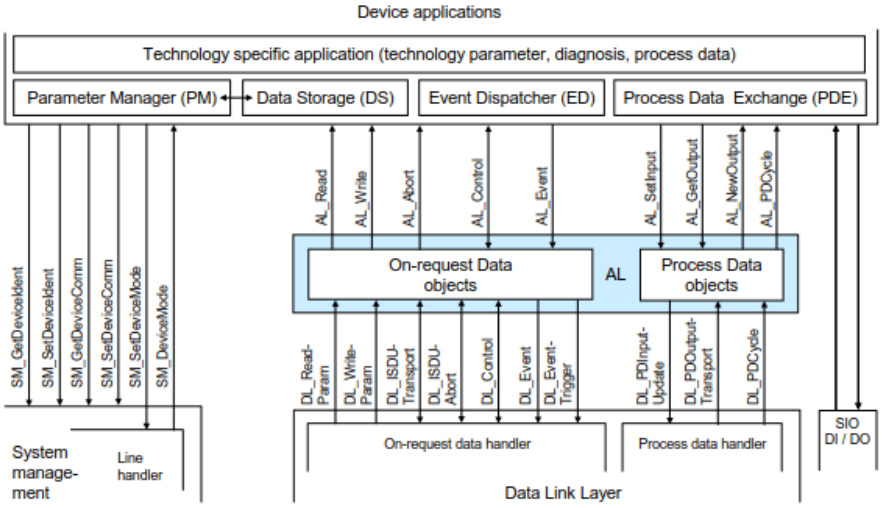


Figure 2.13: The Application layer of an IO-Link Master.

- AL_Read - This service is used to read On-request Data from a Device. Parameters define specific data record to read.
- AL_Write - This service is used to write On-request Data to a Device. Parameters specify, as in AL_Read, data record to write to and also data to be written.
- AL_Abort - This service is used to abort any pending AL_Read

or AL_Write services. Invocation of this service abandons any preparations for response.

- AL_Control - This service contains the Process Data qualifier status information transmitted to and from the Device application. This status information containst validity of both incoming and outgoing Process Data.

- AL_Event - This service indicates up to 6 pending status or error messages. The Event can be triggered by a communication layer or by an application.

- AL_GetInput - This service updates the input data within the Process Data on Master side.

- AL_SetInput/AL_NewInput - AL_SetInput updates Process Data on device side and triggers AL_NewInput data on Master side which can lately retrieve them with AL_GetInput.

- AL_GetOuput - This service updates the output data within the Process Data on Device side.

- AL_SetOutput/AL_NewOutput - AL_SetOutput updates Process Data on Master side and triggers AL_NewInput data on Device side which can lately retrieve them with AL_GetOutput.

- AL_PDCycle - This service indicates the end of a Process Data cycle. The Device application can use this service to transmit new input data to the application via AL_SetInput.

2.1.5 IO-Link Device Description

Every IO-Link Device made must have so called IODD (IO Device Description). This IODD is set of files that formally describes a device. The IODD is created by the device vendor and should be sufficient for IO-Link Tools to properly identify, parametrize, diagnose and communicate with the device.

The set of files consist of the maind IODD file, optional language files and optional picture files. Because this file is mandatory for every device, there is specification that describe detailed structure of IODD [8] and helps IO-Link Vendors create these files [9].

Main parts of IODD are:

- Device's identity - Unique identification of device. This involves Vendor ID (given to Vendors by IO-Link Consortium), Device ID and Product ID. If devices have same Device ID, there shouldn't be any differences in communication.
- Parameters of Device - Some Devices or actuators have parameters, which influence functionality. Every parameter and its functionality is described in this part.
- Diagnostic and process properties - Every possible event and detailed process data meanings must be described here.
- Communication properties - Communication details of Device (e.g. BaudRate, Process Data size, minimal cycle length, ...)
- Description of the Device - Basic description of device. (i.e. functionality, usage and/or contraints)
- Illustration of device - Optional icon picture, device picture and connection picture.
- Manufacturer details - Optional details about manufacturer and its logo.
- Another language options - Optional versions of IODD for multiple languages.

Main file of IODD is structured in standard XML format with "UTF-8" encoding. Names of these files are strictly given based on Device they are describing. Before issuing IODD it has to be verified by IODD Checker software. If no errors are found, the IODD Checker writes a checksum over the file content and Stamps it for further verification. This prevents any interventions to IODD.

Chapter 3

Analysis and design

3.1 Introduction to problem and motivation

During development of IO-Link Masters there are many situations, which require connected device and communication running. For FW developers it is immediate testing of their newly implemented features or bug fixes, HW developers sometimes require full operability for certain HW tests and testers required robust, stable and configurable device, which can be automatized and provides functionality to fully test all features of IO-Link Master.

There was clearly need for single device, which would fulfill needs of all mentioned parties. Device that provides feedback from device side to more easily find any potential errors on Master. Device that is robust enough to not being affected by unstable environment and that can be used during HW tests such as EMC (Electro Magnetic Compliance) or temperature tests. And device, that can be easily configured to change its parameters via some kind of communication protocol.

For this reason project was started, which aimed to create such device. Part of this project was collect every possible requirement from all interested parties to create device that can work as testing partner for every situation.

3.2 Collecting requirements

One of key phases on every project is correct collection of requirements from all stakeholders. Any vital requirement missed can bring a lot of problems in development. In case of HW development it can be need for

new prototype. In case of FW development it can be huge setback and sometimes even need for complete redesign of architecture. These problems can be completelly prevented by collecting all requirements, picking the ones that are going to be implemented in first versions and preparing for possible future integrations of those, that are left.

Whole requirements process was splitted up into several steps:

1. Identifying stakeholders
2. Collecting their requirements
3. Categorizing requirements into categories (e.g. functional and non-functional)
4. Setting priorities to tasks
5. Finalizing requirements

3.2.1 Stakeholders

Most stakeholder were already mentioned in previous sections, but this chapter will offer more detailed insight into their current situation and also their main issues linked to devices, which should be somehow diminished or get rid off entirely by this project. This covers only main 3 stakeholders, which are FW developers, HW developers and Testers. There are of course more stakeholder (e.g. management), but in this case requirements from their side are mostly generic and involves requirements, that apply to every other group as well.

3.2.1.1 Firmware Developers

Getting feedback during FW development of IO-Link Masters can be sometimes very complicated. Debugging communication errors with multiple devices connected and therefor thousands of interrupts per second is in some situations close to impossible and finding of bugs via different methods (e.g. using osciloscope) is not always ideal either. Also most devices, that are used for debugging are from different vendors and debugging from their side is therefor impossible.

Majority of available devices don't offer any insight into running communication and don't help at all with this issue and there are only few of the devices, that supports whole range of optional services that IO-Link offers. Also devices by specification don't have some communication specifics dynamic, so when developer wants to test some functionality, he

has to look for exact device, that supports this functionality. This also involves supporting of legacy protocol, that is not used in new devices, however io-link masters still have to be able to support it.

From these informations we can create several basic requirements for FW developers:

- Feedback from device side - This means either some form of logging of communication or online access to device to monitor status of incoming and outgoing data.
- Dynamic configuration - Some usually static properties (e.g. length of process data, baudrate) shall be changeable during runtime (brief interrupt of communication is expected)
- Support of legacy protocol - Old version of IO-Link protocol is not used in new devices and masters, however Masters still have to be able to maintain backward compatibility.
- Possibility of usage device as IO-Link communication logger - As it has been said there is not many ways to monitor communication between master and device. Test device should be able to be connected between master and desired device to monitor communication.

3.2.1.2 Testers

During IO-Link Master testing, there is always goal to test as many scenarios as possible, because to test all of them is impossible. To do this, tester would need to have all IO-Link Devices ever made, because every one of them is somehow different and offers different set of communication parameters and inside technology. Also IO-Link specification is sometimes not exactly specific on every possible feature, so different devices can behave differently during same operation.

To swap between all of those devices it is required for them to physically go and manually disconnect old device and reconnect new desired device and this prevents testers from some sophisticated automatization. Although this can be theoretically automatized, effort to do so is way too high. Testing on multiple devices will be always required, to ensure compatibility between most masters and devices, but for some form of tests (e.g. integration tests, regression tests) there is need for some automatized way.

There are currently not many ways to perform any robustness tests, since devices behave according to specification. Any defect has to be manually simulated on communication line and that is not always ideal and reliable.

Main requirements from testers are therefor:

- Dynamic configuration - Although this requirement was already from FW developers, its one of main requirements here as well. Device could be configured to substitute any possible configuration available.
- Easy control - Since testers are going to change configuration most frequently, there should be a way to do this as easily as possible. Probably with some form of online connection to device.
- Automatization - To integrate this into automatized tests there has to be a way to automatize changes of configuration as well. This can be done either via online connection, IO-Link service parameters or some different automatization.
- Simulation of communication errors - Device should be able to simulate errors in communication (e.g. wrong checksum, dropping frames, sending invalid data) to test robustness of IO-Link Master.

From FW developers and Testers perspective we can already make some basic use case design shown on figure 3.1. This involves standard usage and connection.

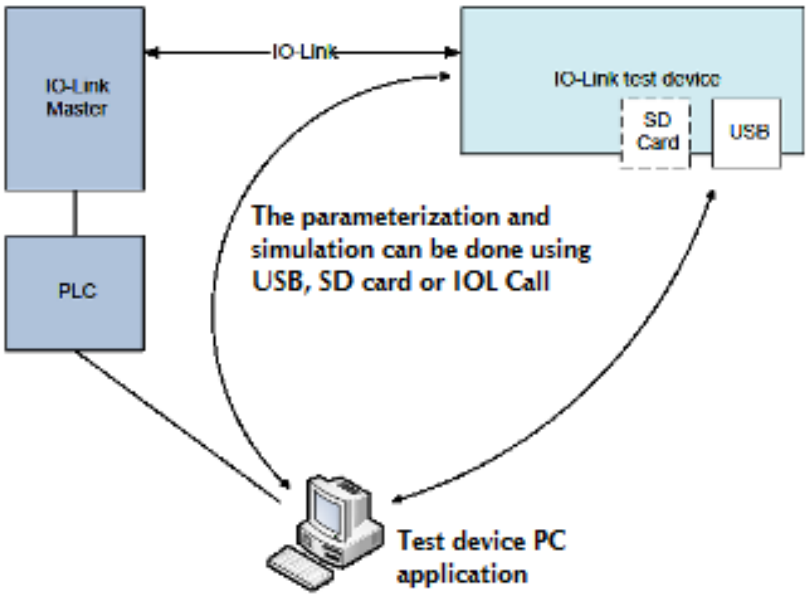


Figure 3.1: IO-Link Master test partner use case - standard use case for testing

3.2.1.3 Hardware Developers

Last of the 3 main groups are hardware developers. There are several tests (e.g. Electromagnetic compatibility tests), performed by Hardware developers, which need stable and robust device partner. Devices have to be able to maintain standard operability in unstable environment, which is quite usual in industrial surrounding. Hence there is major focus on robustness of such partner devices.

Another example of mentioned test is test for current load and ovearheating. This is usually done by connecting external current load to C/Q, L1+ and L2+ lines, because devices load usually can't be modified during runtime. This is not ideal way and could be done way easier by integrating modifiable current load into device.

From running HW tests there are several requirements:

- Robustness - Although this requirement should be mandatory by default in every device, it is not always the case in some devices. Therefore it is heavily stressed out in this test device.
- Current Load - Configurable current load on all used lines. Example can be seen on figure 3.2
- Power supply options - Device electronic shall be powered either from the L1+ power supply or from an external source (24V DC) with priority from external power supply.

3.3 Categorizing requirements

Requirements from previous chapter don't involve any generic requirements for device and don't go much into details. This chapter's goal is to properly categorize mentioned requirements and form them into technical requirements, that can be used as source for future development, list all generic requirements that haven't been mentioned yet and assign priorities for everything, so planning for release versions can be made.

Requirements are categorized into 2 main categories of functional and non-functional requirements. Funtional requirements specifies what the system should do. Any particular functionality shall be described here. After project is finished, it should be relatively easy to say if requirement is met or not. Non-functional requirements are requirements, that specify any performance, quality, usability or design subjects. Evaluating if requirement is met or not is not always straightforward like in functional requirements.

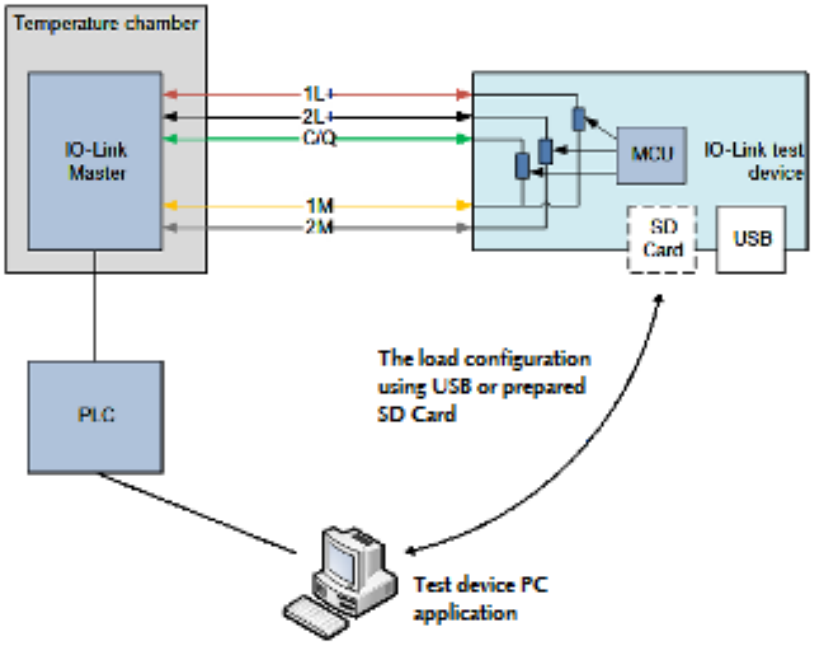


Figure 3.2: IO-Link Master current load test use case - standard use case for current load temperature test

Because there are more functional requirements, there are used additional subcategories. These will distinguish functional requirements based on their scopes and make chapter easier to take in. Subcategories are Basic functionality, HW requirements and User Interface.

Every requirment shall have its priority in future project. Most requirements are going to be implemented in first release, but there are several requirements, that are too difficult to integrate to first release and therefor shall be postponed to future versions and their implementation won't be part of this thesis.

3.3.1 Functional Requirements

3.3.1.1 Basic functionality

- **Protocol version** - The device shall be configurable as both V1.0 (legacy protocol) and V1.1 IO-Link device. This involves every functionality for both protocols specified by IO-Link specification [5].

- **Frame support** - The device shall support all frame types as defined

by the IO-Link specification [5]. This involves different frames for both versions of IO-Link protocol (current and legacy).

- **Configuration** - There are several requirements regarding configuration of the device:

  - **From PC application** - The device shall provide an USB configuration interface. The interface shall support configuration of the device as well as reading of status information and traces from the device.
  - **User application** - A PC GUI application for configuration shall be created. It will allow the user to easily enter and perform any service commands to read/write from/to the device.
  - **Configuration files (Prio 2)** - The PC application should allow to store the complete device parametrization into a file and reopen it later by application to store configuration is case of any issues. The file can also be stored to SD card and used for configuration from that card. These configuration files shall be in plain text format. (XML)
  - **Communication protocol** - A simple communication protocol between the device and the user PC application will be defined. The protocol will be based on read and write services with limited maximum message length. The maximum message length must be easily reconfigurable so that it can be adaptet to various interfaces (USB, data records, IOL-Calls). In case of big data transfer, the protocol must ensure fragmentation of the messages.
  - **Via IO-Link (Prio 2)** - The configuration protocol shall work also via the IO-Link interface - ISDU service. A special index in the device shall be created and used as an access point for the protocol messages. The device shall be able to communicate using both USB and IO-Link. This way of parametrization will be used in case of automatization from the PLC. The PLC can send the parametrization using a sequence of IOL-Calls.
  - **From SD card (Prio 2)** - The device shall be equipped by an SD card socket. If the card is plugged and contain new firmware or configuration file, the device will be updated/configure itself based on these files at the power up. The device shall also store its trace messages at the SD card.

In first release version there is priority at least on some basic way to communicate with device. Most usable is USB and therefor there is low priority on other ways of communication.

- **IO-Link sniffer (Prio 2)** - The device shall have the functionality to be used as IO-Link sniffer and monitor communication between IO-Link master and different device when connected inbetween.

- **Traces (Prio 2)** - The device shall store the trace message according to the configured trace level at the SD card. The traces can be displayed in the PC application. It can be uploaded to the application either directly from the SD card plugged into the PC reader or using the configuration interface from the test device.

3.3.1.2 HW Requirements

- **IO-Link connectors** - The device shall have two IO-Link M12 A-coded connectors - one female, one male. They will be interconnected with each other. This setup allows implementation of the IO-Link sniffer in the future.

- **Galvanic isolation** - The complete galvanic isolation of the IO-Link connector (5-wires) shall be provided for usability for EMC measurements. The 2L+ potential shall be galvanically isolated from the 1L+ potential and the device elctronic.

- **Automatic Load**

  - **1L+ Load** - The 1L+ signal can be loaded in range ¡0; 1 A¿ with voltage range up to 30V. The load current will be controlled by the MCU. The maximum current should be designed for 4 A, in case of external cooling.
  - **2L+ Load** - The 2L+ signal can be loaded in range ¡0; 4 A¿ with voltage range up to 30V. The load current will be controlled by the MCU. The maximum current should be designed for 8 A, in case of external cooling.
  - **C/Q Load** - The C/Q signal can be loaded in range ¡0; 1 A¿ with voltage range up to 30V. The load current will be controlled by the MCU. The load circuit can be completely disconnected from the C/Q signal to avoid disturbances in normal use.

- **Power supply options** - The device electronic will be powered either from the 1L+ power supply or from an external source (24 V DC). The device electronic must be able to automatically adapt on the current power supply source. If the external power supply is available, it shall be used with higher priority than the 1L+ sensor supply from the master.

- **Master - slave communication (Prio 2)** - Master-slave communication channel between more devices shall be defined as a back plane bus. Master device should have access to the PC application by the USB connection and spread incoming requests to other devices on the bus.

- **Digital input and output ports** - The device shall have several input and output ports. Both these parts shall be galvanically isolated from internal electronic. Digital inputs/outputs can be used for synchronization the test run with other equipment like PLC or oscilloscope.

- **Module UID (Prio 2)** - Every module shall have unique identification number for case of serialization of the modules. This shall be realised via 4 bit switches that can be set to desired value.

3.3.1.3 User Interface

- **Connector interfaces** - The device shall have microUSB port, SD card (Prio 2) socket and 2 M12 A-coded connectors (one female and one male). The device shall also have a connector for external 24 V power supply.

- **Port LEDs** - The device shall have the standard red and green LEDs at the port as described in IO-Link specification. Further the device shall have two green LEDs to monitor the availability of 1L+ and 2L+ power supply. Additional green LED will monitor the availability of the external power supply.

- **Display** - The device shall have a simple 2 or 4 row LCD display, which will be used for displaying user information. It can be used for example to display the configuration setup, status information or traces..

- **Control buttons** - The device shall have 4 buttons to navigate through display and to control various functions.

3.3.2 Non-functional requirements

3.3.2.1 Performance

- **Cycle time** - The device shall support all allowed cycle time values with some minimum limit given by the implementation constraints. The limit should not be more than 1 ms. current should be designed for 4 A, in case of external cooling.

- **Cooling** - The device shall have sufficient cooling to prevent from overheating when current loads are used. Cooling has to be solved by with passive techniques.

- **Startup** - The device initialization and startup shall be withing 300 ms to be useful during power restarts tests.

3.3.2.2 Quality

- **Reusability** - The device shall be developed with techniques, that allow reusability of code for different projects and preserving know-how. This involves detailed documentation of development process.

- **Maintanability** - The device shall be developed with techniques, that allow easy maintanability and integration of new funtionalities. This involves usage of internal guidelines for development and usage of various design patterns for programming.

3.4 Planning

With specified requirements we can divide project into 3 main parts:

- **Hardware design** - Although development of firmware and communication can start without funtional prototype, it's only temporary solution. Therefor delivering funtional prototype is highest priority.

- **Communication protocol and PC application** - This part includes developing communication protocol, PC application and creating communication drivers, that could be easily integrated into firmware.

- **Firmware development** - This includes application core, that handles communication with IO-Link Master and internal behavior. It is also linked to both hardware design and communication protocol.

It was mentioned before this project is mainly done by students under supervision of senior developers. This diploma thesis is mainly focused on Firmware development part, both Hardware design and Communication protocol and PC application is made by another students, where Communication protocol and PC application is different diploma thesis [10].

Because Firmware development is tightly linked to other 2 parts, in future chapters there is going to be brief description about cooperation from firmware side.

3.5 HW Selection

There are two pieces of hardware important for Firmware developers of IO-Link Device. It is mainly controller, which needs to be powerful enough to run everything fast enough to fullfil Time Cycle requirement, and IO-Link Transceiver (PHY), that is used to convert 3.3 V UART signal to 24 V on C/Q line and to help with detection of Wake-up signals. Only difference for HW designers is different size of packages and different layout and number of pins. Selection is therefor mainly task for Firmware developers.

3.5.1 Micro-controller selection

The selection of micro-controller for IO-Link Test Device was quite difficult process. There wasn't many requirements to fullfil, but large number of usable hardware solutions on the market made the choice quite complicated. It was required to find a controller, that can make all requirements mentioned in previous chapter possible for best price and while consuming least space on board.

3.5.1.1 Requirements for MCU

Even if project is successful in future, there probably won't ever be any mass production. For that reason price is not among highest priority requirement, although it is still best to choose MCU as cheap as possible.

Requirements for MCU, that ensue from global requirements for device are therefore:

- **ARM Cortex architecture** - Because of most know-how and experience with this architecture this is one of the requirement.
- **At least 2 SPIs** - For communication with external Flash memory, current load controllers, display and backplane bus in future there is need for SPI peripherals. This include extra GPIO pins for chip select pins as well.
- **Enough GPIO pins** - MCU shall have enough GPIO pins for any funtionality described in global requirements for the device. It is neede for control of LEDs, Digital inputs, Digital outputs, UID, buttons and any possible diagnostic pins from used peripherals. Hence minimum of 32 GPIO pins is required.
- **UART** - Although UART is quite common in almost every available MCU, it is required for it to support everything that is used for IO-Link communication. (e.g. even parity, 9b length,...)
- **USB support** - Among peripherals USB is one of the most difficult ones and not many MCUs are supporting it. For basic funtionality it is needed the support of full speed USB device mode.
- **SD card support** - Although SD card has priority 2 in development, everything needs to be prepared for it. Therefor MCU shall support SD card functionality as well.
- **External crystal support** - For least number of communication errors on UART, external crystal with frequency of 11.026 MHz is used. MCU has to be able to use this crystal as system clock.
- **Sufficient RAM and internal FLASH size** - Because of many funtionalities on the test device there is needed sufficient internal FLASH size to store the source code and sufficient RAM size to actually run it and keep track of information like traces. Minimum of 256 Kb of Flash and 128 Kb of RAM is needed.
- **Sufficient power performance** - To maintain IO-Link communication on highest speed COM 3 and simultaneously maintain functionality of other services there is low bound for performance. The MCU shall be able to support at least 48 MHz.
- **JTAG or SWD interface** - For debugging there shall be support of at least JTAG or SWD debugging interfaces.

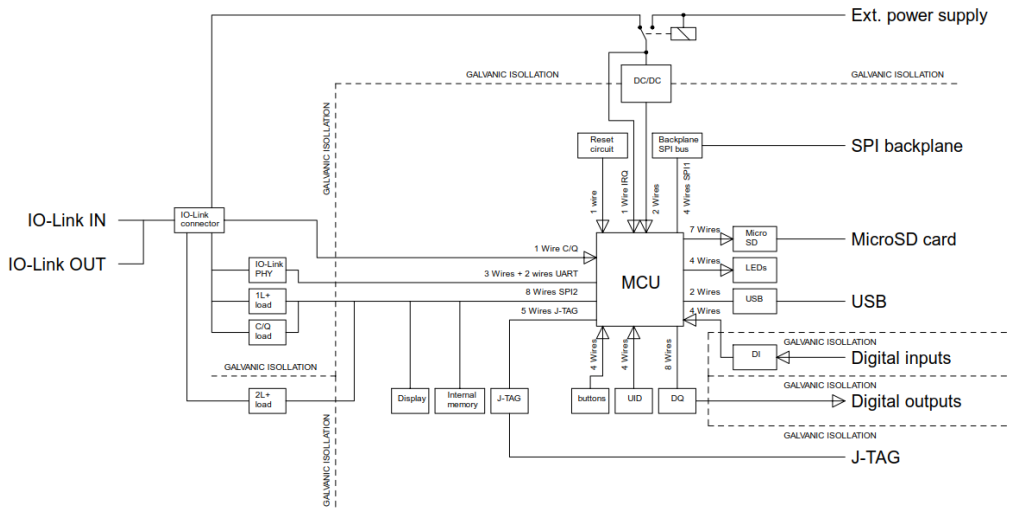


Figure 3.3: Basic Scheme - Possible scheme of all parts and their connections to MCU. Also basic estimation for required number of pins on MCU.

Possible layout of board is shown in figure 3.3. MCU has to be able to support every mentioned function.

3.5.1.2 Viable candidates

As it has been mentioned there are several manufacturers, that make MCUs. Their products comes usually in form of product lines, with same architecture and similar features. Particular products in product lines then varies only with things like package size and layout or memory size or extra peripherals.

There are many viable candidates where we can choose from so the search was reduced to manufacturers that are known to Siemens and that could guarantee trouble-free partnership. Among reduced manufacturers there are these product lines:

1. **STM32L1** - Product series based on ARM Cortex-M3 architecture with ST's proprietary ultra-low-leakage process technology focused on low power consumption [11].

2. **STM32F1** - Standar series of mainstream MCUs with ARM Cortex-M3 architecture. This series covers basically everything in standard design and can be used for large variety of applications [12].

3. **STM32F4** - High-performance MCUs with ARM Cortex-M4 architecture. This series offers not only highest operating frequency, but also highest RAM and FLASH sizes [13].

4. **Freescale KL1x** - General purpose ultra-low-power MCU family based on ARM Cortex-M4. As all ultra-low-power MCUs doesn't provide much RAM, since it's most power hungry part of MCU.

3.5.1.3 Chosen MCU

Although there have been few more MCU's manufacturers and product series to choose from, there was from beggining high preference for MCU from ST. This is mainly because ST provides very well support for their products, there was large know-how regarding ST's MCUs and also there was largest supply to choose from. Only reason to look for different manufacturers was to possibly find something, that would surpassed ST's advatages and that didn't happen.

The only decision making in the end was therefor between STM32F1 and STM32F4 series. Because there was never plan to make this project into mass produced device there was focus to focus on quality rather than quantity. For that reason STM32F4 series was chosen. Particularly MCU STM32F407VGT [14].

Final properties of STM32F407VGT are:

- 1 MB of Flash memory
- 192 KB of SRAM
- Up-to 168 MHz core frequency
- Both JTAG and SWD debug interfaces
- 100-pin package
- 4 USARTs, 3 SPIs, USB 2.0 with full-speed device with on-chip PHY
- Up to 100 I/O ports with interrupt capability

3.5.2 PHY selection

Another important part of HW, that was influencing future FW development was IO-Link Phy. Although every available products support in some way basic IO-Link communication, wake-up detection and whole series of diagnostics, there are minor differences, that set particular products apart (e.g. way of configuration).

3.5.2.1 Requirements for PHY

As has been said in previous paragraph, every available product fullfil basic requirements for PHY. For clarification all requirement will be recapitulated anyway:

- **Support of COM1/2/3 modes** - This is mandatory requirement for basic IO-Link functionality, where transceiver has to be able to communicate in all defined baudrates.
- **Output stage modes** - Output stage shall be configurable as high-side, low-side or push-pull. This is mandatory for several HW tests.
- **Wake-up detection** - PHY has to be able to properly identify wake-up pulse and signal it to MCU via defined signal.
- **Small package** - Package shouldn't be larger than 5x5x1 mm.
- **Basic protection** - Protection from reverse polarity, overtemperature or undervoltage/overvoltage is a must. PHY also has to be able to detect these events and signal them to MCU.

3.5.2.2 Viable candidates

As in MCU selection, there are multiple vendors, that offer IO-Link device PHYs, although there is by far not that large variety of products. Considering PHYs are meant to do the same thing, which is convert 3.3 V or 5 V UART signal to 24 V IO-Link signal, there are not many major differences in particular products. Minor differences are usually in a way of configuration, operating temperature range, protection or package size and layout. Suitable candidates are these:

1. **MAX14820/14821/14826** - These 3 PHYs from Maxim integrated differ only in current load/sink on C/Q line, where least is on MAX14821 with only 140 mA and most is on MAX14820 with 375 mA. Everyone of these PHYs is in 4x4 mm, 24-pin TQFN package. Configuration and monitoring is done via SPI. Alarms are signalised through interrup outputs. Standard protection involves reverse-polarity, short-circuit and thermal protection. Also all lines are monitored for undervoltage conditions.[15]
2. **ST L6362A** - With size only 3x3 mm and 12 pins this PHY is smallest one of all. Also as only one of candidates offers surge protection.

With standard protection like reverse polarity, overtemperature, overvoltage and overvoltage protections this is major advantage. Although small size is advantageous as well, due to small number of pins there is no SPI configuration feature. Very minimal design offers only few control pins, that can set operating modes and 1 output pin, that signal diagnosis on PHY. There is also internal 5 V or 3.3 V, 10 mA selectable linear regulator, for power supply of PHY as well as a local controller and additional circuits. [16]

3. **TI SN65HVD101/102** - Standard PHY with basic features and protections in 4x3.5 mm 20 pin QFN package. As in ST PHY, there is no real configuration done, since everything can be real-time controled via control pins. SN65HVD101 has internal linear regulator, that generates either 3.3 V or 5 V from IO-Link L+ voltage.[17]

As in case of MCUs, there are more IO-Link PHY vendors, but the choice was made from vendors, which are known to us and which products are already being used in known Siemens products.

3.5.2.3 Evaluation process

Because in our development team there was at the time no know-how about devlopment of IO-Link device and know-how about IO-Link Masters, although was helpful, couldn't be used on everything, most of the work had to be done from a scratch. Selection of desired candidate for PHY was therefor done by testing every single one manually and evaluating usability, advantages and disadvantages directly.

We had available development boards for MAX14826, ST L6362A and TI SN65HVD101 PHYs and ST32F4DISCOVERY kit with STM32F407VG MCU. With these and basic firmware, that managed initiating communication and maintained exchange of process data, the basic evaluation had been done. During evaluation there was focus on evaluating difficulty of configurating PHY into operating mode, simplicity of detection of wake-up pulses and reading of diagnosis.

3.5.2.3.1 MAX14826 Because this PHY was only with configuration done via SPI, there was additional effort from the beggining. Number of required pins for complete funtionality is high compared to other PHYs. This is rewarded by most sophisticated diagnosis service among these 3 PHYs. There are several registers to control/monitor PHY funtionalities - Status, CQConfig, DIOConfig and Mode. Wake-up detection is signaled via dedicated pin and bit in Status register.

3.5.2.3.2 ST L6362A This minimalistic PHY was easiest to make work. Its design requires only 5 dedicated pins from MCU to fully operate. Contrary to MAX Phy, when any diagnosis is detected, the dedicated pin is asserted, but there is no additional info about what exact diagnosis was detected, only that there is one.

3.5.2.3.3 TI SN65HVD101 This PHY is mixture of previous 2 designs. There is no SPI configuration as well, but diagnosis are more detailed than in STs case. Compared to ST, there is not 1, but 3 pins, that signalise diagnosis status. This can distinguish events between temperature, current and power ones.

3.5.2.4 Chosen PHY

In the end, there were several factors that influenced result of selection process. Because there is already selected MCU from ST, there was advantage from having PHY from same vendor as well. Minimalistic design, most simple usage and extra surge protection of PHY were major advantages as well. For that reason ST L6362A was selected. It's main properties are:

- 5 V or 3.3 V, 10 mA selectable linear regulator
- Fully protected - Reverse polarity, Overload with cut-off function, Overtemperature, Undervoltage and overvoltage, GND and VCC open wire
- -40 to +125 C operating ambient temperature
- Selectable output stages: high-side, low-side, push-pull
- Wake-up detection supported
- Miniaturized VFDFPN 12L (3x3x0.90 mm) package

Chapter 4

Realisation

4.1 Overview

Before there can be any FW development process, there has to be general idea about structure of desired application, what are the main parts of it and how exactly are they going to be linked. Without this, there are potentional problems during development, where some features may be difficult to add in later stages and it could cause beggining of unmaintainable code, which can sometimes be reffered to as "spaghetti code".

On the other hand, analysis should not be overrated and should not be done to details, because doing so is time inefficient and some details cannot be thought during analysis. There should be therefor certain balance of analysing structure and details of this project.

Basic deployment overview can be seen in figure 4.1. In this picture it is shown how every connected parties communicate during standard operating mode.

4.1.1 Used components

During the development, several software applications, programs and hardware components were used. The firmware is written mainly in C++ with minor use of C. This supports usage of advanced functionalities, that C++ provides (e.g. inlining, overloading, templates) and design patterns, that help with keeping source code clean and structured. Eclipse Mars [18] was used as text editor and Siemens internally configured IAR compiler [19] for embedded device programming for compiling the source code.

Before HW prototype was done, STM32F4Discovery board with STM32F407VG MCU was used for FW development along with ST

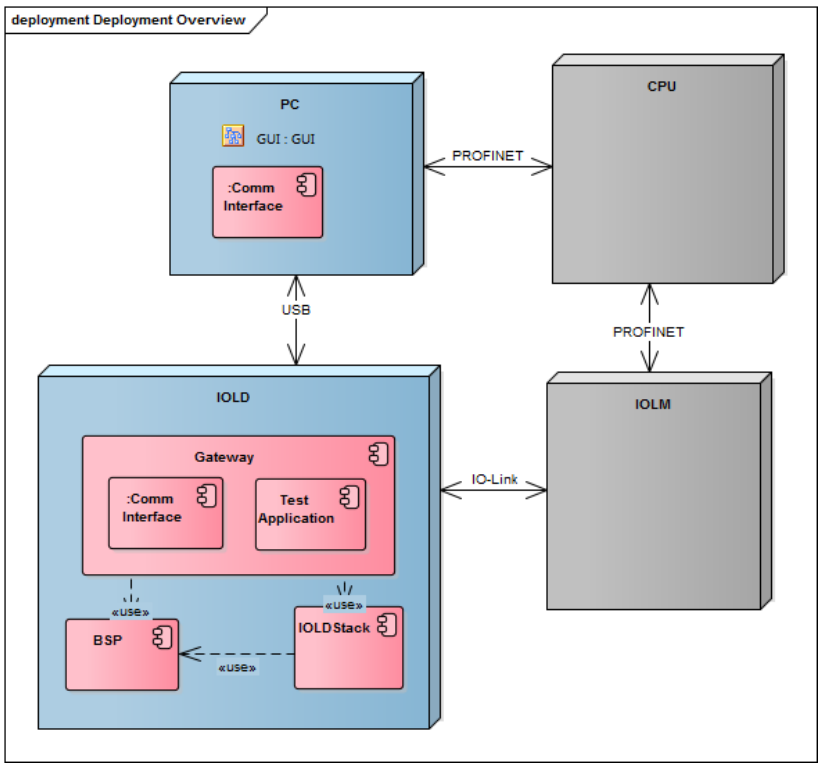


Figure 4.1: Deployment Diagram - Diagram shows basic structure of all main components during standard operation mode of IO-Link Device.

L6362A PHY development board. This could've been used only on early development and basic funtionalities, because most of the advanced one services required HW presented on final board.

All kinds of architecture diagrams were created with the visual modeling and design platform Enterprise Architect from SparxSystems, which is usign UML2.0 for modeling [20].

Everything was also backed up and changes were recorded via Version Control System (VCS) named ClearCase by IBM [21]. This software allows creating multiple development branches, where more developers can work at the same time. Upcoming merging changes is mostly automatic with only few needed interference from user.

4.2 Basic Design

Whole structure will be assembled out of several main parts. Parts that are structured around functionality into separated and isolated pieces, that

communicate with each other only by dedicated interfaces. From lowest to highest level of application the main parts are:

1. **Low level drivers** - Lowest level of drivers, that directly access to registers of MCU and control all kinds of peripherals and internal MCU settings. These drivers are generally provided by MCU vendor, but can be made based on MCU datasheets. In this project, the drivers provided by ST are used.

2. **Board support package (BSP)** - A board support package is an interface between lowest level drivers and higher level application. In this case it shall be designed in a way that it provides abstract layer. This way if MCU is changed and different drivers are used, whole application remains intact and only BSP part has to be modified.

3. **IO-Link Stack** - IO-Link stack is heart of IO-Link communication. It controls everything from reading incoming packets and assembling them into frames or managing events and process data, to assembling replies and keeping all constraints of IO-Link specification. In this project there is used external IO-Link stack from TMG.

4. **Gateway application** - Core application, that manages every funtionality. Internal communication between classes in gateway application have to be designed in a way, that it can't result in chain command, that takes too much time, that could interrupt IO-Link communication.

5. **RTOS** - Real time operating system could be implemented to help with time management of both internal and external communication. There are multiple constraints from IO-Link specification, that commands certain time windows for reply and RTOS could help to solve them. However everything needs to be implemented in a way that it could work without active RTOS to be more versatile for future development, where RTOS couldn't be used for some reason.

Whole possible design can be seen in figure 4.2. In picture there are shown basic dependencies between particular parts as well as main classes (and parts in case of C written drivers) of whole structure.

4.2.1 External parts

As has been said in previous section, parts of this projects are already done or will be done by different authors. This include mainly Low Level

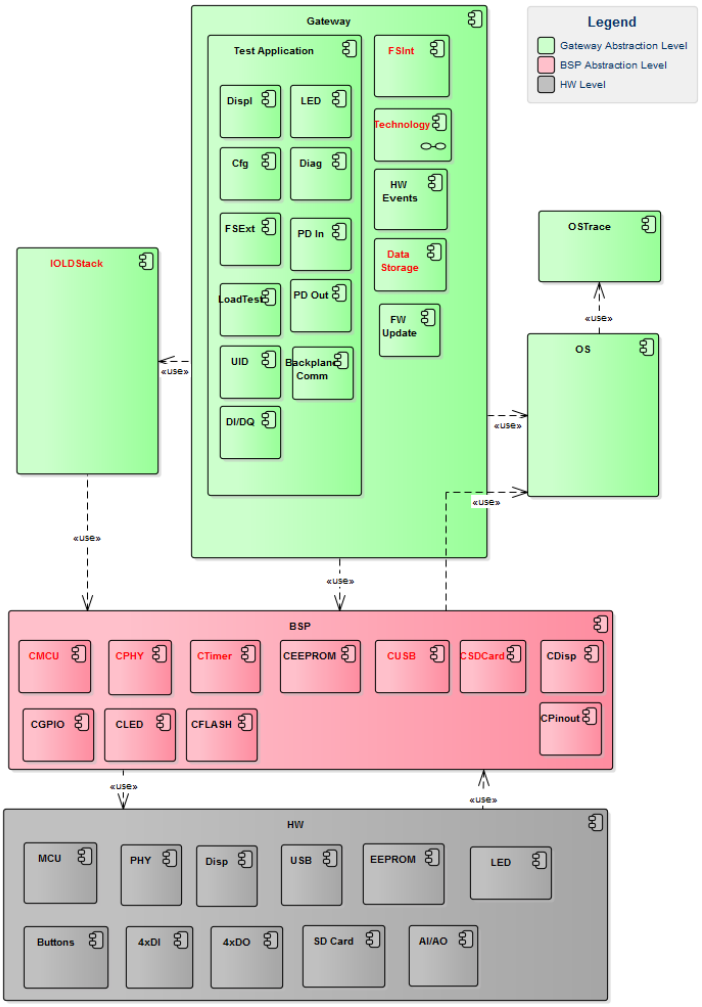


Figure 4.2: Component Overview - Distribution of funtionalities into particular parts.

Drivers, IO-Link Stack and RTOS. Also USB part of BSP will be made by different master's thesis done by another student [10]. This however still leaves integration of these parts into working unit, so in future chapters this will be explained into detail.

4.3 BSP

ST provides currently two types of drivers. HAL drivers, that offers basic abstraction and basically works as BSP, and then low-level drivers, that

are just basic functions for writing and reading registers. However if one of these were used in whole application, there could be major problem in case of reusing these components in future with completely different MCU. Everything would have to be completely rewritten.

For this reason BSP component was designed in a way, so any changes of MCU could be done without necessary changes to whole code. Only small behaviour parts of BSP components need to be changed to reflect changes to MCU.

As shown before in figure 4.2 the BSP component is assembled from several classes. These classes are divided into 3 main categories:

- **BSP** - This involves 2 classes. CBsp and CBspPinout. CBsp encapsulates basic operations required for basic functionality of communication. This involves initialization of all behavior components and peripherals and access to basic functions, that are required for proper funtionality of IO-Link Stack (e.g. Timer handling, UART handling, interrupt disable/enable). CBspPinout contains configuration of all used GPIOs and peripherals and provides getters to access them.

- **MCU** - This includes all peripheral components and also CMcu class, that encapsulates behavior settings of peripherals and provides access to peripheral functions. Main used peripheries are GPIO, UART, SPI and Timers, but can be easily expanded with any currently unused peripheries (e.g. I2C)

- **PHY** - This includes class CPhy that provides abstract interface and define strategy for currently used PHY. This is used for configuration and change of operation modes of PHY.

4.3.1 BSP

4.3.1.1 CBsp

CBsp class was designed to encapsulate functions for IO-Link stack component. IO-Link Stack need for proper funtion access to UART, Timers and Interrupts and CBsp provides interface to access it. Specifically it is:

1. **Start UART** - Because UART is not active by default, IO-Link Stack need to have way to activate it once wake-up pulse is received. After that UART is activated with proper baudrate.

2. **UART Send** - Sending data is through UART is only way to communicate with Master. For this there is function SetTx, that is repeatedly called and send 1 byte of data through UART.

3. **Enable/Disable PushPull** - This functions are to control modes of PHY. By default PHY is prepared for receiving Wake-up pulses and has active either high-side or low-side switch. For communication there has to be disabled PushPull mode and switch PHY to high impedance state for receiving data from Master.

4. **Enable/Disable Interrupts** - In some communication situations IO-Link Stack must not be interrupted. To prevent any interrupts from triggering, there need to be way to momentarily turn them off.

5. **Timer start/stop** - IO-Link Stack is using various timers for keeping track of time constraints of responses and fallback situations. For this timer is needed. Timer functions start and stop are therefor provided.

6. **Stack Fallback** - In case of interruption of communication or fallback command, there is provided Stack Fallback function, that immeadiately deactivate UART, activate wake-up detection and reset status of device to initial state.

4.3.1.2 CBspPinout

Second main class of BSP category is CBspPinout. This class encapsulate any specific information about pinout of Mcu and configuration of all used peripheries. Also stores behavioral settings for mentioned peripherals. Reason for this is to keep every product specific configuration in one place.

4.3.2 MCU

MCU category in BSP component is focused on handling all available peripherals. It is divided into several classes. Main class is CMcu, which provides interface to every available operation and handles behavioral settings, which will be described in upcoming sections of this chapter. Then there is class for every peripheral.

General method of implementation is going to be presented on CGpio class. CGpio class is used for complete configuration and control of all GPIO pins. There are several other classes for other peripherals (e.g. CSpi, CUart, CTimers...), which will not be described into details in this, because their implementation is very similar to CGpio class.

4.3.2.1 Usage of Strategy Pattern

Every MCU family has usually different registers and therefor using different drivers, there needs to be some kind of behavioral settings to specify used approach. For this reason strategy pattern is used.

Strategy pattern defines a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it. It also capture the abstraction in an interface and bury implementation details in derived classes [22]. Design of such strategy pattern can be seen in figure 4.3.

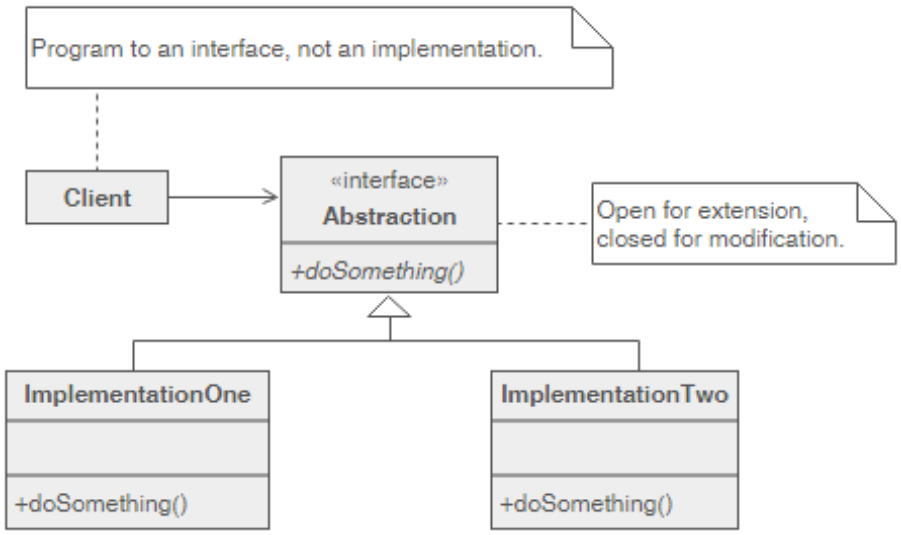


Figure 4.3: Strategy Pattern - Basic design of strategy pattern - Abstraction class, that provides interface to client (application) can change its behavior based on assigned strategy. This assignment must be done before first use and can be changed during runtime.

In this case strategy is used for encapsulating behaviours for every usable MCU and PHY. During initialization the right behavior is selected and its functions are afterwards used to control low level drivers. Although strategy pattern is usually designed in a way, that behaviors can be changed during runtime, in this case there is no such use and behavior stays same after initialization (since user can hardly change MCU/PHY as well).

As has been said, examples are going to be shown on CGpio class. In figure 4.4 it is shown how strategy pattern works in case of CGpio class. Main interface function provides interface to the application while behavior

funtions override those funtions and using private funtions transform abstraction into data usable by particular MCU.

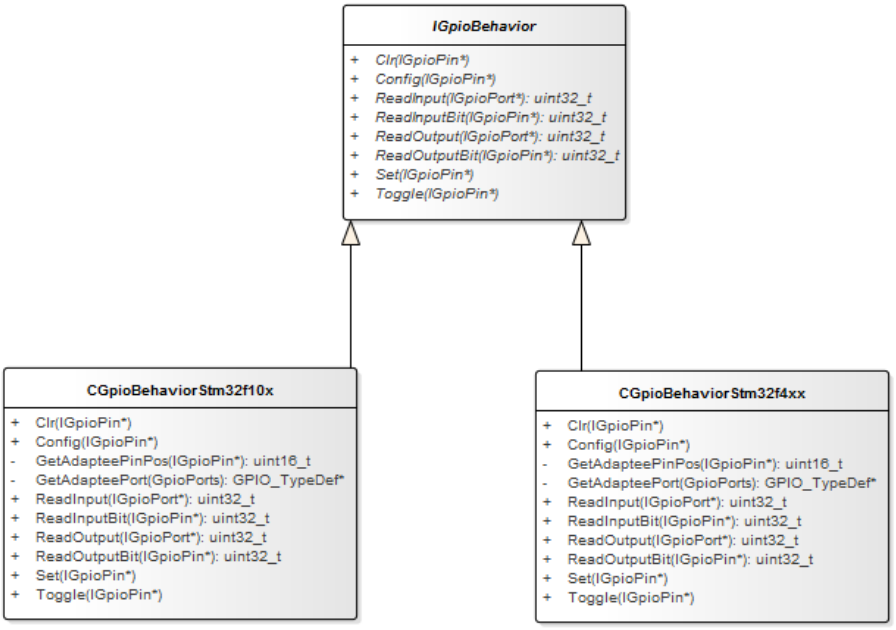


Figure 4.4: Strategy Pattern of GPIO - Implementation of strategy pattern on GPIO example. Interface class offers interface for rest of application, while behavior classes inheriting and overriding those funtions in a way to properly use low level drivers for particular MCU.

4.3.2.2 Abstract used for MCU peripherals

In abstract part of BSP, there have to be stored all possible configuration parameters of particular peripheral. These abstract parameters are then transformed in behavior part into data, that are understandable by low level drivers. Example of such abstraction can be seen in figure 4.5. CGpioPort, CGpioConfig and CGpioPin are designed to support every possible feature GPIO can have. Even if there would be feature not yet support in future, it can be easily added into these classes and there won't be any influence on application at all, because application is accessing only interface functions from IGpioBehavior class.

There are several other packages, that covers all used peripherals. Main packages are for GPIO, UART, SPI, Timer and USB. These main peripherals helps to control every feature currently available on device. If

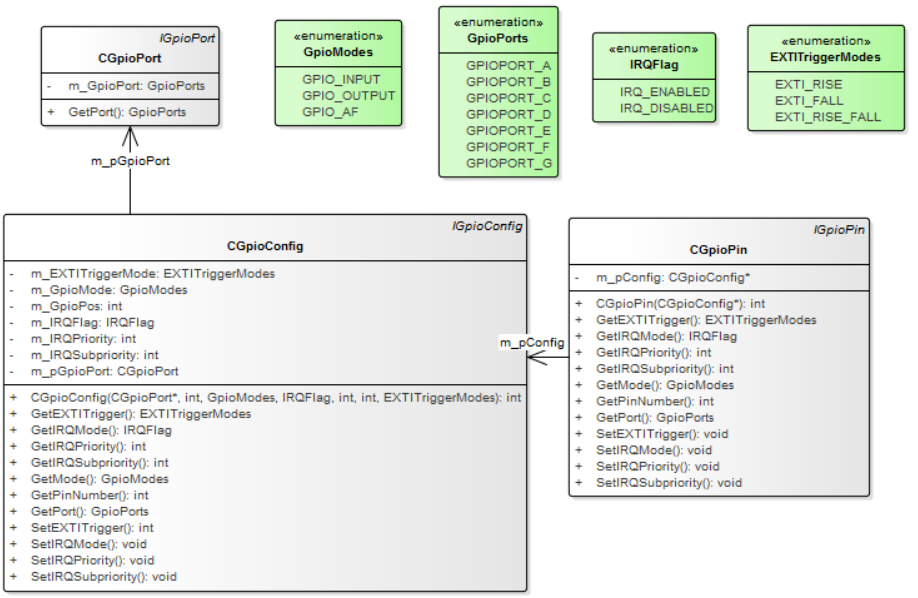


Figure 4.5: GPIO Abstraction - class design for GPIO abstraction. CGpioPin class provides interface for behavior class to access every needed information about GPIO pin and port it is located on. With these informations and informations about GPIO mode and possible interrupts behavior class can configure GPIO pin.

there is need to use peripheral, that is currently not implemented, the design is made in a way that there is easy extension and adding another peripheral class is therefor very simple.

4.3.2.3 CMcu class

Although there has been described how particular peripheral parts of BSP works, there hasn't been described yet the way how application can access these functions and how the behavior is actually set during initialization. For this reason MCU singleton[1] class was designed. This class encapsulates behavior settings and offers interface to access them.

During initialization, every peripheral interface classes in MCU class have to be initialized with specific behaviors for used MCU. After this initialization, MCU offers interface funtions to these behavior classes and serves as main access point for any peripheral manipulation.

[1]Singleton is basic design pattern, that ensure a class has only one instance, and provide a global point of access to it. Initialization is done automatically on first use. [23]

4.3.3 PHY

Last part of BSP is CPhy class. This class is designed to operate product specific PHY. For this there is used same strategy pattern as in case of peripherals. Change of PHYs therefor doesn't influence rest of application.

Because ST L6362A was used as PHY, there currently aren't many functions defined in CPhy class, since ST L6362A has very minimalistic design and don't need any difficult configuration or handling. CPhy is therefor controlling only disabling/enabling Push Pull mode on PHY.

4.4 Stack integration

IO-Link Stack is basically Application Layer of IO-Link, described in 2.1.4. In this device Stack from TMG company has been used, because developing completely new stack would be redundant, since TMG IO-Link Stack was available [24]. Desired integration of TMG IO-Link Stack can be seen in figure 4.6

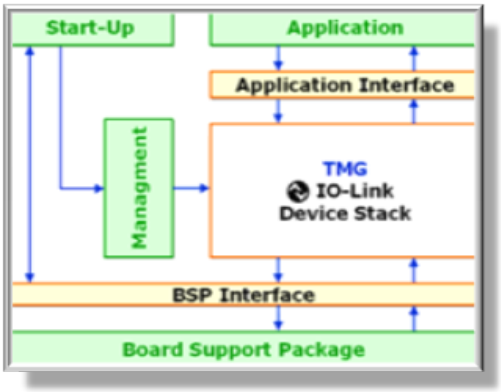


Figure 4.6: Integration of TMG IO-Link Stack - As can be seen there are 2 neede interfaces. BSP interface has been already described in 4.3.1.1.

Because only BSP Interface is needed for communication with IO-Link master, integration of IO-Link Stack has been done before actual development of Application core. Although absence of Application part made any communication pointless, because there weren't any process data, events nor any responses on possible requests from IO-Link Master, basic communication could've been initiated and that created working background to build upon.

Basic requirements on BSP from IO-Link Stack are:

1. **Interrupt for wake-up detection** - Once PHY detect wake-up pulse, there is short pulse on dedicated pin, that needs to be intercepted. For this interrupt routine is used and interrupt starts wake-up procedure, that activate UART peripheral.

2. **UART Rx Interrupt** - Once UART has been activated, IO-Link Device listen on UART channel for startup sequence, which consists out of several reads of parameterization. Every received byte from UART therefor raise UART Rx Interrupt, that invoke receive routine from IO-Link Stack.

3. **Timer with interrupt** - After every response from IO-Link device, there is timeframe for another byte from Master. This timeframe must be measured somehow. For this, configurable timer is used and periodically started/stopped based on specific timer values.

4. **UART Tx** - To send any reply, UART Tx is needed as well. Because of nature of PHY, where there is one channel used for both incoming and outgoing frames, sending automatically raise Rx Interrupts as well. This is expected behavior though, and Stack is using these interrupts to monitor correctness of sent data.

Last part of TMG IO-Link Stack integration was proper configuration. For this, TMG IO-Link Stack provides dedicated configuration file, in which every configurable variable can be set. Things, that has been configured are:

- **Static/Dynamic parameterization** - Because based on specification, parameterization is unique for every device and must not be changed, by default there was static parameterization set. TMG IO-Link Stack provides funtionality to set these parameters during initialization, without any possibility to change it afterwards. However one of requirements for this device is possibility of changing this parameterization during run-time. For this Dynamic parameterization was chosen with extra modification, which automatically interrupt communication with master in case of changed parameterization during run-time. Communication than has to be reestablished and new parameterization is used.

- **Timer settings** - To properly set timers, there was need to exactly define frequency of used timer and its configuration mode.

- **Timer constraints** - There is possibility change timer constraints for communication. This can be set in range defined by IO-Link specification.

4.5 Gateway application

Gateway component is core of the test device. It is responsible for majority of FW functions. This involves:

- **Mail system** - Communication between all classes inside Gateway must be designed in a way that messages don't create chain events, that could cause interruption of communication. This is solved by usage of Mail system.
- **Configuration** - Maintain configuration settings and invoke communication reset in case of configuration change to manifest this change to IO-Link communication.
- **Process Data** - Contain both incoming and outgoing process data, which can be used for various test functions (e.g. virtual loopback, physical loopback via DI/DQ).
- **DI/DQ** - This funtionality provides interface to control 4 Digital Input and 4 Digital Output pins. This can be interconnected with Process Data components to create physical process data loop.
- **Events** - Gateway administer events manipulation. This involves keeping track of all active events and managing their reporting to IO-Link Master.
- **File System** - File System mainly stores all non-volatile data to on-board flash. At the time this involves mostly configuration settings.
- **LEDs** - This functionality is designed to control all present LEDs. Without much effor it is possible to set LED on or off or create any kinds of blinking.

4.5.1 Mail System

Internal communication between particular classes can be quite difficult in real-time environment. If classes would directly access other classes

interfaces to immediatelly make their requests, this could lead to chain event of requests, that would take too long to accomplish and could jeopard time sensitive communication, that needs to be active periodically.

For this reason some communication system had to be designed. The goal of this system was to prevent communication threatening chain requests between classes from happening. Design pattern Publisher-subscriber was chosen for this task.

4.5.1.1 Publisher-Subscriber pattern

Core of publisher-subscriber pattern consists from publishers and subscribers how name suggests. Its main feature is sending mails (messages) only to components, that are interested in such mails. This is caused by filtering mails based on their source and contents. [25]

During initialization all components, that wants to receive certain mails, register to main publisher class CMailPublisher. During registration it is specified what kind of mails are they interested in. When CMailPublisher class publishes mail, it goes through all subscribed classes and either send mail to them or not, based on their mail filter settings. Infrastructure of publisher-subscriber pattern can be seen in figure 4.7.

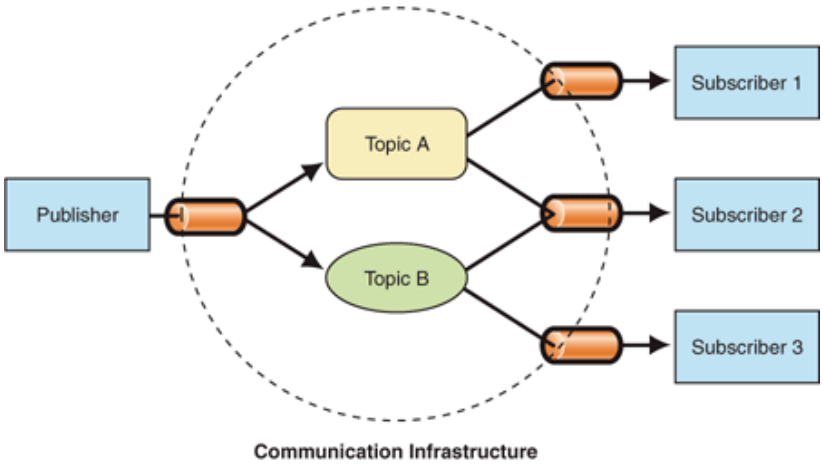


Figure 4.7: Publisher-Subscriber pattern - Subscriptions to topics control the mail types that reach each subscriber.

4.5.1.2 Mail Queue

Usage of publisher-subscriber pattern would not be enough by itself. It needed to be combined with proper batching management, that would not flood system with multiple mails at once. Without such management, publisher-subscriber pattern could just keep publishing any incoming mails and there wouldn't be any difference compared to direct approach of classes.

For this situation CMailManager class combined with standard FIFO queue system was designed as can be seen in figure 4.8. CMailManager is periodically called from main loop or designed RTOS task to publish one or few mails. This approach prevents mails from flooding the system. Only thing that needs to be cared for is frequency of publishing mails. In case of low frequency message queue could overflow and mails could be lost in the process.

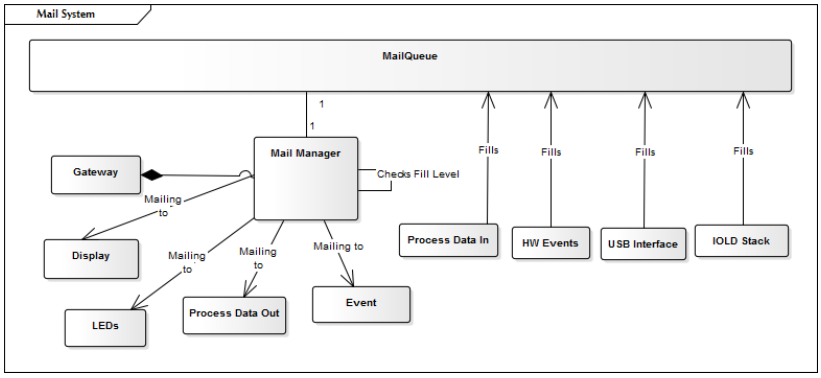


Figure 4.8: Mail Queue - All incoming mails are stored in Mail Queue. MailManager class takes every publish iteration 1 mail and sends it to subscribed classes based on their filters.

4.5.2 Configuration

Configuration functionality is provided by class CConfiguration, which stores whole parameterization of IO-Link device. Stored parameters are:

- **Vendor ID** - ID of Vendor. Every IO-Link vendor has assigned static Vendor ID to its products.
- **Device ID** - ID of device. Every IO-Link device has assigned unique Device ID to for identification of device.

- **Length of input/output process data** - Sets length of process data. Length can be set up to 32 bytes.

- **Revision ID** - Revision ID distinguish between legacy and standard IO-Link protocol. Legacy protocol has Revision ID 1.0 while standard has 1.1.

- **Device Capabilities** - In this single byte variable there are encoded used M-Sequence frame types for both operate and preoperate modes.

- **MinCycleTime** - MinCycleTime sets minimum cycle time supported by IO-Link device. Real cycle time, assigned during initialization with master, can't be lower than this value.

When new configuration is received, CConfiguration initiates fallback and restart of communication to project changes into IO-Link Master as well.

4.5.3 Process Data

Process data funtionality is handled by 2 classes - CProcessDataIn and CProcessDataOut. While both serve for handling of incoming and outgoing process data, details of their funtionality are little different.

CProcessDataOut are directly set from Application Layer by incoming process data from Master. Once Process Data are changed, CProcessDataOut send Mail to all interested classes about change of process data. This can result in responses from other parts of device (e.g. Digital Outputs are set).

CProcessDataIn on the other hand changes based on information from rest of application (e.g. Digital Inputs) via incoming mail and reports changes directly to Application Layer.

4.5.4 Digital Inputs and Digital Outputs

Digital Input and Digital Output classes are generally quite similar to Process Data classes. Only difference is they control only 4 bits of data each (because there are only 4 digital inputs and outputs). Another difference is that there is no communication with Application Layer at all. Instead all communication runs through mail system.

4.5.5 Events

While handling events, there is need to track which events are active and which are waiting for activation (TMG IO-Link Stack allows sending of only 1 event at a time). For this reason there is need to have at least 2 queues/lists. One for active events and one for event request queue.

Event funtionality is provided by 3 classes. CEvent, which is representing single event, CEventManager, that provides interface for Enqueing events, and CEventQueue, that is storing active events and events waiting for activation.

In CEven class certain informations has to be stored:

- **Event Type** - Types of events are:
  - Error - Serious even on device, that usually block some funtionalities
  - Warning - Warning about some active threat, that could result in error if not cared for
  - Message - Simple information message without any threats.
- **Event Mode** - Modes of events are:
  - Coming - Event is going to appear in industrial ethernet upon activation
  - Going - Event is going to disappear in industrial ethernet upon activation
  - Single - Single information pulse to PLC
- **Instance** - Clarifies Event based on source of occurence. It can be Unknown, PL, DL, AL or APPL.
- **Code** - Unique identification of event.

Another class is CEventManager. When Event is created, it is handled by CEventManager class, that puts this Event into waiting queue. Event flag is then set on communication channel with master. Once Master asks for this event and successfully accepts it, it is processed and based on Event Mode goes into active events queue or tries to delete event in the same queue. In case of Single shot does nothing and only disappear from waiting queue.

4.5.6 File System

Final part of Gateway component is FileSystem. This parts handles every read/write operation to any available FileSystem. Because larger write/read operations can have high time complexity on certain FileSystem technology, there had to be designed a way to support these operations.

File System funtionality is most complex in Gateway. This is caused by attempt, to create design, that could be used universally in wide area of products. Because every product uses different set of FileSystem components, there was need to create design, that could change its layout based on initial configuration and operate all FileSystem components without much effort during assimilation.

In figure 4.9 it is shown how basic design of FileSystem is designed. There are several main components, that are going to be described here:

1. **CFSRequest** - Class, that represent single read/write request to part of available memory. It contains detailed information about destination of request, pointer to buffer to read/write to/from and pointer to Callback class to notify submitter once request is completed.

2. **ICallback** - Contains callback functions to announce successful completion of request or errors during execution.

3. **IMemoryLayout** - Class, that represent virtual layout of available memories in used product. Virtual layout is based on physical layout, with addition of possibility to divide physical memory into more than one virtual memories. This allows easier manipulation and more arranged approach.

4. **IFSRegion** - Memory Layouts are assembled out of several Regions. Each Region represent one kind of memory and offers interface to drivers to execute read/write requests.

5. **Drivers** - Lowest layer of component handling lowest level of communication with File System component, using BSP peripherals usually.

6. **CFileSystem** - CFileSystem class offers main interface to whole FileSystem component. It allows creating read/write requests. Once request is received, class CFSRequest is created and inserted into CFSRequestQueue. Queue system is created to prevent FileSystem from blocking rest of application for too long, since operations can be

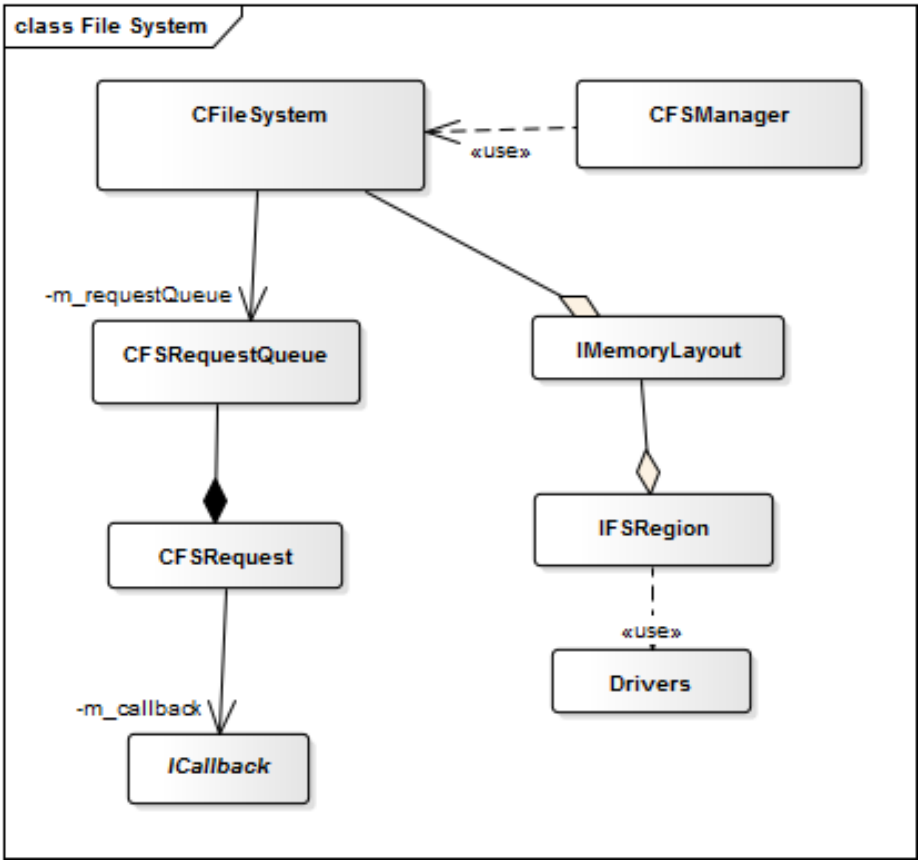


Figure 4.9: File System Design - Basic File System layout. Class IMemoryLayout allows variety FileSystem settings based on particular product.

very time demanding and several operations at once could block the application.

7. **CFSManager** - CFSManager encapsulates some basic operations to FileSystem used by rest of application. For example backup or restore of configuration is encapsulated in 2 functions, which makes usage of FS component fast and clear.

4.5.6.1 Batching of requests

As has been said in previous section, CFileSystem uses queue to handle multiple requests at once. This however doesn't solve problem of large FileSystem request. Therefor batch system was designed with objective to solve large request problem.

In class CFSRequest there is additional variable, that specify size of the batch. For every iteration of FileSystem cycle, there is executed only this small part of request. Once all parts are executed, Callback is called and request is completed.

4.6 RTOS integration

Last part of development was to integrate RTOS. Because IO-Link device is quite small device and didn't need any sophisticated solution for RTOS funtionality, there was selected FreeRTOS solution. FreeRTOS provided everything, that was expected from such OS and already preprepared demos from FreeRTOS developers for ST MCUs made integration quite easy [26].

4.6.1 Optional usage

From beggining the design was made with thought of RTOS being only optional. For this reason the whole application does work even without RTOS. Behavior is slightly different, because time specific constraints created by tasks can't be replaced by single loop, but IO-Link specification is kept

Conclusion

Within the scope of this thesis, a funtional firmware for IO-Link Test device has been developed. This was achieved by integrating already working TMG IO-Link Device Stack and FreeRTOS both written in C, into newly developed core application and peripheral controllers written in C++. Furthermore during development there were successfully collected requirements for both FW, HW and Configuration needs. Working device can be seen in figure 4.10.

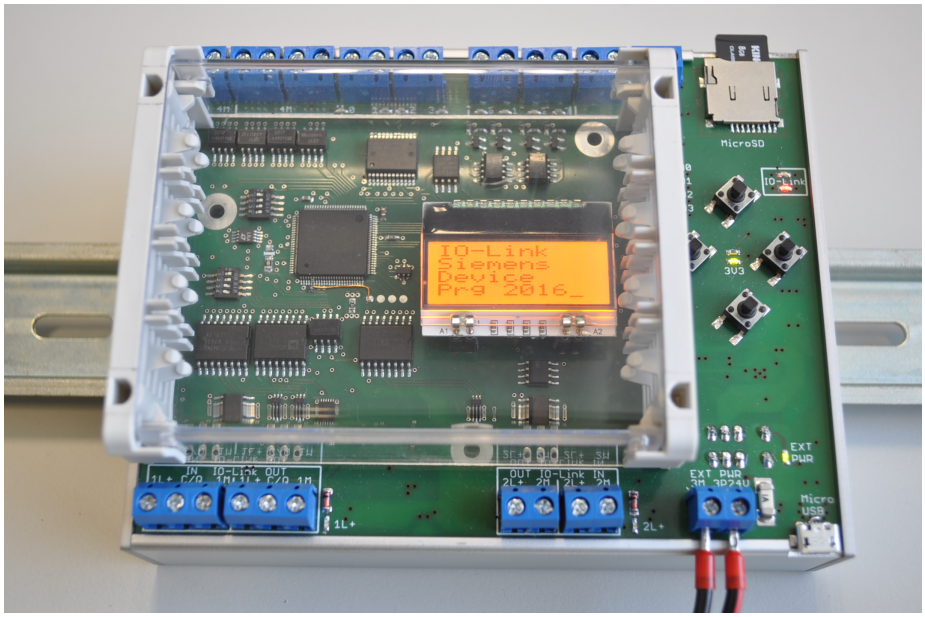


Figure 4.10: Test device board - final prototype design of IO-Link Test device board.

During development there were found multiple HW bugs, as is not uncommon with first prototype, but they all have been fixed quite easily and development didn't had to be stopped with wait time for another release.

Development of Firmware was done with abiding Siemens internal coding guidelines and from funtional point of view fully abides IO-Link specification. Every new funtionality was immeadiatelly tested with self made unit tests and after that with real funtional IO-Link Masters. By this approach any possible bugs were found quite early and didn't cause problems in later stages.

Cooperation with fellow colleagues during development was quite successful and integration of USB part worked immediately without any problems.

At the moment of finalization of this thesis, device is sent for evaluation to Hungary, where local test team experiments with possibilities it offers. Result of this testing unfortunatelly won't be part of this thesis.

There is still lot of development in future, before this test device is going to be fully completed and no further improvements will be neccessary. But at the current stage, the device is already used for several test cases and is not just another work, that ends up in a drawer.

Bibliography

[1] Co se skryva pod vyrazy Industry 4.0 / Prumysl 4.0. [cit. 2016-3-19]. Available from: http://automatizace.hw.cz/mimochodem/co-je-se-skryva-pod-vyrazy-industry-40-prumysl-40.html/

[2] IEC 61131-9:2013 - Programmable controllers - Part 9: Single-drop digital communication interface for small sensors and actuators. [cit. 2013-9-11]. Available from: https://webstore.iec.ch/publication/4558

[3] IO-Link Overview. Available from: http://io-link.com/en/Technology/what_is_IO-Link.php?thisID=76

[4] IO-Link - What it Is and 5 Key Advantages. [cit. 2017-5-03]. Available from: https://www.bannerengineering.com/us/en/company/expert-insights/io-link.html

[5] IO-Link Interface and System specification. [cit. 2013-7-01]. Available from: http://io-link.com/share/Downloads/Spec-Interface/IOL-Interface-Spec_10002_V112_Jul13.pdf

[6] IO-Link popis digitalni komunikace pro senzory. [cit. 2010-2-21]. Available from: http://automatizace.hw.cz/iolink-popis-digitalni-komunikace-pro-senzory

[7] IO-Link System Description. [cit. 2016-2-01]. Available from: http://io-link.com/share/Downloads/At-a-glance/IO-Link_Systembeschreibung_engl_2016.pdf

[8] IO Device Description V1.1 Specification. [cit. 2011-8-01]. Available from: http://io-link.com/share/Downloads/Spec-IODD/IO_Device_Description_V1.1_Specification.zip

[9] IO Device Description V1.1 Guideline. [cit. 2013-7-01]. Available from: http://io-link.com/share/Downloads/Guide-IODD/IO-Device-Desc-Guideline_10022_V11.zip

[10] Schneider, A. *Development of a Human Machine Interface and Communication System with a future IO-Link Master Test Device*. Master's thesis, CTU FEL, 2016. Available from: https://support.dce.felk.cvut.cz/mediawiki/images/b/b2/Dp_2016_schneider_alexander.pdf

[11] STM32 L1 series of ultra-low-power MCUs. Available from: http://www.st.com/en/microcontrollers/stm32l1-series.html?querycriteria=productId=SS1295

[12] STM32 F1 series of mainstream MCUs. Available from: http://www.st.com/en/microcontrollers/stm32f1-series.html?querycriteria=productId=SS1031

[13] STM32F4 series of high-performance MCUs. Available from: http://www.st.com/en/microcontrollers/stm32f4-series.html?querycriteria=productId=SS1577

[14] STM32F407VG, Datasheet - production data. Available from: http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf

[15] IO-Link Device Transceiver - MAX14826. Available from: https://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/MAX14826.html

[16] IO-Link communication transceiver device IC. Available from: http://www.st.com/en/power-management/l6362a.html

[17] SN65HVD101 IO-LINK PHY for Device Nodes. Available from: http://www.ti.com/product/SN65HVD101

[18] Eclipse Mars. Available from: https://eclipse.org/mars/

[19] IAR. Available from: https://www.iar.com/iar-embedded-workbench/

[20] Enterprise Architect. Available from: http://www.sparxsystems.com/products/ea/

[21] Rational ClearCase. Available from: http://www-03.ibm.com/software/products/en/clearcase

[22] Strategy Design Pattern. Available from: https://sourcemaking.com/design_patterns/strategy

[23] Singleton Design Pattern. Available from: https://sourcemaking.com/design_patterns/singleton

[24] IO-Link Protocol Software for Device - TMG. Available from: https://www.tmgte.de/en/IO-Link-component/io-link-protokoll-stack-device.html

[25] Publish/Subscribe pattern. Available from: https://msdn.microsoft.com/en-us/library/ff649664.aspx

[26] FreeRTOS. Available from: http://www.freertos.org

Appendix A

Acronyms

HW Hardware

FW Firmware

SDCI Single-Drop Digital Communication Interface

UART Universal Asynchronous receiver/transmiter

SPI Serial Peripheral Interface

SIO Standard Input and Output

IEC International Electrotechnical Commission

PD Process Data

OD On-Request Data

PL Physical Layer

DL Data Link Layer

AL Application Layer

IODD IO-Link Device Description

XML eXtensible Markup Language

EMC Electro Magnetic Compliance

DC Direct Current

USB Universal Serial Bus

SD Secure Digital

GUI Graphical user interface

LED Light-Emitting Diode

RAM Random Access Memory

JTAG Joint Test Action Group

ARM Advanced RISC Machine

MCU Microcontroller Unit

BSP Board Support Package

RTOS Real-Time Operating System

DI Digital Input

DO/DQ Digital Output

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format