MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ansorge Michal** Personal ID number: **420123**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Open Informatics**

Branch of study: **Computer Engineering**

II. Master's thesis details

Master's thesis title in English:

IO-Link Device test platforms as a support for DevOps

Master's thesis title in Czech:

Podpora DevOps vývoje IO-Link Device testovací platformou

Guidelines:

1. Examine possibilities of DevOps development methods for testing of IO-Link master products.
2. Pay attention especially to process automation using IO-Link Device test platform.
3. Pay attention also to integration test requirements for IO-Link Master products testing.
4. Design extension of IO-Link Device test platform firmware to support DevOps development.
5. Design extension of software configuration application for IO-Link Device test platform.
6. Implement relevant proposed changes to improve current development processes (in the context of savings in time and financial resources).
7. Prepare demo to demonstrate usage of proposed changes.

Bibliography / sources:

[1] IO-Link Interface and System specification, V1.1.2, available from http://io-link.com/share/Downloads/Spec-Interface/IOL-Interface-Spec_10002_V112_Jul13.pdf
[2] IO-Link System Overview, available on http://io-link.com/share/Downloads/At-a-glance/IO-Link_Systembeschreibung_engl_2016.pdf
[3] IO-Link Test Specification, V1.1.2, available from http://io-link.com/share/Downloads/Testspec/IOL-Test-Spec_10032_V112_Jul14.pdf
[4] IO Device Description V1.1 specification, available from http://io-link.com/share/Downloads/Spec-IODD/IO_Device_Description_V1.1_Specification.zip
[5] IO Device Description V1.1 Guideline, available from http://io-link.com/share/Downloads/Guide-IODD/IO-Device-Desc-Guideline_10022_V11.zip
[6] IO-Link Device for testing of IO-Link Masters, Diploma Thesis, Ondřej Volf, 2017

Name and workplace of master's thesis supervisor:

Ing. Miloš Fenyk, Siemens, s.r.o., Praha

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **15.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

[redacted]

Ing. Miloš Fenyk
Supervisor's signature

[redacted]

prof. Ing. Michael Šebek, DrSc.
Head of department's signature

[redacted]

prof. Ing. Pavel Ripka, CSc.
Dean's signature

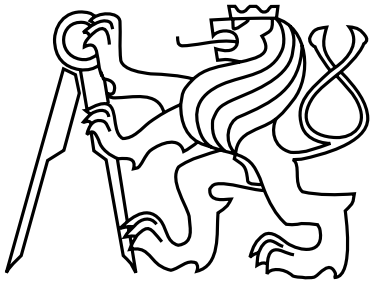
III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

14. 12. 2018
Date of assignment receipt

[redacted]
Student's signature

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering



Master's Thesis

IO-Link Device test platforms as a support for DevOps

Bc. Michal Ansorge

Supervisor: Ing. Miloš Fenyk

Study Programme: Open Informatics, Master programme

Field of Study: Computer Engineering

January 7, 2019

Aknowledgements

I would like to especially thank my supervisor of the master's thesis, Ing. Miloš Fenyk, for patience and inspiring suggestions to guide my work that helped to complete it in the final form. I would also like to thank my other colleagues from the development team for supporting me during my work.

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague on January 8, 2019 .....

Abstract

IO-Link technology was developed as a standardized communication protocol for modern and featured sensors and actuators. This thesis describes modern development methodologies which can be used in the IO-Link Masters development and proposes possible particular adoption of DevOps culture in the development. Further, design and implementation of a test partner IO-Link Device for the Masters with test automation support, which is needed for the adoption of the modern methodologies in the IO-Link Masters development, are described.

Keywords:
IO-Link, DevOps, IO-Link Master, IO-Link Device, Test automation, IO-Link Test Device

Abstrakt

Technologie IO-Link byla vyvynuta jako standardizovaný komunikační protokol pro moderní a chytré senzory a aktuátory. Tato práce popisuje moderní metody vývoje, které mohou být použity při vývoji IO-Link Master zařízení a konkrétně popisuje možnou integraci přístupu DevOps při vývoji. Dále práce popisuje návrh a implementaci IO-Link Device zařízení, které slouží jako podpora při automatizaci testů IO-Link Master zařízení během jejich vývoje. Toto testovací zařízení je potřebné při vývoji IO-Link Master produktů za využití moderních vývojových metod.

Klíčová slova:
IO-Link, DevOps, IO-Link Master, IO-Link Device, Automatizace testů, IO-Link Test Device

Contents

List of Figures

List of Tables

Nomenclature

AL Application Layer

CD Continuous Deployment/Delivery

CI Continuous Integration

Device General IO-Link Device

DevOps Development and Operations

DI Digital Input

DL Data-Link Layer

DPP Direct Parameter Page

DQ Digital Output

DS Data Storage

FW Firmware

HMI Human Machine Interface

HW Hardware

IDE Integrated Development Environment

IOLTD IO-Link test Device / IO-Link Device test platform

ISDU Index Service Data Unit

LSB Least Significant Bit

Master General IO-Link Master

MCU Microcontroller Unit

MSB Most Significant Bit

OPCUA Unified Architecture from OPC foundation

PD Process Data

PHY Physical Layer on chip

PL Physical Layer

PLC Programmable Logical Controller

SM System Management

SPI Serial Peripheral Interface

SW Software

TFS Team Foundation Server

$UART$ Universal Asynchronous Receiver and Transmitter

VCS Version Control System

Chapter 1

Introduction

Background

As technology goes forward and industry automation is evolving, also elements, like sensors and actuators, are becoming smarter and more complex. Such elements no longer use a simple analog or digital signals to be driven or to send the measured values. Capabilities, like sending diagnostic information or configuration data, require complex communication protocol. IO-Link was developed as a standardized (IEC 61131-9) technology to communicate with modern and featured sensors and actuators based on Master/slave model [19]. Siemens, as one of the key companies on the industrial automation market, also develops IO-Link Master products to integrate the IO-Link technology with PROFIBUS/PROFINET systems.

Modern product development is generally getting faster. Agile development practices are widely integrated in the development across all IT fields to keep up with the modern practices, quickly evolving market, and the competition. Even though not all such practices are suitable for all the fields, some of them can be adopted in the IO-Link Master development, to make the product development more efficient while preserving or even improving its reliability. One of the most important practices in modern agile methodologies is test automation. Since the IO-Link technology uses the Master/slave model, developed master need the slave connected to demonstrate the validity of their implementation. Such slave should be able to simulate any IO-Link environment possible to provide proper capabilities as a test partner for IO-Link masters in the test automation. Otherwise, manually exchanging a significant number of various devices to provide good enough coverage during the testing would be needed.

Thesis structure

Research chapter of the thesis describes the IO-Link technology, its usage, and its capabilities. Description of IO-Link topology, communication model, and communication features are the main parts of this section. The chapter further provides an overview of DevOps culture, goals, and practices, which can be adopted in the IO-Link Master development. End of the chapter introduces IO-Link Device test platform (IO-Link test device - IOLTD) in its current state.

Third chapter describes, how particular DevOps practices can be achieved using the IO-Link Device test platform, how the IOLTD can use IO-Link technology features to support automated Master testing and specifies the required features and capabilities of the IOLTD. Last section of the chapter describes possible DevOps culture adoption in the IO-Link Master development, which particular tools can be used, and which particular practices can be adopted during the whole development cycle.

Chapters four and five describe design and implementation of the IOLTD FW extension to support DevOps and implement all the requirements. Both chapters describe utilization of already existing parts of the FW and design and implementation of the new parts. Last section of the implementation chapter also shows proof of the concept of the FW extension.

Chapter 2

Research

2.1 IO-Link

Following section contains information about the IO-Link technology based on the IO-Link Interface and System Specification document [20], if not stated otherwise for additional information.

2.1.1 Introduction

IO-Link is relatively new technology (widely integrated since 2010) and was developed to keep up with the progression of the industrial automation, as the sensors and actuators are becoming smarter and offer more features. It is a standardized protocol using point-to-point serial communication line. Main advantages of IO-Link technology are easy interconnection with other fieldbus technologies (e.g. profinet, EtherCAT), fast and automatic transfer of measured values, settings, or diagnostic information.

Sensors and actuators (IO-Link Devices) are connected to the control systems through gateways (IO-Link Masters). These gateways also serve as remote I/O devices in this control system. One IO-Link Master can handle multiple IO-Link Devices, from which it obtains measured values, and to which it sends control information. This means, that sensors and actuators are remotely manageable and capable of sending process data or information about their state to their Master.

2.1.2 Topology

IO-Link technology uses star topology with the Master in the middle. Master establishes point-to-point communication with the connected Devices. Master always has one physical

port for each Device, and therefore there is no need of addressing the Devices. The main purpose of the IO-Link technology is to standardize communication protocol for the sensors and actuators, so it can be designed and developed regardless of the Master Vendor and the superior fieldbus system. See figure 2.1 below.[14]

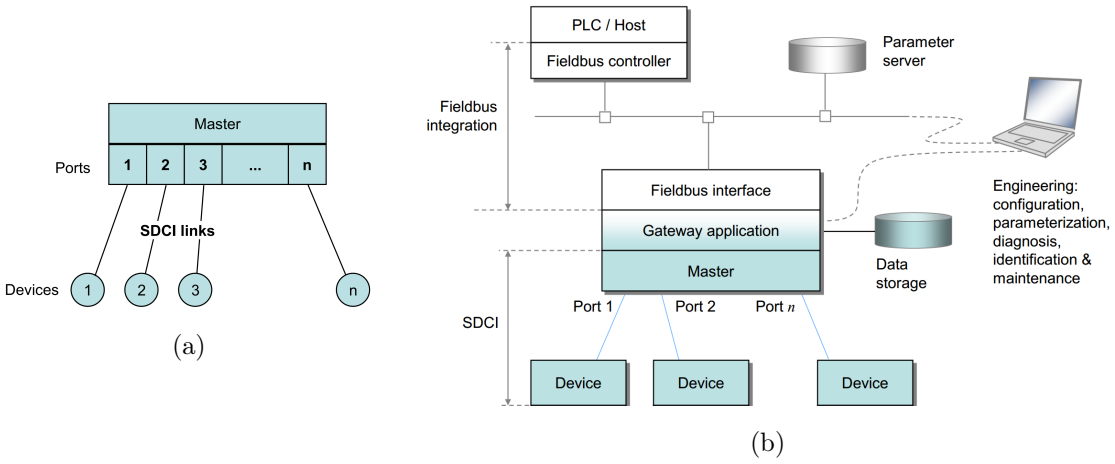


Figure 2.1: Star topology of IO-Link alone (a) and IO-Link together with a superior fieldbus (b).

2.1.3 Communication features

Every Device has to support sending of the input Process data/Event and receiving of the output Process data to/from the Master. Each Device has a specific parameterization and configuration needs for its particular purpose. For such parameters and configuration data, the IO-Link specification defines parameter space, which is accessed by Indexes and Subindexes. This space can have up to 65535 Indexes, where each Index addresses up to 232 octets. Each Index can also be divided into Subindexes. The Subindex can address any subsequence of octets in the Index. One Index can have up to 255 Subindexes and each Subindex can have arbitrary length, as long as this length will not exceed the end of its superior Index. The only exception is Subindex 0, which always addresses all data of its superior Index.

First two Indexes (0 and 1) are reserved for Direct Parameter pages 1 and 2 (DPP1,2), where each of them has 16 octets. The DPP1 serves mainly for the commands from Master and retrieval of Device specific operational and identification information. The DPP2 does not have any pre-defined type of information and its purpose is purely device-specific. See figure 2.2 for illustration of Index and Subindex principle.

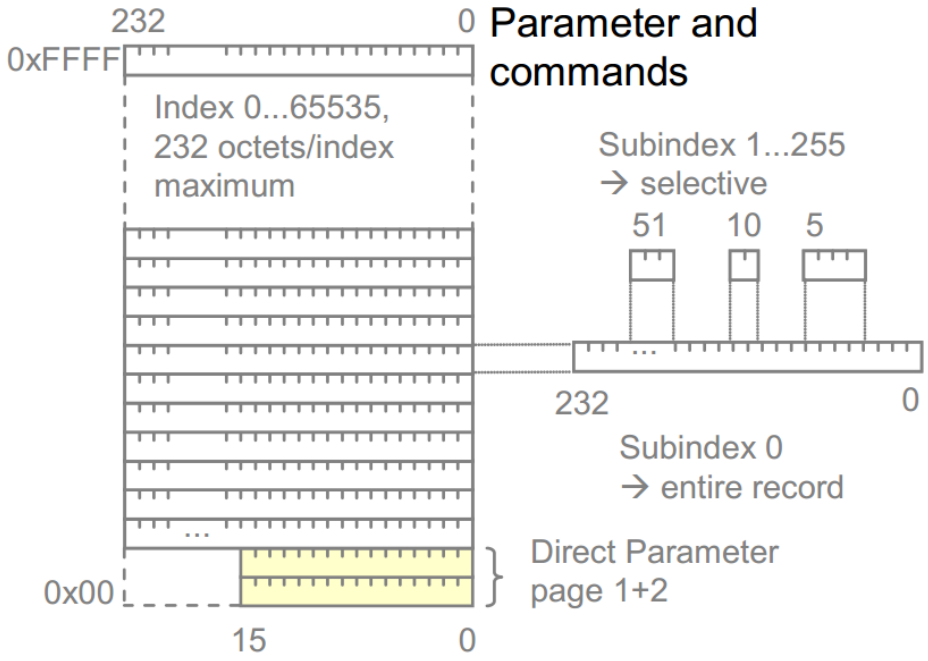


Figure 2.2: Index and Subindex space principle illustration

2.1.4 Communication model

The IO-Link communication model consists mainly of physical layer (PL), data link layer (DL) and the application layer (AL). Above the application layer is the application itself and besides all layers operates the system management (SM). The IO-Link specification also defines features above the AL. For the Master, these are Process Data Exchange (PDE), On-request Data Exchange (ODE), Configuration Management (CM), Data Storage (DS) and Diagnosis Unit (DU). For the Device, these are Process Data Exchange (PDE), Parameter Management (PM), Data Storage (DS), and Event Dispatcher (ED). See table 2.1 and figure 2.3 for description and illustration.

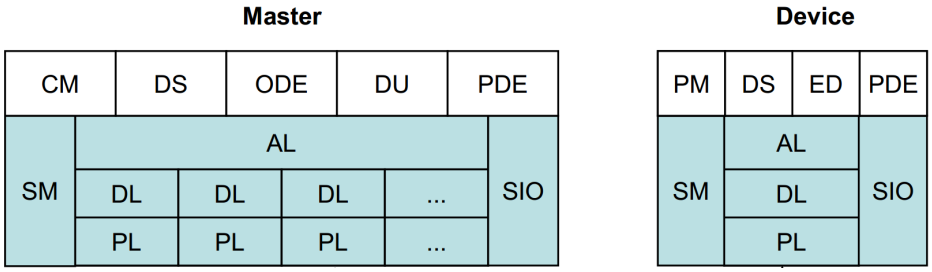


Figure 2.3: IO-Link communication model layers illustration

Model Layer	Description
Application Layer (AL)	Defines and realizes services for access to the Process data, On-request data and Events.
Data link layer (DL)	Defines and realizes data transfer between the Master and the Device, handles communication errors and serves as an interface between AL and PL
Physical layer (PL)	Defines physical properties of the communication and takes care of the access on a transmission medium.
System management (SM)	Defines and realizes start-up and coordination of the communication and the ports on both Master and Device.

Table 2.1: Communication model description

2.1.5 Physical Layer

IO-Link uses three wire connection and the wires have the following purposes. For illustration, see figure 2.4.

- L+ as 24V power supply
- C/Q has either communication line (C) or switching signal (Q) state
- L- as ground line

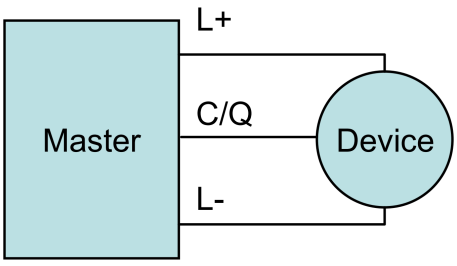


Figure 2.4: IO-Link wiring illustration

Besides the two main states, the C/Q line has one more state, which is *inactive*. In this state, the line is in the high impedance. The two main states (C and Q) of C/Q line are meant to be alternating. Normally the line is in the mode of switching signal (DI/DO) and

when the communication mode of the line is needed, Master sends a wake-up signal pattern, which is then recognized by the Device and the line goes to the communication mode.

Transmission channel characteristics

The C/Q line in the communication mode is based on the UART interface and has one of the following settings.

- COM1 - 4.8 kb/s
- COM2 - 38.4 kb/s
- COM3 - 230.4 kb/s

Minimum threshold voltage for recognition as logical high ranges from 10.5V to 13V and maximum threshold voltage for recognition as logical low ranges from 8V to 11.5V. Modulation method NRZ (Non-Return to Zero) is used and the Logical level is evaluated as a voltage difference between lines C/Q and L-, where the difference of 0V means logical "1" and the difference of +24V means logical "0".

Since the IO-Link communication is based on the UART interface, UART frame with one parity bit and one stop bit are used for the transmission. See the structure of UART frame in figure 2.5.

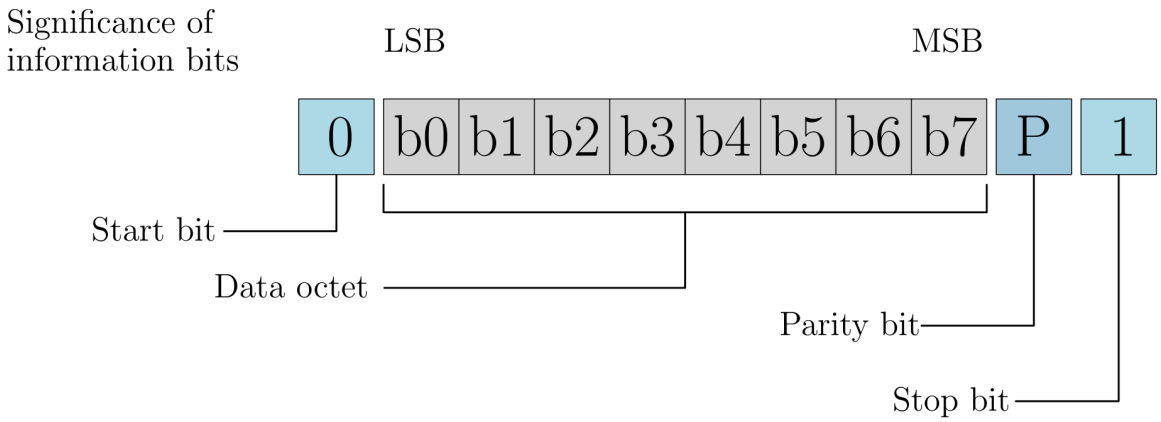


Figure 2.5: Used UART frame structure

Connectors and cable

IO-Link specification defines four possible types of the connectors divided into two classes. Class A connectors with possible types M5, M8, and M12 and with a maximum of four

pins. Class B connectors are of type M12 with 5 pins. In both classes, the M12 pin always has 5 pins on the Master side (female connector). Contribution of the fourth line in the class A connector is a possibility of having separate communication line and switching signal (DI/DO). In case of class B, 2 extra pins are used for an extra power supply with another pair of L+ and its reference L-. Thus the class B connectors are used in cases of power devices. The Mandatory fifth pin on M12 class A female connector is not connected. See figure 2.6.

Cable for IO-Link connection has following limitations:

- Max. length - 20 m
- Max. overall loop resistance - 6 Ω
- Max. effective capacitance 3 nF

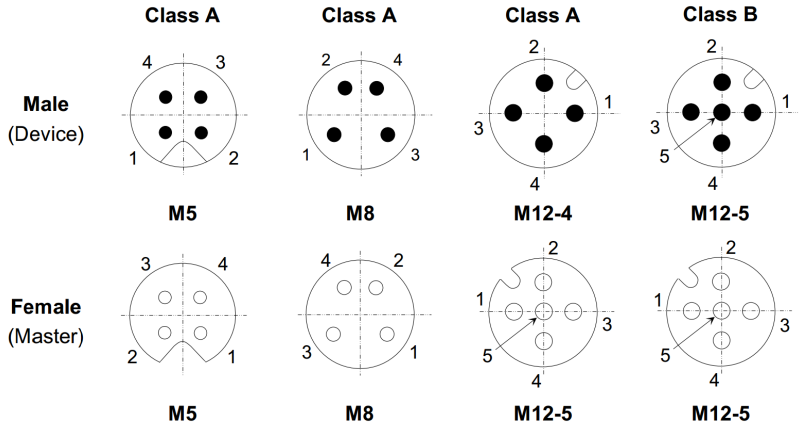


Figure 2.6: Pin layout in connectors

2.1.6 Data Link Layer

Before sending the data of any specific application over the physical layer, the data link layer is needed. The Data link layer serves as an interface between the signals transmissions on the physical layer and the handling application data. This layer offers a set of DL-services for the exchange of different types of the data from the application layer and another set of DL-services to the system management (SM) so SM is able to obtain Device identification parameters. For the communication with the physical layer, the data link layer uses a set of PL-services for the UART frame exchanges. The DL also handles error in the

communications. Both parity error from PL and checksum errors recognized in the DL are handled.

The Data link layer decodes and classifies the requested services and processes those designated to it. Other requests are delegated to the competent layer and can be attached with additional information (for example checksum). It also generates error/status messages to inform the connected device or any other layer in the same device. From the principle of the master-slave model, the data link layers of the Masters and the Devices differ in several things. Master, for example, initiates the start of the communication and its settings, thus the Master's DL is more complex.

Communication data are divided into the following data types:

- Process data (PD) - High priority information transferred automatically once for every communication cycle. Process data typically carry the measured values from Device to Master or the control data from Master to Device.
- On-request data (OD) - Acyclic request sent from Master to Device. Typically r/w requests.
  - Diagnosis data - Special case of the OD requested by the Master to gather diagnosis information about the Device.
  - Events - Special case of OD, where the Device adds information about having new event into the next data transfer and the Master then requests event data.

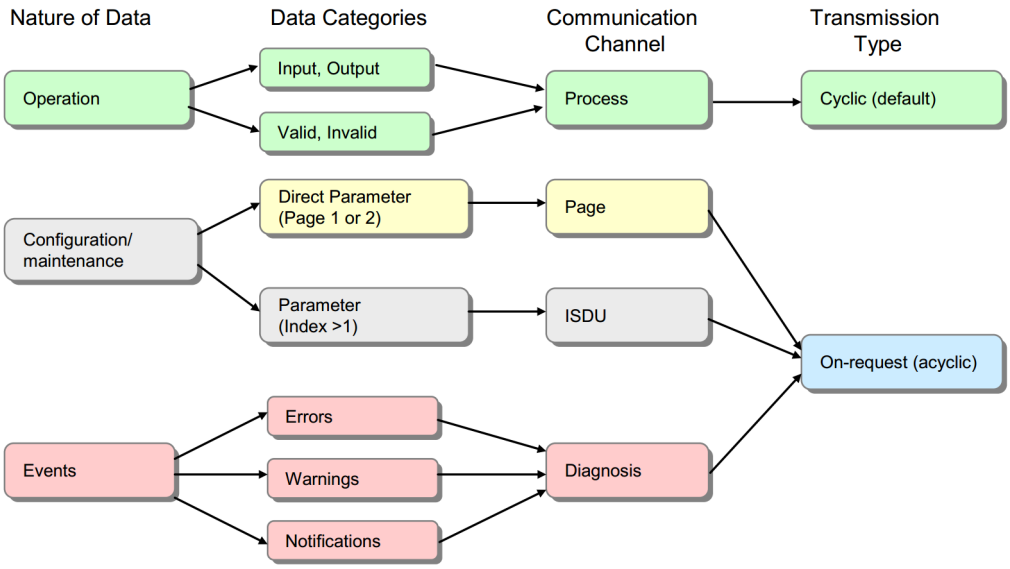


Figure 2.7: Relation between data types and channels used for their transmission

The DL also uses separate communication channel for each of these data types. See figure 2.7.

- Cyclic transmission type
  - Process channel serves for the input/output process data, which can be associated with qualifiers, such as valid/invalid.
- On-request (acyclic) transmission type
  - Page channel serves for a configuration data and Device specific operational information (Direct Parameter pages 1 and 2).
  - ISDU channel serves for other maintenance parameters (Indexes > 1)
  - Diagnostic channel serves for the transmission of the events, which are reported with one of three severity levels (error, warning, notification).

The DL associates individual UART frames into a message sequence (M-sequence) telegrams. These telegrams differ depending on the type of carried data and data length. Each M-sequence telegram contains its type in its header. Basic structure of such telegram can be seen in figure 2.8.

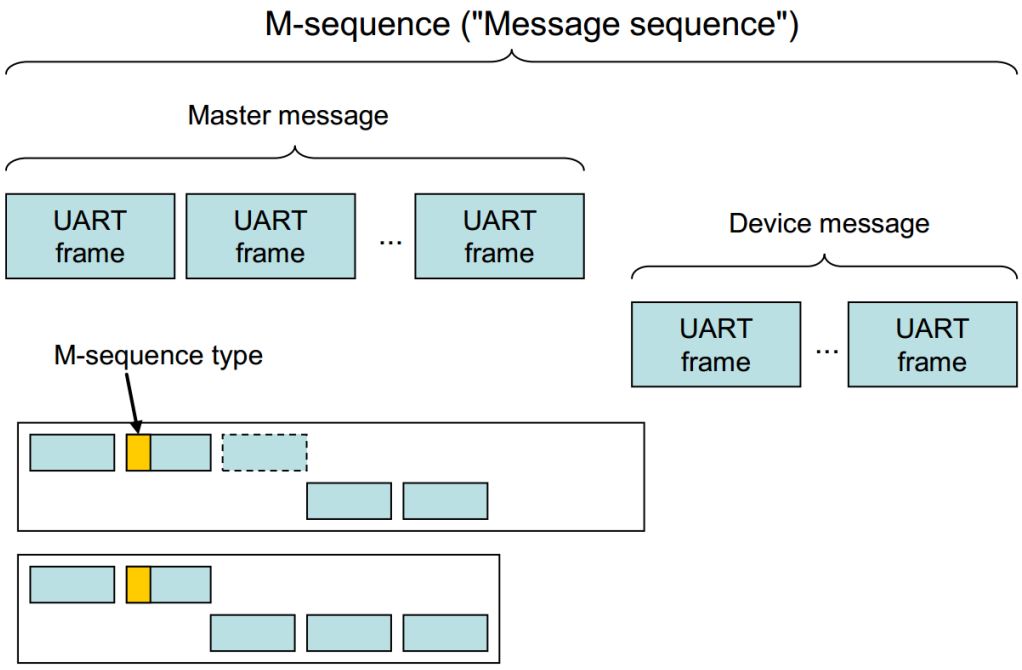


Figure 2.8: M-sequence telegram form

Telegram from the Master always starts with a M-sequence control octet (MC) and is followed by a CHECK/TYPE octet (CKT). Optionally, the CKT can be further followed by the Process data (PD) and/or the On-request data (OD).

Telegram from the Device starts optionally with the Process data (PD) and/or the On-request data (OD) followed by a CHECK/STAT octet (CKS).

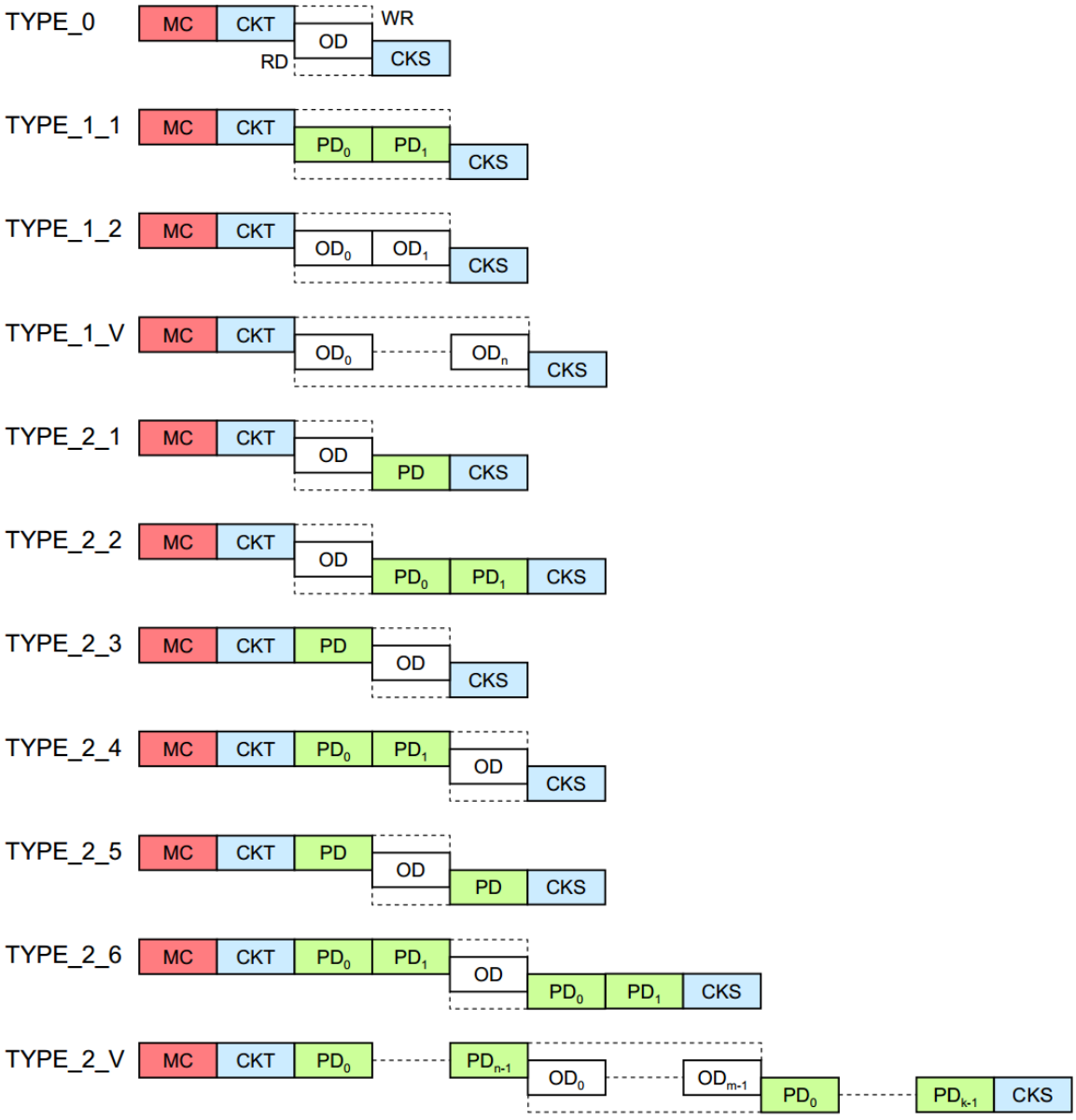


Figure 2.9: Telegram types

Eleven types of telegram exist. Nine of these types have fixed length and two have variable length (TYPE_1_V and TYPE_2_V). Various types of telegram exist, so one of them can meet particular Device requirements as effectively as possible (for example requirements for the amount of the PD). See figure 2.9 for illustration of particular telegram types.

Particular octets have following meaning:

- MC - carries information about operation type (r/w) and type of communication (PD, OD, Diagnosis, or event).
- CKT - specifies type of the telegram and carries checksum of the sent data
- OD/PD - data
- CKS - The active events information and the availability of PD

2.1.7 Application Layer

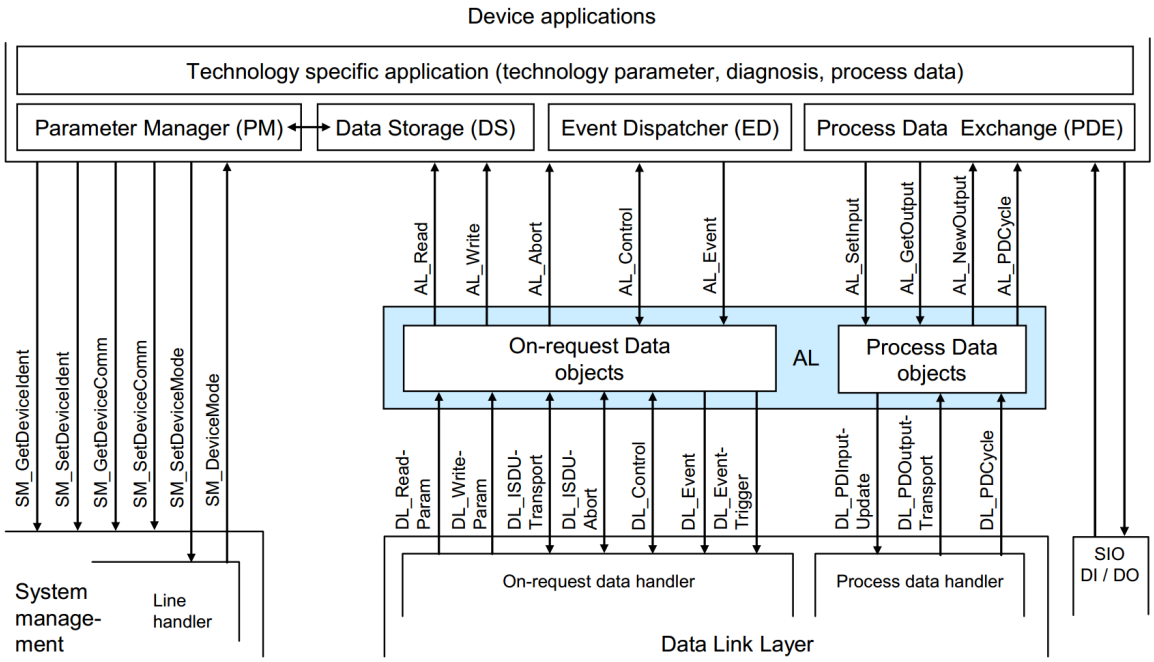


Figure 2.10: Telegram types

The Application layer (AL) serves as an interface between the data link layer and the application itself. After the DL decodes frames and telegrams in the application data, DL triggers the AL with information, which service the AL shall process and what data it shall use. The

layer application implements a set of services for its neighboring layers. These services are handling requests for operations on On-request data, Process data, Events and Control data. Figure 2.10 shows the structure and the services of the application layer in the Device.

2.1.8 System management

The System management is responsible for coordinating the start-up of the communication and ports between the connected Master and Device. In this layer, there are the most significant differences between implementations in the Masters and the Devices.

The Master system management is able to set the ports in all possible configurations and operation modes. When the Device is connected, the Master establishes required communication protocol revision, checks Device compatibility and chooses adequate telegram types, communication cycle time and port configuration.

The Device system management controls the C/Q line through all phases of the initialization. Line is in the SIO mode (switching signal DI/DO) by default. SM recognizes pulse for communication start-up, switches the line to the communication mode and then also switches the line back to the SIO mode. SM also provides the Device identification parameters through the application interface.

2.2 DevOps

2.2.1 Overview

DevOps is an approach, which combines several concepts of the agile development and integration techniques to achieve interconnection of development and operations. Such interconnection, illustrated in figure 2.11, is meant to result in the creation of a single effective unit, from which both development and operations can benefit. Name of this methodology originates from the connection of words Development and Operations. DevOps is not precisely defined methodology, it is rather a modern culture and view, how today projects can be managed. It mainly builds on already existing and widely used agile techniques, but it also strongly focuses on cooperation with the operations environment and integrating its feedback to the development. This process can be also seen in the figure 2.11 as a continuous cycle of development and using operations feedback as an input to the planning.

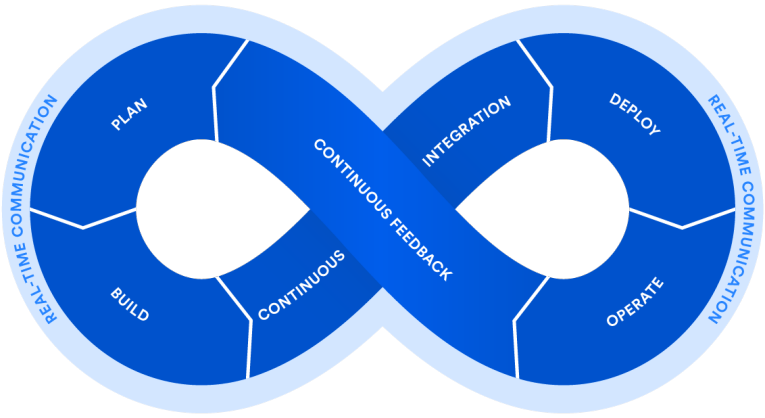


Figure 2.11: DevOps loop illustration [13]

2.2.2 DevOps goals

Following section describes essential DevOps goals based on the article [13].

Fast development

Ability of fast development and releasing is an essential attribute when the development aims to instant adapting to the customer needs and the situation changes on the market. Continuous integration and continuous delivery are typical examples of practices, which make the fast development possible.

Reliability

Stability and reliability of the product developed using DevOps are also qualities, which are meant to be ensured by the continuous integration and delivery. To achieve the best of these qualities, automated testing is a must. Such testing should cover as much of implementation as possible, so that potential problems are found immediately, which would save both money and time.

Scalability

Design of the product should count on the scalability from the very beginning. This is a major quality for fulfilling the requirements and quickly changing needs of the customer or the market.

Generally, two ways to achieve the scalability exist. First way is called vertical scaling and it means adding more or more powerful hardware capable of handling higher load of the system. Second and the one meant in the DevOps is called horizontal scaling. In this case, universal and expandable architecture is designed from the beginning of project. Expandability will ensure the ability to easily add new features resulting from new requirements and the universal design is meant to be (as much as possible) independent from specific conditions, which are at the moment but might change in the future. For example, product handling data should perform well independently on the data amount when there is considerable possibility of its future changes.

Fast delivery

Other DevOps goal is rapid and frequent delivery of the new products or versions. Fast delivery increases ability of the operations to find out their next needs and the development to improve their releases as often as possible. More frequent and smaller releases also help to reveal sources of potential troubles sooner, and thus to solve them in less time and costs.

Early error detection

Fast, small and frequent releases also enable the early error detection and mean less effort to find and fix the problem. When the bug is found during the automated testing or reported by the operations, only a small difference between last operational and faulty version has to be investigated to find a problem.

2.2.3 Essential practices in DevOps

Automation

Process automation plays a significant role when adopting DevOps techniques. Because the development part of the DevOps culture significantly builds on the fast and agile practices, there is also need of an infrastructure, which is able to support such development, so time of the developers does not need to be used on tasks, which can be automated. Practice called "infrastructure as a code" can be used for infrastructure automation. Its concept is creating of the scrips, which are automatically executed and capable of proceeding every needed step from the very beginning when developers tag new release in the VCS to the very end, where also the deployment of the release to the customer is done automatically.

All procedures in between are automated, for example, build of the project, creating virtual testing environment or a static code analysis.

Another significant must and also a challenge is the testing automation. Even with fast agile approach and automated product delivery, product reliability has to stay as one of the most desired qualities. For this reason, effective and significant test coverage is a must. Automated testing is often able to process considerably more tests, but their design and implementation and result evaluation have to be very robust.

Continuous integration

Continuous integration (CI) is a practice, which aims to early detection of bugs. CI builds on frequent publishing new changes in the version control system. With the CI, the developers publish every new fix or a feature into the shared repository and merge it into main branch, where automated analyses and testing can detect eventual bug and report it. This approach can save significant amount of time and costs, which would be needed for the investigation of the bug and its origin when only big changes (pack of fixes and features) would be merged and tested as a whole.[11]

Continuous delivery

In addition to the CI, continuous delivery (CD) means, that new product versions can be released very often. In the most strict case, every change, which passes all tests can trigger new automated release to the customer (this case is also called continuous deployment). In more free interpretation, this allows to release the product simply as often as required.

Continuous Feedback

Frequent gathering of the feedback from the operations is one of the key practices, which is promoting the DevOps culture above the traditional agile approaches to the development. In the planning phase of each cycle of the DevOps loop (shown in figure 2.11), feedback from the operations is taken into account. Because the operations are directly influenced with evolving market, they can swiftly provide new requirements for the product. Using the continuous delivery with frequent releases, this ensures the developed software is up-to-date and also the satisfaction of the customer. Also, the misbehavior of the product or misunderstanding the feature request at the beginning can be quickly corrected using the continuous feedback.[12]

2.3 IO-Link Device test platform

2.3.1 Overview

In the agile development techniques in the development of embedded systems for the industrial automation, testers of such systems must be able to perform significant amount of tests, so automation of these tests is critical for maximizing benefits of the agile development. For the test automation, there is need of Device, which would support all possible configurations, which are allowed by the IO-Link specification, in other words, all possible configuration, which Master has to expect and has to be implemented to handle.

Such Device also has to be stable, so it will not fail in any circumstances, that could occur during long run tests, which should be run without the need of any attention from the tester. For example, this Device has to pass same tests like EMC and temperature tests [18], as any other Devices or Masters. But also FW of such Device has to be stable and robust enough, so potential error in Master (and thus undefined behavior of communication) would not deactivate this Device and other tests could proceed. The IOLTD is being developed for these purposes.

2.3.2 Current state

Until this date, the IOLTD has come through these three development phases:

- HW component research end design.
- Implementation of core FW, including communication with the IO-Link stack and possibility to run under the FreeRTOS system.
- Implementation of FW module with capability of communication parameter re-configuration over the USB connection and Windows C application as a human interface of this module.

Hardware

Since this Device was not meant for a bulk production, more costly MCU STM32F407VGT was chosen for its performance capabilities and for developers know-how with ST MCUs. Also, a physical layer (PHY) was chosen from the same vendor, particularly ST L6362A. This PHY is fully protected against mistreatments (reverse polarity, over-temperature, under-voltage etc.), supports Wake-up pulse detection and comes in 3x3mm package. Main hardware sources on the IOLTD are:

- STM32F405VGT MCU with up-to 168 MHz core frequency
- 1 MB of flash memory
- Both JTAG and SWD debug interfaces
- Up to 100 I/O pins with interrupt capability
- 4x USART
- 3x SPI
- USB 2.0 with on-chip PHY

IO-Link stack

The IOLTD uses IO-Link stack developed by Technologie Management Gruppe (TMG). This stack was available to use in this Device and already proved its qualities before. Stack fully supports IO-Link specification in both versions 1.1 and 1.0 and all telegram types. See figure 2.12 for basic illustration of Device FW using TMG stack. The Stack is written in ANSI-C language.[10]

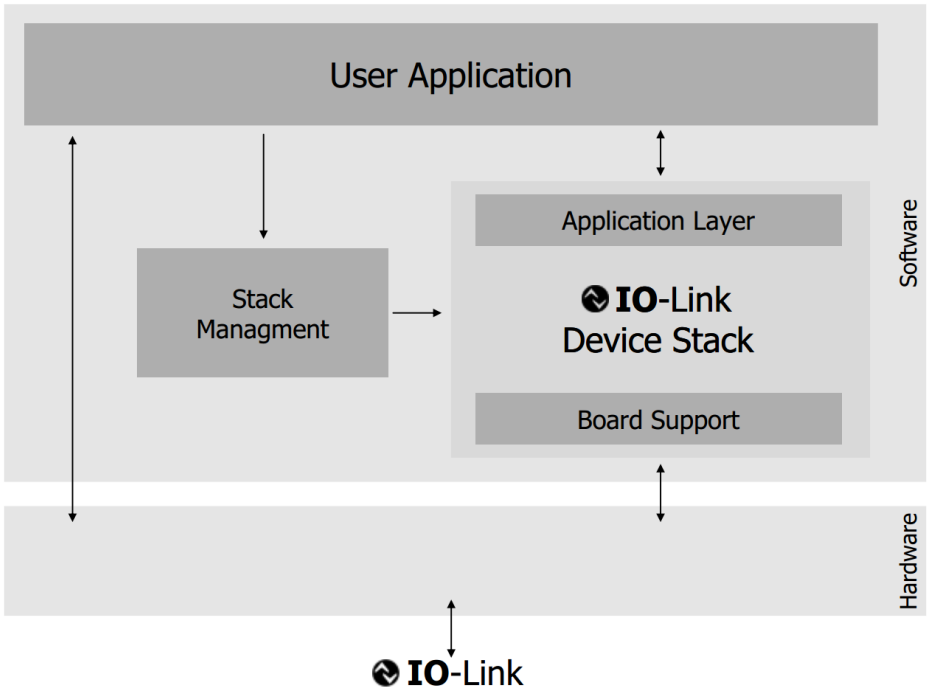


Figure 2.12: Illustration of Device FW using TMG stack

OS

Device implements FreeRTOS, for its simplicity, which meets with low OS requirements of this Device. Usage of the OS by Device FW is optional. Current FW was developed to be standalone from the beginning, so hardcoded switch exists to pick between usage with or without FreeRTOS.[9]

Core application

Core component responsible for the majority of current FW functions is called Gateway application. Gateway application implements features listed in table 2.2 below and was written in language extended embedded C++.

Feature	Description
Mail system	Serves for the communication between all classes inside of the Gateway application. Using this mail system guarantees granularity of processed tasks and thus prevents chain events from happening, which could lead to breach of real-time nature of the Device.
Configuration	Maintains and manages current configuration of the Device and in case of configuration changes causes re-initialization of the communication with the new configuration
Process Data	Maintains and manages input and output Process data, their propagation to and from physical input and output pins (with DI/DQ feature), their validity and synchronization with stack
DI/DQ	Provides an interface between the Gateway application and a digital I/O pins (4x DI/4x DQ)
Events	Maintains and manages the events, their state in the stack and thus their propagation to the Master.
File System	Handles storing needed Non-volatile data to the onboard SPI flash, for example, the active configuration, so it can be loaded even after power cycle.
Gateway To Stack	Provides interface between the Gateway application and the stack (and its available AL services).

Table 2.2: Current state features [15]

USB configuration interface

In the current state, the IOLTD is partly configurable through USB interface [2]. USB component in the FW handles communication with PC and propagates the requests to the Gateway application, where it can change limited amount of configuration or parameter data, or read information about state of the Device. Also, PC GUI application written in C# programming language exists. This application serves as the HMI of the USB component in the Device.

Advantage of this interface is a possibility of reading status of the Device and changing communication parameters, even when the connection to the Master fails for some reason. On the other hand, this feature does not support full configure-ability or ability to set the Device into a completely arbitrary state, which is allowed by the specification. But maybe even more important is the fact, that for changing configuration of the Device, tester has to be present with his PC and manually proceed desired changes. In such case, tests have to be crumbled on smaller parts. Tester is not able to change configuration and parametrization of the IOLTD directly from the script, which is performing the testing and also his frequent presence is needed.

Chapter 3

IOLTD as support for DevOps in IO-Link Masters development

3.1 Techniques allowing DevOps

Main benefit of the IOLTD for the DevOps approach is its ability to support the continuous integration and the automated testing. IO-Link technology allows manufacturers to design their devices in many different ways and options to fit the needs of produced device as much as possible.

Since the IO-Link Master has to be able to support all of these ways and options, they also need to be properly tested in all ranges allowed by the IO-Link specification. Using many different Devices with different designs to meet all requirements of proper testing is not only expensive but also unpractical. For example need of manual replacement of different Devices during testing is very time consuming and also significantly limits abilities of automated testing.

Idea of using specialized IO-Link Device designed specifically as the test device brings new possibilities. Device able of being re-configured automatically during testing allows automation of much more complex tests. Such re-configuration is also possible from within of the same program, which is executing the tests. Such device then could simulate every possible IO-Link Device with any configuration.

3.2 ISDU request as configuration command

Suitable way to design remotely re-configurable the IOLTD is to use the communication features described in section 2.1.3 and illustrated in figure 2.2. Each Index has its Index number. Using this Index number, Index can be accessed with a read and write request. Index number ranges are divided into groups by their specific usage (see figure 3.1). The Index numbers from group "Reserved" are good candidates to be used for the re-configuration, still, keep all valid Index numbers for regular usage and thus be able to simulate regular IO-Link Device [16].

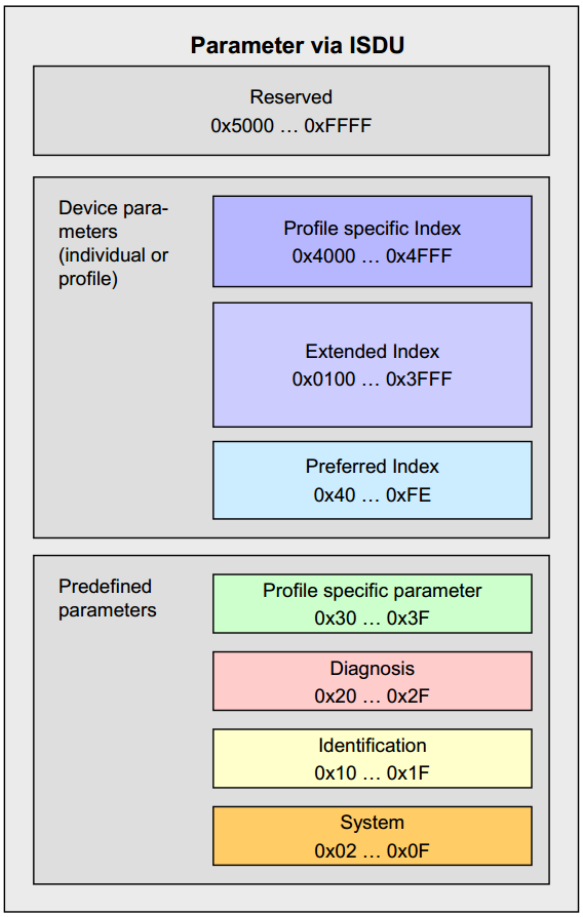


Figure 3.1: Index number ranges groups [20]

In this way, all ISDU requests are handled in a slightly different way in the IOLTD, than in the regular IO-Link Device. Each incoming request is evaluated and processed according to its type. In regular Device, it means decision, whether it is read or write request. The

IOLTD design adds one more option, the command. Illustration can be seen in figure 3.2. The decision, whether it is a command or not, is made based on the Index number of the request. Recommendation from previous paragraph advises to use a subset of Index numbers from "Reserved" group in figure 3.2.

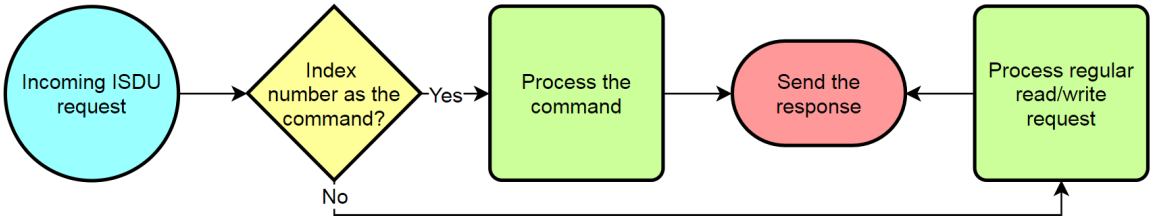


Figure 3.2: ISDU request in IOLTD handling illustration

3.3 Required features to support continuous integration

Following list of requirements was established based on demands directly from the IO-Link testers to satisfy their needs and automation support. Also, IO-Link specification needs to be taken into consideration, when developing the IOLTD. Even when following requirements go beyond the specification, all rules related to Devices must be respected.[20, 17]

1. IOLTD shall be fully configurable via ISDU requests.
    - This requirement is superior to others and is more general. All possibilities of remote re-configuration of the IOLTD shall be designed to use ISDU requests.
2. IOLTD configuration indexes shall be readable and contain current configuration
    - All configuration parameters of the Device (communication, storage, ..) are always present in some ISDU Index and all of them shall be readable over ISDU read request and always contain up to date configuration.
3. IOLTD shall raise IO-Link diagnostic events from event codes supplied via ISDU request
    - Many different events can be raised by IO-Link Device with many different parameters (event mode, type, ..) and IOLTD shall be able to raise such event in reaction to command from ISDU request with event information.

4. IOLTD shall allow up to 32B process data to be configured and accessible over ISDU request
    - IOLTD shall be able to work with 32 bytes of both I/O process data, always keep up to date PD in ISDU Index accessible using ISDU read request and also allow to change process data by ISDU write request.
5. IOLTD shall allow input process data validity and mirroring to be toggled over ISDU request
    - In PD validity is IO-Link feature for Devices to be able to tell there is something wrong with input PD due to some misbehavior or failure.
    - Device shall be able to mirror output PD to input PD.
6. IOLTD shall support dummy parameters of all possible data types and at least one R/W Index in data storage.
    - IOLTD shall have pre-set (or shall be able to have configured) ISDU indexes with different data types, sizes or access rules.
7. IOLTD shall allow data storage reset to default values via ISDU request.
8. IOLTD shall support data storage lock and block parametrization features.
    - Data storage lock serves for IO-Link Devices to prevent their data storage to be uploaded to the Masters, or to be rewritten from backup in Master.
    - Block parametrization shall be supported, so Master can overwrite data storage without rising upload request flag when needed.

3.4 IOLTD in topology

Principle of the remotely re-configurable IOLTD offers several possibilities, how it can be used in IO-Link Masters testing automation. figure 3.3 shows topology position of the IOLTD in common industrial automation system.

One possibility is to write testing program directly into a PLC. However, this solution would be significantly limited by capabilities and hardware sources of the PLC.

Another way to effectively use the IOLTD is to have a PC program capable of executing single commands. This option is not aimed to test automatically but can be very helpful in

cases, where some very special Device configuration or behavior is needed. Such program can be then used for simple fast test also by person, who is not experienced in PLC or common programming.

Way to achieve the most of the IOLTD capabilities is to have working communication channel from PC through PLC to both IO-Link Master and IOLTD. In this case, script or program can be written with ability to change the settings on the Master and also on the IOLTD. This then allows the script to test variety of configuration combinations in almost whole scope allowed by IO-Link specification. Such testing is then able to run automatically for a very long time while gathering diagnostic data for further evaluation.

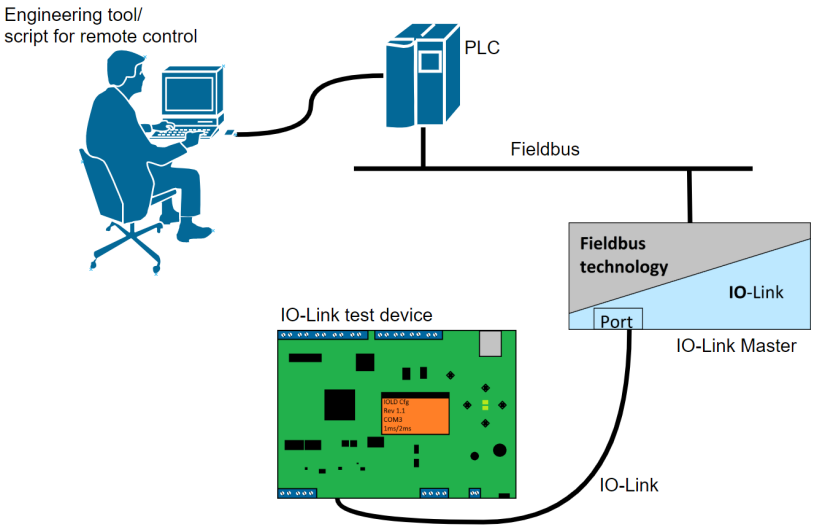


Figure 3.3: IOLTD in the topology

3.5 DevOps in IO-Link Master development

This section describes concept of possible adoption of DevOps in the IO-Link Masters FW development. Because the IO-Link Masters are designated to operate in the industrial automation inside of factories, the real operations are not suitable for the "Ops" in DevOps. One reason for this is, for example, frequent FW update demanded for the continuous delivery, which would require frequent stops of the production lines to process the FW upgrade and therefore significant loses for the customer. Therefore the integration test of the IO-Link Master FW development takes over the role of the operations.

Following sub-sections describe possible way, how particular essential practices in DevOps can be integrated and how the goals can be achieved. Taking figure 2.11 as a pattern, these

sub-sections are plan, build, CI, deployment / CD, operate, feedback.

3.5.1 Plan

The DevOps lifecycle intuitively encourages to plan the development in iterations, where one iteration is the whole cycle from plan to feedback. Using the agile nomenclature, this one iteration is called sprint. The sprint always begins with planning. In the phase of the planning, goals, and milestones for the next sprint are defined based on the project requirements and the feedback from the operations.

In the concept of DevOps IO-Link Master FW development, the goals consist of features, change requests and bugs, which should be implemented/fixed in the sprint. Features are given by overall project needs and are specified on the very beginning of the project, change requests can be taken as new features, which will arise from the operations feedback and the bugs are FW misbehaviors or failures discovered in any level of testing. The concept also counts with commonly recommended 2 to 4-week iterations.

Many sprint planning tools exist and commonly offer functionalities like backlog of tasks (features, change requests, bugs), iteration planning (assigning task to an iteration and to a particular developer), history tracking etc. For this concept, the Team Foundation Server (TFS) was chosen, because of already existing know-how and experiences in the team and also for integration in the MS visual studio IDE, which is used for the development. Core features of TFS planning are following.[7]

- **Product backlog** for all features, change requests and bug reports for the whole product.
- **Kanban** for the workflow visualization.
- **Sprint backlog** for features, change requests and bug reports planned for current sprint.
- **Task board** for meetings on daily basis, to summarize work which is done and which is in progress.

3.5.2 Build

In context of the IO-Link Masters FW development, build means making changes in the code and preparing the release for the end of the current iteration. For coding, IDE and a

VCS are the key tools. The IDE and the VCS should be also chosen based on the integration of particular VCS in particular IDE because the developers use the VCS very often and the integration is not only comfortable but also time-saving. Using these tools, the developer works on particular tasks, which were assigned to him/her for this iteration during the planning.[8]

For this concept, Visual Studio IDE was chosen, not only because of already existing know-how, but also for the integration of the TFS planning system, so all the tasks can be viewed and managed using the IDE. The Visual Studio also offers integration of two version control systems, which are the GIT and the Team Foundation Version Control (TFVC). Both these systems are capable of versioning projects shared between more developers. However, the GIT was chosen for this concept, because the developers are already familiar with this system and also because it uses distributed version control. This offers a possibility for the developer to use version control also locally without need of the connection to the server hosting the VCS.

3.5.3 Continuous integration and Deploy

CI in DevOps for the IO-Link Masters development can be understood as a series of automated processes, which are supposed to ensure early bug detection and save developer's time. In this DevOps concept for IO-Link Master FW development, it means automated FW compilation, unit testing, static code analysis and deployment of tagged releases.[8]

Several CI tools capable of running such automated processes and which are also widely used exist. The most used is Jenkins [6], which is open-source and also generally popular. For this concept, TFS CI capabilities are used. TFS was again chosen for easier integration of other tools and already existing know-how. It can be configured to run all needed automated processes using both time schedule or event trigger. This allows TFS CI to run these processes automatically as a reaction for example to every new push to the git repository and not only in the release or master branches, but also in the development branches.

Testing in the CI has two levels in this concept. First is the static code analysis, which is not running the code itself, but goes through it, understands the structure and is looking for possible threats and mistreatments. Basic static code analysis is already provided in the compilers, but tools like sonarqube or cppcheck offer more advanced analysis. Second are the unit tests. These tests are written by the developers themselves and are made to test smaller parts of the FW separately. Such part is then the "unit" in the unit testing and generally also the smallest part of the program, which can be tested this way.

Deployment to the testers is also done automatically in this concept. Tagged version in the git, which passes the testing is prepared as a package of respective files and sent to the testers. Although the Masters can be also downloaded with the new FW automatically, this cannot be automated generally in an easy way. The test does not need to be located in the same location or infrastructure, so the testing stations do not need to be directly reachable using any kind of network.

3.5.4 Operate

As mentioned before, the integration test substitutes the operations in the IO-Link Masters FW development. Following testing station illustrated in figure 3.4 is proposed in this concept and is capable of simulating the real IO-Link environment, usage, and configuration.

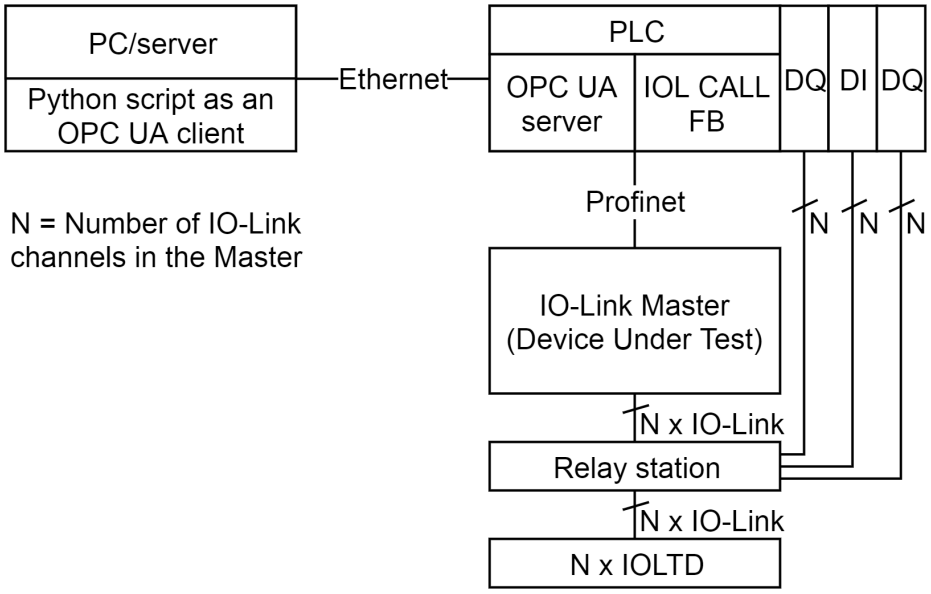


Figure 3.4: Proposed testing station

PC/Server

This machine is the main controller of the testing station. It is capable of remote control of all the other devices (PLC, Master, and IOLTDs) through the python script. The script downloads the new tested FW using ethernet and profinet to the IO-Link Master. Using the OPC UA and the IOL CALL function block (described in next paragraph), ISDU requests can be sent from the PLC to the IOLTD, so the script is capable of remote reconfiguration

of the IOLTD, which uses subset of ISDU requests as the commands as illustrated in figure 3.2 and described in its respective section. The script obtains and evaluates every return value for the ISDU commands, which tells it, whether there was some problem of processing the command in the IOLTD. Further, the script checks diagnostic information sent to the PLC for possible failures of the Master.

PLC

IOL CALL is a function block for the Siemens PLCs capable of sending ISDU requests to the IO-Link Devices through the IO-Link Masters using profinet. The OPC UA is an industrial communication protocol developed by the OPC foundation [5]. OPC UA server hosted on the PLC and used by the client included in the python script is capable of following features used in this station.

- Calling the PLC function blocks (IOL CALL FB)
- Reading of any memory space inside of the PLC
- Writing of any memory space inside of the PLC

First DQ module is responsible for the wire-break simulations, where it can disconnect the IOLTD from the Master using a relay station. DI and the second DQ module serve for the SIO mode testing, where regular DI/DQ signals are sent to the IO-Link Master.

3.5.5 Continuous feedback

The DevOps loop is closed with continuous feedback. In this concept, the testers are responsible for evaluating the failures and misbehaviors, which occurred during the operation. These findings are reported back to the developers, who are responsible for reproducing the behavior based on materials provided by the testers, evaluating the priority of every particular finding and place them in the product backlog. Also, change requests and wanted features can be reported by the testers, for example, features to simplify the Master testing or failures evaluating.

Chapter 4

IOLTD FW extension to support DevOps - solution approach

Based on project goals and requirements, a design of an extension to already existing firmware [15] was created. This design can be divided into 6 following extension groups, which are described in later sections.

- Utilization of already existing firmware implementation
- Storage for ISDU Index data
- Backup of data and configuration in non-volatile memory
- Diagnostic events
- Configuration of IOLTD state and communication parameters
- Process data management
- User interface
- Automatic configuration test

4.1 FW extension modules

The FW extension design relies on following modules to implement all the required features to support DevOps using IOLTD. Module roles and inter-operation is illustrated in figure 4.1. Description of groups and modules follows in upcoming sections.

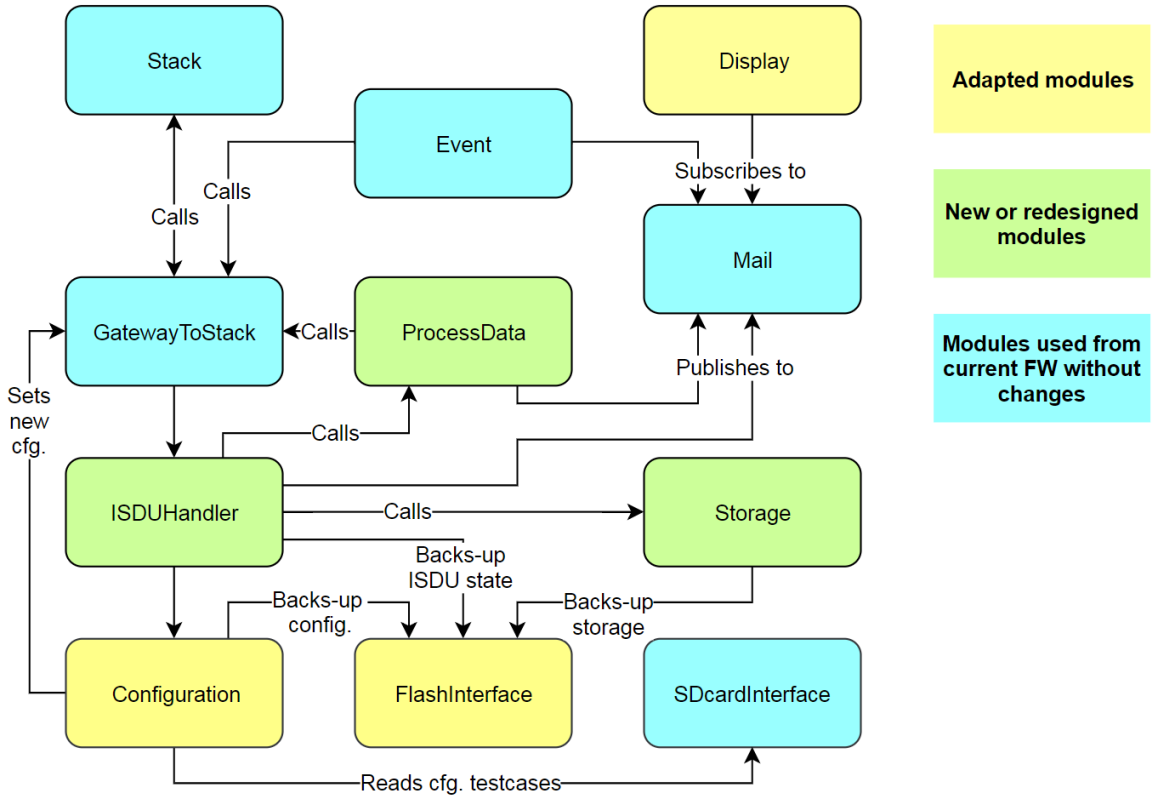


Figure 4.1: Modules inter-operation illustration

4.1.1 New or significantly redesigned modules

- **Storage** - This module will serve as a memory space for storing the data of the ISDU indexes and also be capable of storing/restoring actual state in/from the non-volatile memory.
- **ProcessData** - This module will handle both input and output process data. In case of the input PD, validity and mode are going to be configurable. Supported modes will be:
  - Mirrored - output to input PD
  - Physical digital input
  - Set via ISDU command
- **ISDUHandler** - This module is going to consist of three submodules.
  - DataSorageHandler to handle data storage commands and indexes

  - ISDURequestHandler to handle regular ISDU request like read or write
  - ISDUCommandHandler to handle special write/read request treated as commands to support remote reconfigurability of the IOLTD.

4.1.2 Adapted modules

- **Configuration** - This module will mainly handle active configuration of communication parameters. Further, it will be able to store/restore its state in/from the non-volatile memory and also perform automatic configuration testing, where it will read particular test-cases from a file on the SD card one by one and try the functionality of the configuration, with currently tested IO-Link Master.

- **Display** - This module is handling the display on the IOLTD and currently is capable of displaying communication parameters configuration. In the extension, it will be also capable of displaying information about data storage, process data and active commands from the superior systems (e.g. Master, PLC, ...).

- **FlashInterface** - This module can be currently used to store active configuration of communication parameters in non-volatile memory and will be extended with capability of storing larger sets of data from the Storage module and the ISDUHandler module.

4.1.3 Already implemented modules without changes

- **GatewayToStack** - This module serves as an interface to application-layer functions of the Stack module.

- **Mail** - This module is designed for communication between classes in cases, when immediate response is not needed.

- **Stack** - This module contains fully featured stack as mentioned in section 2.3.2.

- **Event** - This module is capable of storing and handling diagnostic events.

- **SDCardInterface** - This module serves as an interface to FAT file system on the SD card

4.2 FW extension groups

4.2.1 Utilization of already existing firmware

Current firmware is based on object-oriented design in extended embedded c++ language. It implements "Mail System" designed for communication between classes. This system allows shorter response times in interaction with tester/testing program or IO-Link Master. It uses the Publisher-Subscriber model and allows transfer of 4-byte information in one message.

Further, as mentioned in section 2.3.2, fully featured IO-Link stack and FreeRTOS are implemented.

4.2.2 Storage for ISDU Index data

Since this extension significantly depends on work with the ISDU indexes, its design also depends on the storage system to maintain the Index data and information about them. Several types of indexes exist and sometimes overlap in the IO-Link technology. For this reason and memory usage optimization, a decision was made to use one advanced data structure to manage all data, information, and configuration regarding ISDU indexes. Differences of particular storage records are to be held in the properties of the objects stored in this system.

For the test automation support and capability of the re-configuration, the IOLTD has to be able to add and remove the indexes. With the common approach, this would be implemented using dynamic memory allocation. In the case of this design, maximum possible storage is allocated statically and its usage will be handled by the extension itself.

As described and illustrated in section 2.1.3, the ISDU indexes can also contain further subindexes. This, however, applies mainly on statically created indexes given by IO-Link specification and dynamically created indexes only need Subindex number 0, which only addresses complete data of its superior Index. Allowing dynamic indexes creation with only Subindex 0 can significantly simplify storage design using static memory allocation.

4.2.3 Backup of data and configuration

To simulate real IO-Link Devices, the IOLTD has to be capable of keeping its state also after a power reset. Regular IO-Link Device has one fixed state, which is always valid. Such state includes communication parameters configuration, ISDU indexes, subindexes and also their data. To simulate these conditions, IOLTD has to be capable of storing its whole state in the

non-volatile memory during operation and restore it after start-up. Since IO-Link Masters are developed to operate with regular IO-Link Devices, this feature is a must, because IO-Link Master does not need to be able to handle the same Device with the different state after power reset. This, for example, involves ISDU indexes, their number, data, size of data storage or its arrangement.

If such inconsistency occurs in test set-up, tested IO-Link Masters and then IOLTDs could go into an undefined state. This is however strongly undesired situation because it could easily interrupt automated testing and in the final consequence also suspend the desired continuous integration.

4.2.4 Diagnostic events

Device events are sent as a pre-defined 2B event codes and a so-called Event-Qualifiers, whose structure is shown in figure 4.2 and which carry additional information about the event. List of all possible event codes can be found in the IO-Link specification[20]. Event-Qualifier consists of MODE, TYPE, SOURCEa, and INSTANCE. As illustrated in figure 2.7, events are reported using diagnosis channel in On-request communication.

Event raising in the IOLTD is going to occur as a reaction to the particular ISDU command supplying all the information needed to build the Event-Qualifier in the data part of the command. Following sections are describing particular parts of the Event-Qualifier. Event-Qualifier will be built according to structure in figure 4.2.

Figure 4.2: Event-Qualifier structure

INSTANCE

The instance bits of the qualifier indicates the particular source of the event in the device. In other words, on what level of the device's firmware the event occurred.

Value	Definition
0	Unknown
1 - 3, 5 - 7	Reserved
4	Application

Table 4.1: Event instance values

SOURCE

Source is one-bit information indicating origin of the event, which can be Device or Master.

Value	Definition
0	Device
1	Master

Table 4.2: Event source values

TYPE

Type of event indicates whether it is notification, warning or an error.

Value	Definition
0	Reserved
1	Notification
2	Warning
3	Error

Table 4.3: Event type values

Mode

IO-Link technology recognizes three modes of events. single-shot indicates short-term events, Event appears indicates start of long-lasting event and Event disappears indicates its end.

Value	Definition
0	Reserved
1	Event single-shot
2	Event disappears
3	Event appears

Table 4.4: Event mode values

4.2.5 Configuration of IOLTD state and communication parameters

To support full remote reconfiguration of all parameters needed to simulate every possible state of common IO-Link Device allowed by the specification, following properties of the IOLTD are going to be implemented as reconfigurable using ISDU requests as commands.

Communication parameters

Communication parameters reconfiguration will occur as a reaction to ISDU command, where new parameters to be set will be included in the body of this command. Reconfigurable communication parameters will be:

- **MinCycleTime** - Informs the IO-Link Master about the shortest possible cycle time, which is the Device able to achieve. Smaller MinCycleTime means faster responses in case of bigger operations and transmissions, which need more cycles to proceed, but also needs respectively powerful HW resources. Device will support reconfiguration to all acceptable values of MinCycleTime given by table 4.5.

Cycle Time	Step	Comment
0.4 to 6.3 ms	0.1 ms	0.4, 0.5, ..., 6.3 ms
6.4 to 31.6 ms	0.4 ms	6.4, 6.8, ..., 31.6 ms
32 to 132.8 ms	1.6 ms	32, 33.6, ..., 132.8 ms

Table 4.5: Acceptable cycle time values

- **Baudrate** - Transmission speed defined by the IO-Link specification. Acceptable values are listed in table 4.6 below. Faster transmission speed can result in possibility

to achieve shorter cycle time, but again, needs respectively powerful HW resources. All acceptable values will be reconfigurable via ISDU command.

Communication mode	**baudrate**
COM1	4.8 kbit/s
COM2	38.4 kbit/s
COM3	230.4 kbit/s

Table 4.6: Acceptable COM values

- **M-sequence Capability** - Informs about frame type used for communication (see figure 2.5) and whether the ISDU is supported by the Device. Frame types can differ between preoperate and operate mode of the Device. Preoperate is the communication mode right after the start-up of the Device, where only On-request data take place. Operate mode takes place after preoperate as regular communication mode of the Device also with process data and all communication features as described in section 2.1.3. Structure of M-sequence Capability can be seen in figure 4.3 and all possible combinations will be reconfigurable via ISDU command.

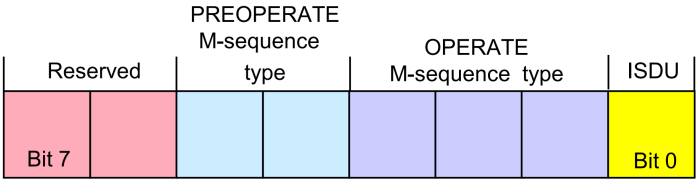


Figure 4.3: M-sequence Capability structure

- **RevisionID** - This information consists of two 4-bit values together defining the protocol version used by the Device. First value is so-called MinorRev and second is MajorRev. Taking protocol version 1.0 as an example, MajorRev equals 1 (before the dot) and MinorRev equals 0 (after the dot). Acceptable values for both MajorRev and MinorRev are 0-15. Structure of RevisionID parameter can be seen in figure 4.4. All acceptable values will be possible to reconfigure via ISDU command.

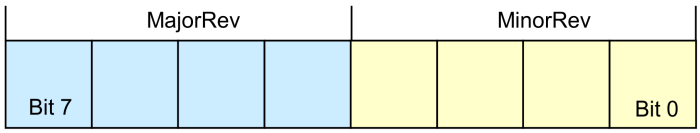


Figure 4.4: RevisionID structure

- **ProcessData (In and Out)** - These two parameters informs about the length of the process data. They can be set independently, but have the same structure. The length is encoded in a single byte and acceptable values are 0-16 bits and 3-32 bytes. The MSB of this byte is called the "Byte bit" and indicates whether value stored in first 5 LSBs means bits or bytes. The actual value in these LSBs can be 0-31, so in case of bytes, 1 has to be added to satisfy 3-32 bytes condition. Second MSB also indicates support of switching signal for SIO mode. Structure of ProcessData parameter can be seen in figure 4.5. All acceptable values will be possible to reconfigure via ISDU command.

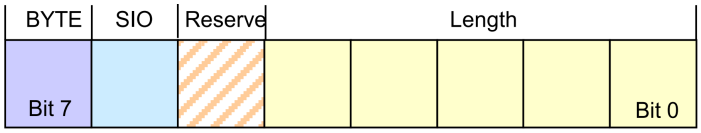


Figure 4.5: ProcessData (In and Out) structure

- **VendorID** - 16-bit value containing unique number defining the vendor of the particular Device. These numbers are assigned by the IO-Link community and their list can be found on the websites.[1]
- **DeviceID** - 24-bit value uniquely defining each IO-Link Device of the vendor. Acceptable values are $1 - 2^{24}$. All acceptable values will be possible to reconfigure via ISDU command.
- **FunctionID** - 16-bit value for future use. Only acceptable value in current version of the IO-Link specification (1.1) is 0

ISDU state

ISDU state of the IO-Link Device is given by all the indexes and subindexes, which are supported and also by their data. According to the IO-Link specification, Indexes are mandatory, conditional or optional.

- **Mandatory** - Indexes, which must be implemented to be consistent with IO-Link protocol specification. These indexes are:
  - **Direct Parameter Page 1** - Defined set of communication parameters described in section 4.2.5.
  - **Direct Parameter Page 2** - Vendor specific set of parameters.
  - **Data Storage** - Index with information about all data storage parameters. See section 4.2.5 below for more detailed information.
  - **Vendor Name** - Contains the name of the vendor assigned to the VendorID in maximum of 64 B data object.
  - **Product Name** - Contains complete product name in maximum of 64 B data object.

  Besides several parameters of the DPP1, all mandatory indexes and subindexes are read-only data objects.
- **Conditional** - In current protocol specification, only Index "Device Access Locks" and its Subindex "Data Storage" is conditional. When Data storage feature is supported by the Device, this Index and Subindex is then mandatory and contains information, whether the Data storage is locked for changes or not.
- **Optional** - IO-Link specification also lists several optional indexes with defined purpose, which are not required for communication. In comparison to mandatory indexes, optional indexes are informative. Complete list can be found in the IO-Link specification. [20]

All mandatory Indexes will be implemented in the IOLTD statically and since it will support Data storage, also the conditional "Device Access Locks" Index will be implemented in this way. All other possible indexes of variable sizes and contents will be configurable using the ISDU commands.

This will make the IOLTD capable to simulate every possible ISDU state allowed by the IO-Link specification.

Data storage

Data storage is a mechanism enabling consistent up-to-date buffering of the (IO-Link Device specific) subset of ISDU Indexes (parameters) on upper levels like PLCs or in case of IOLTD,

mainly the tester or his script. The IO-Link Device also keeps additional information about the Data storage. These information are:

- **Size** - Memory space requirements of all Data storage Indexes information and data. Maximum Data storage size is 2048 B.
- **Revision** - Checksum of data storage or revision number indicating changes.
- **Index List** - Linked list of all Data storage Indexes. One list can contain up to 70 Data storage indexes. When one Index List is full, another one has to be implemented and linked from previous Index List. Since the 70 DS Indexes is only upper bound, IO-Link specification allows implementing much more Index Lists. With maximum DS size of 2048B, the worst case is 409 DS Indexes with size 1B and each of them having its own Index List.

Next purpose of the Data storage is ability to store Device's parameter state into the Master using Data storage upload mechanism. On every change in any Data storage Index, which is not caused by the IO-Link Master, the Device adapts Data storage revision and raises an upload event. Using the Data storage information mentioned above, Master will read Data storage indexes one by one from the Device and stores them in its own storage. This allows for example replacing the faulty Device with a new one and using Data storage download mechanism force the state stored in Master into new Device and let it continue exactly where its predecessor ended.

IOLTD extension will be capable of dynamically creating any combination of Data storage, DS Index count and even maximum possible number of Index Lists. Also, all other features regarding data storage mechanism will be implemented.

4.2.6 Process data management

Process data management design distinguishes between the input and output process data. Output process data are normally sent from the Master to the Device and further to its physical output pins and this functionality is already implemented in current version of IOLTD. In the IOLTD FW extension, special ISDU Index will be implemented for both process data to make them accessible using ISDU read request.

From the test automation with the IOLTD support point of view, Input process data handling is much bigger challenge. Three modes of Input process data handling will be implemented to provide comfortable testing capabilities.

- **Physical DI** - This mode is already implemented in current version of the IOLTD FW. input from 4 physical pins is sampled and sent to the Master either as a 4-bit or a 4-byte value. From these two options can be chosen by a switch before FW compilation.
- **Mirroring** - In this mode, the IOLTD FW extension will take output process data sent by the Master and sets them also as input and send them back to the Master.
- **ISDU** - For this mode, an ISDU command will be implemented. This command will be capable of setting new input process data. These data will be stored into particular ISDU Index and also immediately set to the stack to make them active as soon as possible.

4.2.7 User interface

The IOLTD is equipped with a simple 4x10 symbol display for an informative data. In the current FW, this display serves for a basic information about active communication setting (cycle time, COM) and active protocol revision.

In the FW extension, following three types of information will be also available on the display.

- **Active command** - This information will contain data about the procedures currently happening in cooperation with the Master. For example Data storage uploading, downloading etc.
- **Data storage information** - Data storage size, number of DS ISDU indexes and number of Index Lists currently active in the IOLTD will be displayed.
- **Process data information** - Process data length of both in and out PD and input Process data mode will be displayed.

Due to limitation of 40 symbols on the display, there will be different screens with these types of information. Tester will be able to change between the screens using HW buttons installed on the IOLTD.

4.2.8 Automatic configuration test

Despite that the IOLTD will have all its communication parameters reconfigurable via ISDU commands, reconfiguring in this way to test larger subset of all possible communication parameters combinations could last for a long time due to communication overhead. For

this reason, feature "Automatic configuration test" will be implemented. Since the IOLTD is already equipped with micro SD card slot, it will use SDCardIterface module to access the SD card file system. On the SD card will be placed file containing particular test cases. This solution will offer possibility to create even very large sets of test cases and also can be easily used by the tester, who will be able to use test case generating tool to create it and place it into the IOLTD. This will allow an effective approach of a combinatorial testing [3] to provide significant test coverage while keeping the time requirements of the testing acceptable.

Device will then take test cases one by one and test them, whether the Master is able to set-up the configuration and communicate.

4.3 Configuration application extension

4.3.1 Current implementation

The PC GUI application for the IOLTD reconfiguration via USB is written in C# language and uses the Model-View-Controller pattern. This application is capable of changing the communication parameters of the IOLTD and the event raising. These capabilities are processed using read and write commands, which are sent over direct USB connection and developed communication protocol.[2] Following figure 4.6 illustrates the implementation. The communication is briefly described below.

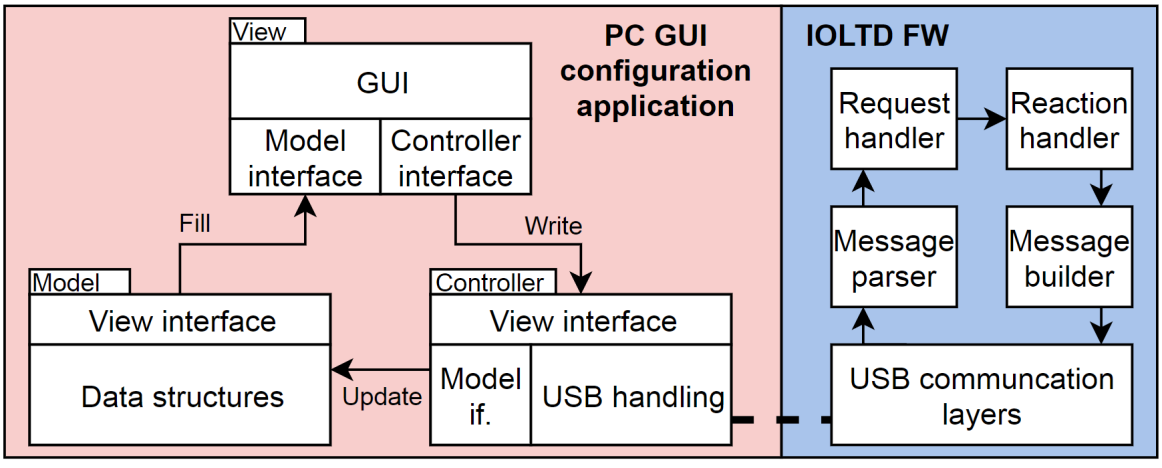


Figure 4.6: Brief USB configuration interface illustration

Communication

The USB communication is implemented based on the libraries from ST Microelectronics as a vendor of the onboard MCU and its USB peripheral. This vendor also provides USB driver for the PCs. Once the connection is established, PC GUI application is sending write and read requests to the IOLTD, which handles these requests and sends responses, if needed. In the USB communication carrying the requests, the first two bytes of the payload serves as a header, which is used to distinguish the communication channels in both PC GUI application and the IOLTD FW and the read/write requests. Currently implemented channels are configuration and event. When the USB packet is accepted in the IOLTD as a reaction to an interrupt, its payload is stored for further use. When the USB FreeRTOS task is scheduled, it checks whether some packet was accepted and needs to be handled. If so, the first byte of the payload is evaluated to determine the communication channel and the second is evaluated to distinguish the read and the write requests. Rest of the payload is then cast to a respective data structure and sent to the respective handler function.

4.3.2 Extension design

This design extends the current implementation with capability to read and display the ISDU state of the IOLTD. This capability can be useful when a significant communication failure of the Master or the IOLTD occurs, the communication will not be restored and thus, the ISDU state will not be readable using ISDU requests. The USB connection then can be used to read the configuration and the ISDU state of the IOLTD to identify the cause. Read of the ISDU state in this design is done using read requests to particular ISDU Indexes.

Communication

Communication extension design adds ISDU channel besides already implemented channels. The Configuration and event channels are distinguished using first byte of the payload, which carries value 1 for the configuration and value 6 for the event. This design adds value 7 in the first byte for the ISDU channel and the ISDU Index and Subindex numbers in the payload. Respective data belonging to this Index and Subindex are returned in the response. Following figure 4.7 illustrates the payload division.

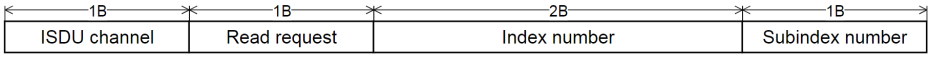


Figure 4.7: ISDU channel payload illustration [2]

Request handling

This design extends the request handling in respect to previous implementation. Handling functions are called based on the channel specific first byte of the payload. Function to handle ISDU read requests from the USB is added in the USB task. This function calls directly the ISDUHandler module in the same way, as GatewayToStack does in the modules inter-operation illustrating figure 4.1. In this way, any ISDU Index can be accessed and thus also the complete ISDU state of the IOLTD. After the requested data are returned by the ISDUHandler module, the USB response message is built and sent back to the GUI application.

GUI application extension

Current GUI of the application displays USB communication parameters, Application logger and then uses tabs to choose between the configuration and the event interface. This extension adds an ISDU tab with UI for ISDU read requests as illustrated in figure 4.8.

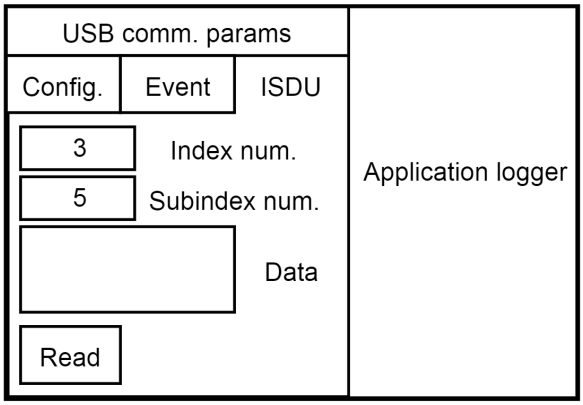


Figure 4.8: GUI extension illustration

The MVC pattern is extended with the ISDU read requests capability respectively to the read request for configuration in the design. Data structure with ISDU Index number, Subindex number, and the read data are added to the Model. Controller is then extended with capability of creating message from the ISDU read request with the respective channel identification byte.

Chapter 5

IOLTD FW extension to support DevOps - implementation

This section describes the implementation of the IOLTD extension to support DevOps in IO-Link Masters development, significantly builds on section describing the design and uses similar structure. There is described implementation of particular group and modules designs.

5.1 Development requirements

In addition to common requirements given by general coding guidelines and already implemented parts of firmware, there is also requirement to use exclusively static memory allocation. This requirement has several reasons. Static allocation offers determinism in the data placement in memory right after linking of compiled program, which ensures ability to fit all variables in memory. Further prevents memory leaks during mistreatment with dynamic allocation, which is hard to find during the debugging. On common embedded system without OS capable of memory management, static allocation also prevents memory fragmentation caused by repeated allocation and free of memory. In case of hard real-time system, there would also be reason of non-deterministic time consumption of dynamic allocation.

5.2 FW extension modules implementation

5.2.1 Mail module

Mail module is implemented using the publisher-subscriber pattern to offer alternative way for inter-class communication. It is used in non-time-critical cases, where the whole procedure does not have to be done immediately. For example for informative content on the display, where the information can be delayed in tens of milliseconds without any impact on the function or usage.

Mail system itself is implemented as a message queue with basic access functions. Classes registered as the publishers create new message (with structure as illustrated in figure 5.1) and call Mail system function to enqueue (send) it. The message contains unique identifier for every source class. These identifiers are then used by the classes, which are registered as the subscribers. During the registration of new subscriber class, list of the identifiers has to be provided to be able to obtain particular messages. This list serves as a filter and specifies, which messages will be received based on their source.

The message can carry up to 4B of information. This size offers two possibilities of usage. First, it can carry smaller structure or information directly, or second, pointer to the memory can be set in the message body to carry information, where some bigger structure is located. For the second usage, the particular structure type has to be determinable by the source in the message header.

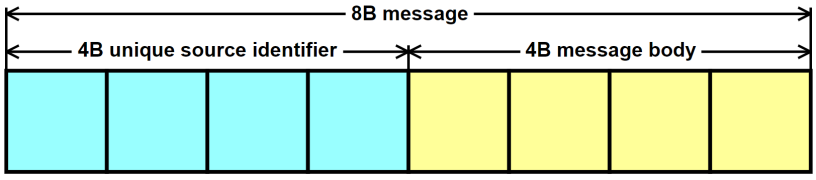


Figure 5.1: Mail system message structure

5.2.2 Event module

Event module consists of EventManager and EventQueue. Stack module is capable of sending one event at a time, so the queue of events is needed. New events are obtained from the mail system, are handled by the event manager and pushed into the queue. Event manager then periodically pops events one by one and sends them into stack using GatewayToStack interface.

Event manager also distinguishes between appear, disappear and single-shot events. In case of the single-shot, event is sent to the stack and deleted from the queue. For appearing or disappearing events, event manager handles list of active events and this list is updated with every appearing or disappearing event.

5.2.3 SDcardInterface module

SDcardInterface module is an abstraction layer with access to SD card and its file system from the application. Currently, FatFS[4] module is used to access SD card file system. FatFS was implemented because it is specifically designed for embedded systems, written in C language and compatible with FAT/exFAT file systems. Another benefit of this module is usage and naming of its functions, which both are very similar to standard libraries for c or c++ languages and therefore intuitive.

In the IOLTD FW extension, this module is used for automated configuration testing. Test cases are read from a CSV file placed on the SD card and eventual faulty configuration combinations are stored back in the text log file.

5.2.4 FlashInterface module

FlashInterface module is an abstraction layer with access to Flash module, which is responsible for handling SPI flash access. SPI flash is in the IOLTD FW extension used for non-volatile storing of configuration, ISDU Index information, and the storage. Each of these three structures uses different subsectors of the flash memory, so backup or restore of each structure can be handled independently, since the whole subsector of SPI flash has to be erased before any write to it. Modules using FlashInterface are then ISDUHandler, Storage and Configuration.

Flash module contains a request queue, which every module can enqueue to using FlashInterface. This queue is checked periodically for new waiting requests. Each request contains information about amount of data to be read or written and a pointer to the respective structure. During the communication with SPI flash, the task handling the request must not be rescheduled. Therefore FreeRTOS rescheduling is disabled and requests are divided into max. 256B chunks, so the transfer does not take too much time and the other tasks can be scheduled on time.

5.2.5 Display module

Display module handles the information to be displayed and the display itself. Because of the limitation of 4x10 characters, which can be displayed at once, each type of information has its own screen and switching between the screens is done using HW buttons on the IOLTD.

Displaying of the information is not time-critical procedure, so all the information are sent using mail system. Display module is, therefore, one of the subscribers of the mail system.

In the IOLTD extension, three screens are added to the display module.

- **Command log** - Screen with last four commands from the Master. List based on FIFO approach is used to keep these commands in the order, in which they were sent by the Master. Two types of commands are displayed:
  - Parametrization commands
  - Data storage commands

  Also, information about finishing the handling of particular commands is displayed at the end of each row.
- **Data storage** - ISDUHandler module, as the keeper of data storage state and also as the mail system publisher sends new mail as a reaction to every change in the data storage. Each of these mails is accepted by the display and respective screen information is edited. These information are following.
  - Size of all data of ISDU indexes currently configured in the data storage
  - Number of ISDU indexes currently configured in the data storage
  - Number of Index lists currently configured in the data storage
- **Process data** - ProcessData module, as the keeper of all information about the process data and also as the mail system publisher sends new mail as a reaction to every change in the process data state. Process data them self are not displayed, only following information is displayed and updated on their every change.
  - Output process data length
  - Input process data length
  - Input process data mode
  - Input process data validity

5.2.6 Configuration module

Communication module has several responsibilities regarding configuration of communication parameters. It is also the mail system publisher, since the main screen, which was already implemented in previous state of the IOLTD, displays information about current configuration and the ProcessData module needs information about configured data lengths.

This module maintains active configuration of the IOLTD's communication parameters. Every change of these parameters is handled by the Configuration module. It responsible for setting configuration to the Stack structure using GatewayToStack module, reinitializing the communication with new settings and also for backup and restore of new configurations.

Another functionality provided by this module is automated configuration combinations testing. The IO-Link Master has to be tested for various communication parameters combinations, therefore automated testing functionality is implemented. It uses SDCardInterface to access test case file with pre-generated test cases, to test ability of the Master to communicate using every communication parameters combination. Tested parameters are following.

- COM
- Cycle time
- On request data lenght
- Input process data length
- Output process data length

5.2.7 Storage module

Storage module contains allocated memory space and an access interface. Storage serves for storing regular indexes, data storage indexes and for configuration. Whole storage is statically allocated space, where particular records can be allocated dynamically. Each record is implemented as an object, which points to the particular position in the storage as for start of its own memory space. Pre-set indexes according to specification also contain subindexes pointing to the particular subset of their Index data. Illustration of memory space usage can be seen on the figure 5.2.

Dynamically created indexes only contain Subindex number 0, which is default and points to the Index start. Such approach results in easier implementation while fulfilling tester requirements and needs. Other modules access storage for read or write and passing data

between modules is implemented as passing pointers into the storage. This passing of pointers is used in both ways - directly or using the mail system, depending on usage and desired response time. As result of such design, structure of particular records is known across all modules with access to storage.

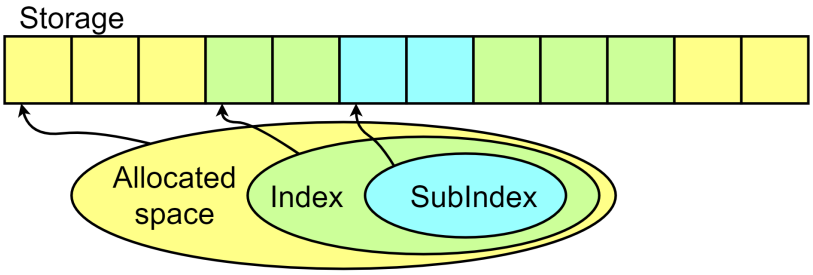


Figure 5.2: Storage memory space usage illustration

The storage access interface implements following operations.

- Add record
- Remove all dynamically created records
- Modify record
- Get record data
- Get storage size
- Get unused space

5.2.8 ProcessData module

Module ProcessData maintains current process data of the Device. Manipulation with the process data is done by direct R/W access to storage, where current process data are stored for ISDU Index accessibility. Command for such manipulation will be either obtained from the ISDUHandler module, or by their real change. Each process data change will result in their storing and setting into stack.

Module handles output and input process data differently. Output process data are obtained from the Master and set to the digital output pins. Output process data length

configuration possibility is offered by the module. Input process data have more configuration possibilities in the IO-Link technology. Besides the length, also validity and mode are configurable.

This module is also both publisher and subscriber to the mail system. Subscription is used for obtaining information about changed configuration of the Device and setting the current process data lengths. Publishing Process data information on every change then allows Display module to give up to date information on the display.

5.2.9 ISDUHandler module

ISDUHandler module is the core module allowing IOLTD remote reconfiguration using ISDU requests as the commands. ISDUHandler parses incoming command and decides about further actions to be taken. It also handles regular read and write requests. Both requests and commands are coming from the GatewayToStack module.

This module maintains information about all active indexes in the Device. When new Index is created or old are removed, module edits its structures to keep always up to date information. Kept information about each Index is following.

- Index type (DS ISDU Index, regular ISDU Index)
- Data length
- Storage position
- Access restrictions (R/W)

Statically defined (hardcoded) indexes, which are demanded by the specification can also contain additional information about subindexes - list of subindexes and number of subindexes.

This module is also publisher to the mail system. It is also responsible for handling changes in the data storage, so it publishes these changes to the mail system, to be sent to the Display module, where these information are displayed to the user.

Since the ISDUHandler module is responsible for handling all ISDU commands and requests, it also has to directly access or communicate with the biggest number of other modules in the FW extension. Procedures and more details about handling of particular commands are described in later sections. Accessed modules are following.

Using the mail system:

- Event

- Display

Directly:

- GatewayToStack
- Configuration
- Storage
- PorcessData

5.3 FW extension groups implementation

5.3.1 Utilization of already existing FW

As mentioned, main part of previous implemented FW is the mail system. Following table 5.1 summarize the mail system usage by the FW extension.

Module	Subscriber	Publisher
ISDUHandler	-	DS info, New event req.
Configuration	-	New cfg. parameters
ProcessData	New cfg. parameter, New input data	PD info
Display	DS info, New cfg. parameters, PD info	-
Event	New cfg. parameters, New event req.	-

Table 5.1: Mail system usage summary

Following FreeRTOS tasks are used in the IOLTD FW extension.

- **Stack task** - Handling ISDU requests, events and the output PD.
- **Mail task** - Handling the Mail system queue.
- **Flash task** - Handling the SPI flash request queue.
- **Event task** - Handling the event queue.
- **PD In task** - Handling the input PD.
- **Auto cfg. test task** - Running the automatic configuration combinations testing.

5.3.2 Storage for ISDU Index data

Initialization

During the initialization at the Device's start-up, a memory space for the storage is allocated. Size of the allocated space is calculated as $size = statically\_defined\_indexes\_size + maximum\_data\_storage\_size$, where the statically defined indexes are the ones demanded by the specification and hardcoded in the IOLTD and $maximum\_data\_storage\_size$ is 2048B as defined also in the specification. List of statically defined indexes can be seen in the table 5.2.

Index name	Index number	Content
DPP1	0	DPP1 parameters filled on startup
DPP2	1	For future use
Sytem command	2	W only for commands from the Master/PLC
DS Index	3	DS info. including the initial IL
Device access locks	12	DS and parametrization locks
Vendor name	16	"SIEMENS"
Product name	18	"IOLTD"
HW Revision	22	HW revision string
FW Revision	23	FW revision string
APP specific tag	24	"Best IOL Device!"
HW Revision	22	HW revision string
PD In	40	Up to date input PD
PD Out	41	Up to date output PD

Table 5.2: Statically defined ISDU Indexes

Storage management

An up-to-date pointer to a first empty position in the storage is kept. New records are stored on this first empty position and after removing the records, clean-up is executed and the pointer is reset to the default position, so that first empty position is always at the end of the currently occupied storage. Following flowcharts 5.3 and 5.4 illustrate the procedure.

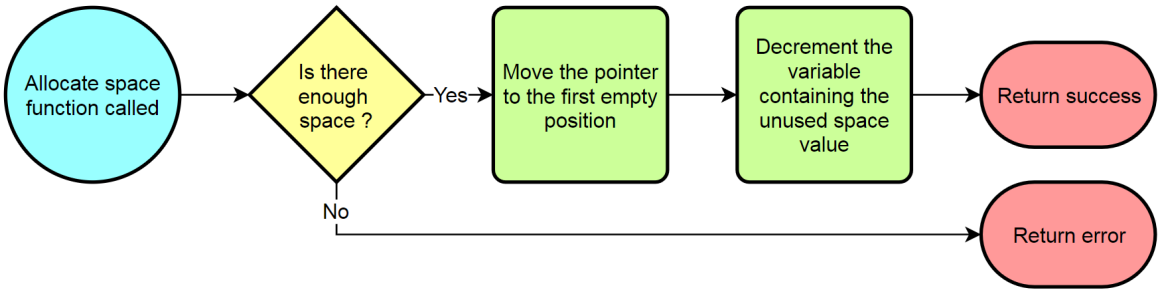


Figure 5.3: Storage space allocation

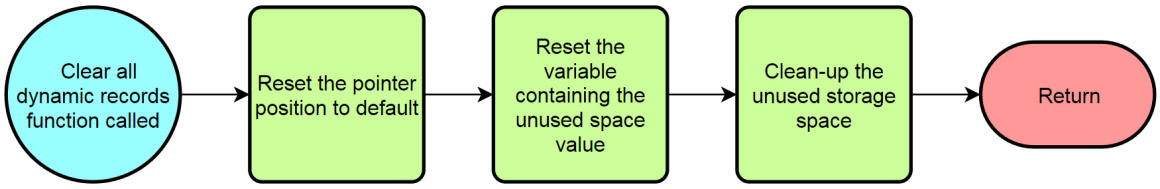


Figure 5.4: Storage space reset to default

5.3.3 Backup of data and configuration in non-volatile memory

To make the IOLTD state persistent also after IOLTD power reset, SPI flash is used to backup ISDU data and configuration, particularly M25PX32 flash memory. This memory has (among others) following features.

- SPI interface
- 32Mb space
- 4KB subsector granularity
- Bulk erase (32Mb)

Configuration, the whole storage, and the ISDU state are stored into SPI flash. While configuration is automatically stored on its every change, storage and ISDU state are stored on demand by ISDU command. Each of these three structures uses different subsectors, so each can be handled independently. Different data in the same subsector cannot be handled independently, because each write into the SPI flash is preceded by erasing of the whole subsector. Summary of the SPI flash usage is described in table 5.3.

Structure	Subsector num.	Address
Configuration	0	0x0000 - 0x0FFF
Storage	1	0x1000 - 0x1FFF
ISDU state	2	0x2000 - 0x2FFF

Table 5.3: SPI flash subsector usage

5.3.4 Diagnostic events

Event code and all the information contained in the Even-Qualifier can be supplied over the ISDU command. This command is going to be parsed in the ISDU handler, where new mail is created with all needed information. This mail is sent to the Event module responsible for raising the event through GatewayToStack module and further to Stack and IO-Link Master. Following flowchart 5.5 illustrates the procedure.

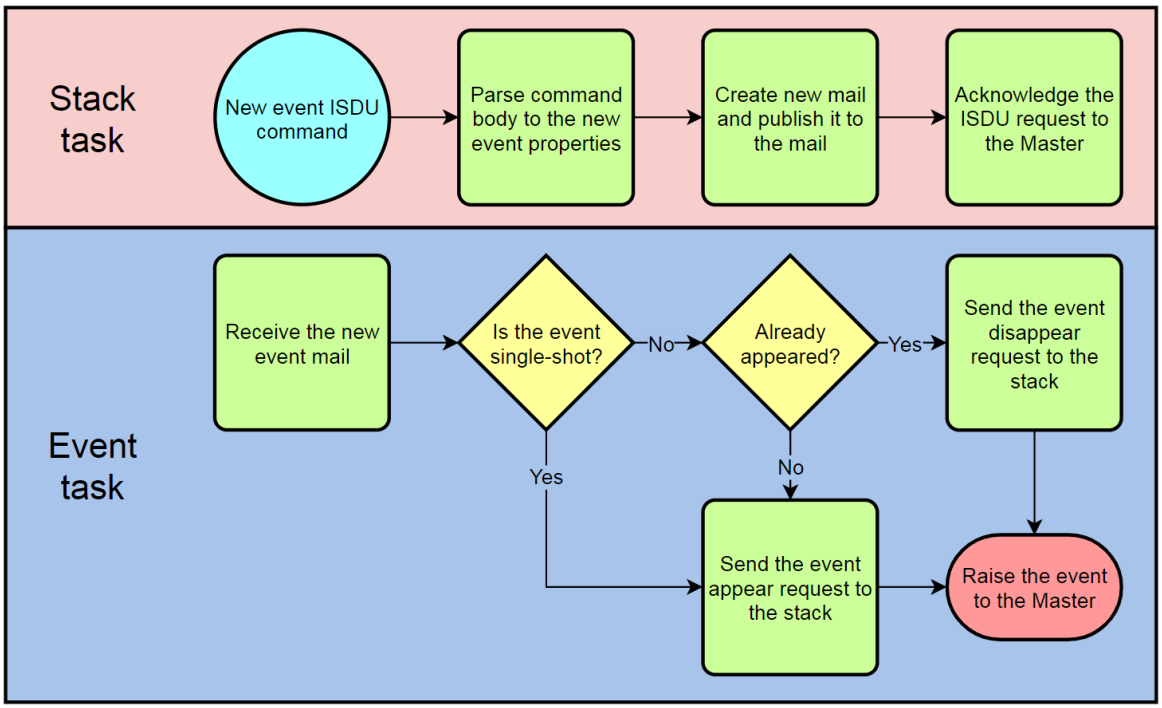


Figure 5.5: Raising the new event

5.3.5 Configuration of IOLTD state and communication parameters

Communication parameters

Following parameters are reconfigurable using ISDU commands.

- Baudrate
- Min cycle time
- M-sequence capability
- RevisionID
- Process data lengths
- VendorID
- DeviceID
- FunctionID

Besides the baudrate, all these parameters are also part of the DPP1. Only RevisionID and DeviceID parameters are also writeable, all other parameters are read-only in the DPP1. The RevisionID and DeviceID are also writeable, to allow the Master to set compatibility mode with older IO-Link revisions and Device's version. For this reason, special ISDU command is used for internal write access to parameters and the DPP1. This command can be used either to access the whole DPP1 parameters or only one parameter by specifying particular Subindex of the command. An extra command is implemented for the baudrate parameter since it is not part of the DPP1. As a reaction to any of these two commands, new configuration is checked for validity and set to the stack. If the new configuration is valid and operational, it is stored to non-volatile memory. Following flowcharts 5.6 and 5.7 illustrate the procedure.

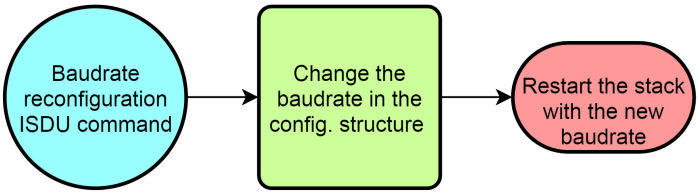


Figure 5.6: Baudrate reconfiguration

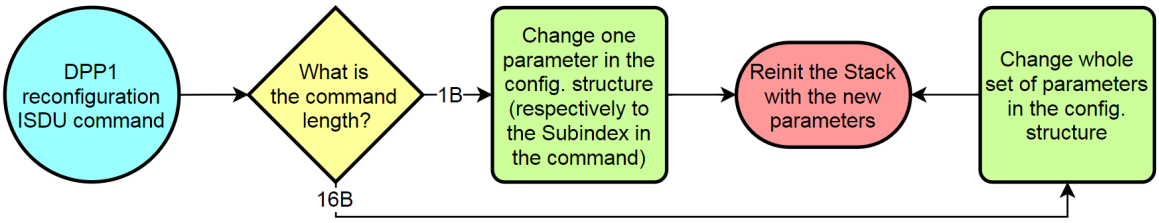


Figure 5.7: DPP1 reconfiguration

ISDU state

ISDU state in the IOLTD is defaultly given by the statically defined ISDU indexes and is further configurable using ISDU commands. Commands influencing the ISDU state are capable of creating and removing indexes, creating Index lists and clearing the whole dynamically created state. The create and clear procedures are illustrated on following flowcharts 5.8 and 5.9.

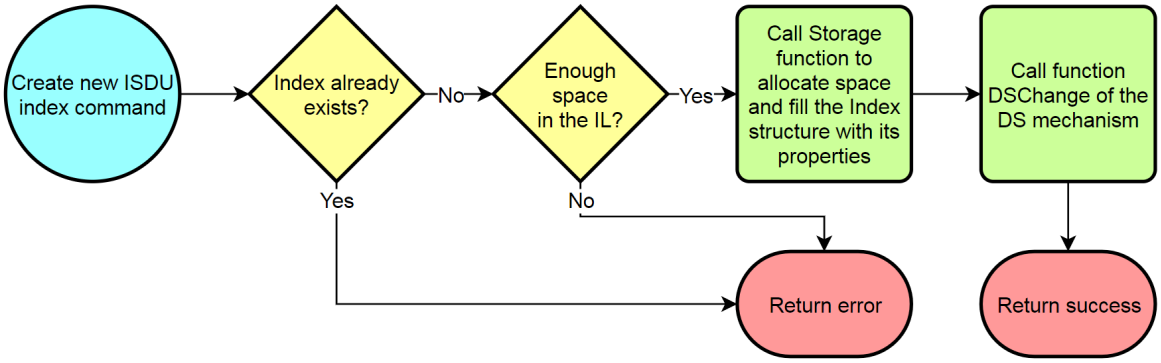


Figure 5.8: Creation of new ISDU Index

Figure 5.9: Clearing the whole dyn. created state

Data storage

The IOLTD keeps following (mandatory) information about the data storage.

- Size
- Revision
- Index list

When any change in any data storage ISDU Index occurs and is not initiated directly by the Master itself, following procedure defined by the IO-Link specification must happen. The procedure is also illustrated on flowchart 5.10 below.

1. Edit the ISDU Index data
2. Increment or recalculate data storage revision (The revision can be both checksum or revision number)
3. Edit data storage size (except simple write request, which does not affect the size)
4. Rise the data storage upload request event to the Master

Two exceptions for the raising of the event exist.

1. A bulk parametrization is active. This, however, can only be initiated by a superior entity to the Master, for example, the PLC program or the tester's script.
2. Data storage download from the Master to the Device is active.

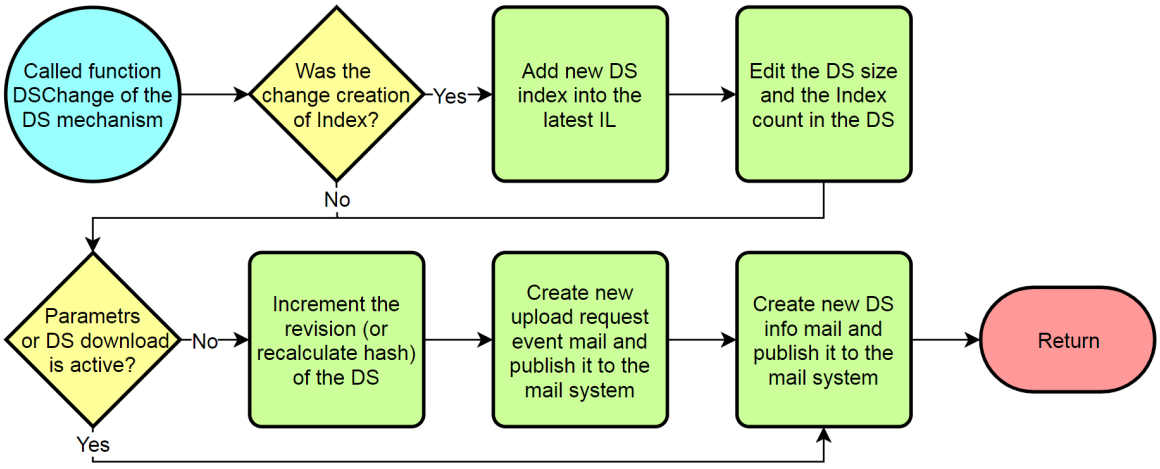


Figure 5.10: DS Change handling

Index list reconfigurability mechanism

When the data storage is enabled in the IOLTD and therefore Data storage Index is implemented, first Index list is already defined in this Index on Subindex 5. It contains links to all defined ISDU Indexes belonging to the data storage and two terminating zero octets at the end. These links consist of 2 Bytes of the Index number and the Index size.

Index List contained in the DS Index is the only active Index List in the Device, therefore all newly created ISDU indexes are appended to this Index List. Because this Index List must exist by default in the IOLTD, its final size is not known during the initialization. It is the only Index List with dynamically changeable size. Therefore also the size of the DS Index, which contains this Index List, changes with every new ISDU Index added to this Index List. Size of any other Index List is then tester's responsibility to define when it is dynamically created. Then, when new ISDU Index is created, it is always stored in the last Index List.

The IOLTD extension implements following procedure on the ISDU command to create new Index List. This procedure is also illustrated on flowchart 5.11 below.

1. Create new ISDU Index (not contained in the DS, size of the Index is provided in the command body)
2. Overwrite previous last Index List terminating bytes with the number of ISDU Index from 1.
3. Mark the new Index List as the one designated for next created ISDU Indexes

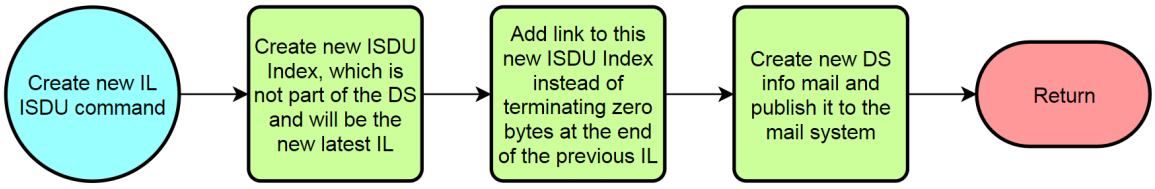


Figure 5.11: Creation of the new IL

5.3.6 Process data management

As mentioned in the ProcessData module section, the length of the output process data and the length, validity, and mode is configurable. Process data lengths are given in the DPP1 and therefore configurable using ISDU command for the DPP1 configuration mentioned

above. Both mode and validity use their own ISDU command to be set. In case of the mode, value defining which mode from mirroring, physical DI, ISDU Index will be used is contained in the command body. For the validity, command body simply contains true or false value to switch validity of input process data in the IOLTD. The mode reconfiguration procedure can be seen on following flowchart 5.12.

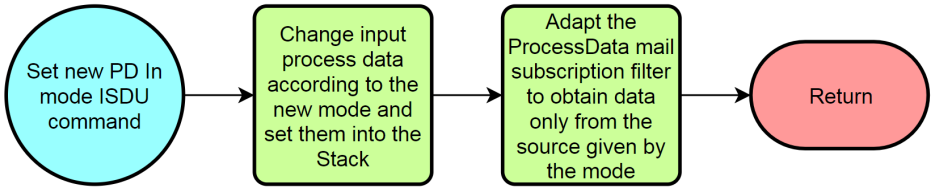


Figure 5.12: Input PD mode configuration

5.3.7 User interface

User interface in the IOLTD is supported using the on-board display. Changes on the display do not happen as s direct reaction to dedicated command. They are triggered by other changes in the IOLTD state. Particular values displayed are shown in figure 5.13 and any change of any of these values is published to the mail system by a respective module. Even when particular screen is not the one displayed, its data are taken from the mail system and kept up-to-date, so the user can get right information immediately after switching to another screen. Following figure shows the "main" screen with used IO-Link revision, screen with process data information (length, validity, mode) and screen with data storage information (DS size, IL count, DS Index count).

(a) (b) (c)

Figure 5.13: Main screen (a) PD screen (b) DS screen (c).

5.3.8 Automatic configuration test

This feature implements a possibility to process hundreds/thousands of tests automatically. Particular test cases are stored in the CSV file on the SD card and ConfigAutoTester, as a

part of the Configuration module, reads them one by one and tests, whether the Master is able to communicate with such combination of parameters. Following (briefly summarized) procedure is implemented, to make testing both fast and robust.

Additionally, implementation contains safety timeout on the operate status waiting to avoid the infinite loop. Because the Stack has one task in the FreeRTOS dedicated to its operations, automatic configurations testing has its own task, so it can wait for the operate status and 10x cycle-time, which both need the Stack task running. Following flowchart 5.14 illustrates the implementation and the task usage.

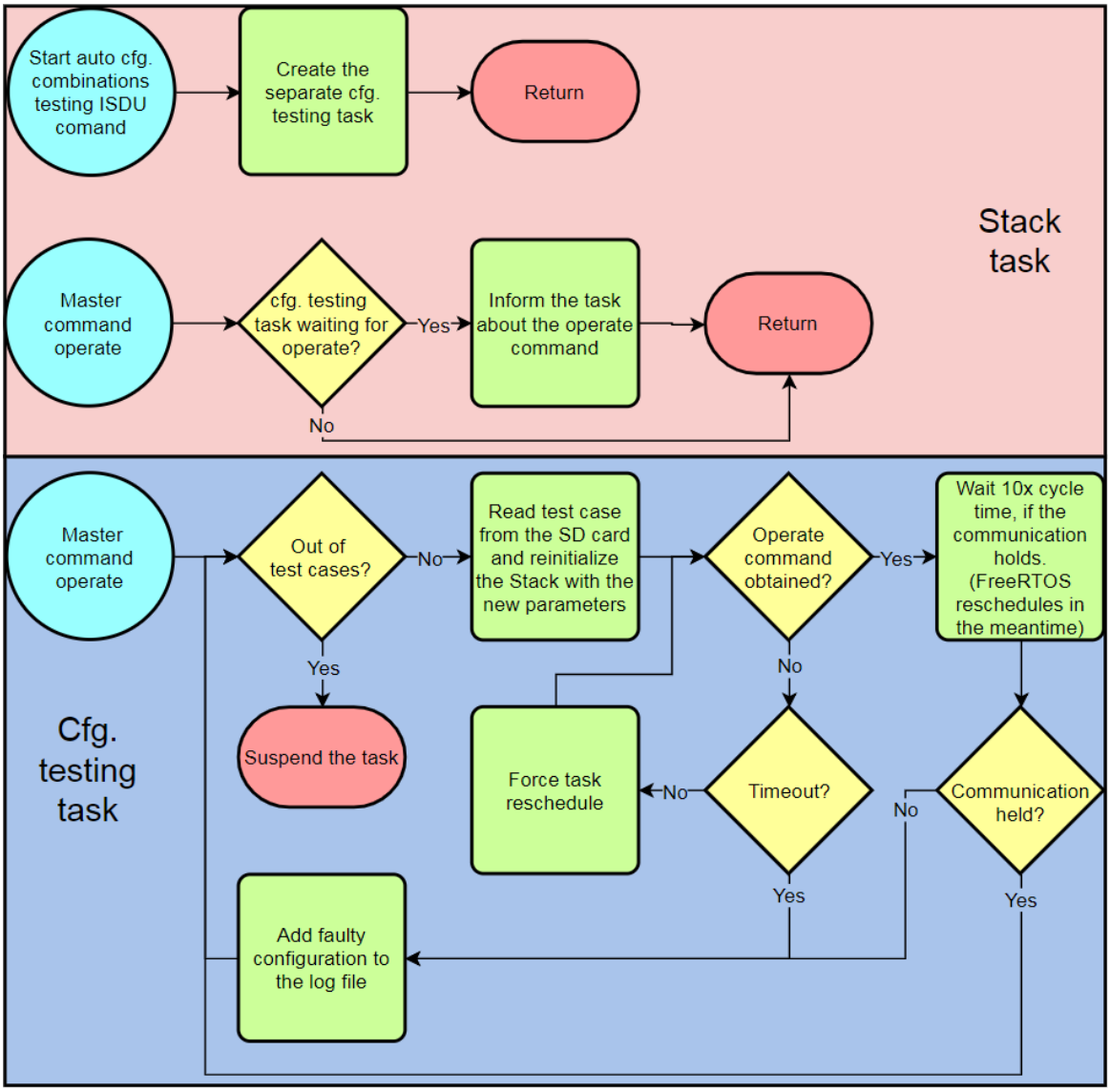


Figure 5.14: Auto cfg. testing implementation and the task usage

5.4 Proof of Concept

This section shows and describes demonstration of how the IOLTD can be used as the support for automated testing of IO-Link Masters. This demonstration uses simplified setup described in 3.5.4 as can be seen on picture 5.19. This simplified setup consists of the PC with the python script including the OPC UA client, PLC with the UPC UA server feature, IO-Link Master and the IOLTD.

Figure 5.15: Proof of concept station: PLC on the left, IO-Link Master in the middle and the IOLTD on the right

Following subsections demonstrate basic reconfiguration operations, which can be done using python script and the OPC UA to send IOL CALLs (ISDU requests), which are processed as the commands in the IOLTD. Pictures in the subsections are output console of the python script used for demonstration.

5.4.1 ISDU Index create

```
Clearing dynamic data storage.
Reading from index 64
Data read: None
iSDU index 64 read failed with IOL_STATUS=0x00008011
Creating index 64 with length 10.
Writing [0, 17, 34, 51, 68, 85, 102, 119, 136, 153] to index 64
Reading from index 64
Data read: [0, 17, 34, 51, 68, 85, 102, 119, 136, 153]
```

Figure 5.16: Console output from the ISDU Index creation demonstration

Steps of this demonstration are following.

1. **Clearing dynamic DS** - reset the dynamic ISDU state of the Device for the test purposes
2. **Reading from the ISDU Index 64** - Data read results as "None" because this Index does not yet exist, which is also the reason of the error message, where status returned for the request is 0x00008011, where 0x8011 means standard error: ISDU Index does not exist in the Device.
3. **Creating the Index 64 with length 10** - Creation of 10 bytes long Index, which is still empty.
4. **Writing to the Index 64** - Filling the ISDU Index 64 with data as shown on the picture.
5. **Reading from the Index 64** - Same command, which failed at the beginning is now able to read from the created ISDU Index.

5.4.2 DPP1 reconfiguration

```
Reading from index 0
Data read: [0, 68, 68, 45, 17, 16, 16, 0, 170, 0, 11, 187, 0, 0, 0, 0]
Writing [0, 68, 20, 45, 17, 131, 131, 1, 2, 3, 4, 5, 0, 0, 0, 0] to index 20994
Reading from index 0
Data read: [0, 98, 98, 45, 17, 131, 131, 1, 2, 3, 4, 5, 0, 0, 0, 0]
```

Figure 5.17: Console output from the DPP1 reconfiguration demonstration

Steps of this demonstration are following.

1. **Reading from the ISDU Index 0** - Index 0 stores current DPP1 configuration in the Device. Read data shows the configuration.
2. **Writing to the ISDU Index 20994** - This Index number is reserved for the DPP1 reconfiguration command in the IOLTD FW extension. Write request to this Index initiates the reconfiguration using data provided in the request.
3. **Reading from the ISDU Index 0** - Again, the read request for DPP1 as in the first step. Changed values according to the previous command can be seen and reconfigured values are: cycle time, PD lengths, VendorID, and the DeviceID.

5.4.3 Backup/restore commands

```
Clearing dynamic data storage.
Creating index 64 with length 10.
Writing [0, 17, 34, 51, 68, 85, 102, 119, 136, 153] to index 64
Reading from index 64
Data read: [0, 17, 34, 51, 68, 85, 102, 119, 136, 153]
Calling DS backup command
Clearing dynamic data storage.
Reading from index 64
iSDU index 64 read failed with IOL_STATUS=0x00008011
Data read: None
Restoring dynamic data storage.
Reading from index 0x0040
Data read: [0, 17, 34, 51, 68, 85, 102, 119, 136, 153]
```

Figure 5.18: Console output from the backup/restore demonstration

Steps of this demonstration are following.

1. **Clearing dynamic DS**
2. **Creating the Index 64 with length 10**
3. **Writing to the Index 64**
4. **Reading from the ISDU Index 64** - This Index now exists in the Device and is filled with data.

5. **Calling backup command** - The whole storage and the ISDU state was saved in the non-volatile memory.
6. **Clearing dynamic DS**
7. **Reading from the ISDU Index 64** - This Index now does not exists, since the clearing command was executed.
8. **Calling restore command**
9. **Reading from the ISDU Index 64** - This Index now exists again and is filled with data as before, since the restore command was executed and it existed in the backup.

5.4.4 Index List creation

```
Clearing dynamic data storage.
Reading from index 3
Data read: [0, 0]
Creating index 64 with length 10.
Creating index 65 with length 10.
Creating index 66 with length 10.
Creating new index list
Creating index 67 with length 10.
Reading from index 3
Data read: [0, 64, 0, 0, 65, 0, 0, 66, 0, 80, 1]
Reading from index 20481
Data read: [0, 67, 0, 0, 0]
```

Figure 5.19: Console output from the IL creation demonstration

Steps of this demonstration are following.

1. **Clearing dynamic DS**
2. **Reading from Index 3** - Reading of the IL, which is now empty since the clearing command was executed.
3. **Creating Indexes 64 - 66** - Creation of ISDU Indexes, which are placed in the default IL.
4. **Creating new IL** - The new IL is created and the link to it is appended to the previous (default) IL.

5. **Creating Index 67** - Creation of ISDU Index, which is placed in the new IL.

6. **Reading from Index 3** - Same command as before, but now the default IL is not empty and contains 3 Indexes 64 - 66 and the link to the new IL. The last two bytes 80 and 1 mean 20481. when taken as a single 16-bit value, which is the Index number of the new IL.

7. **Reading from the ISDU Index 20481** - The new IL contains one ISDU Index 67 and two zero bytes, which serve as the IL termination.

Chapter 6

Conclusion

Within the scope of this thesis, three main parts were described. These were DevOps culture integration for IO-Link Masters development, a way of using IO-Link communication features to develop a Device as a test partner for automated Master testing, and FW extension for the IO-Link Device test platform's FW design and implementation to support such automated testing. As described below, all the goals given by the assignment of this thesis were successfully achieved and described within the scope of the written part of thesis.

The DevOps culture was first described generally with principles and benefits. The proposed DevOps adoption was designed and described based on research and understanding of the culture and the agile methodologies to provide a suitable way of integration of DevOps for the Siemens IO-Link Masters development. Ways of adoption of the particular modern development methodologies were described and also particular tools and their capabilities, which could be used to fulfill the methodologies purposes were recommended. Also, a specific test setup was designed and its usage and benefits were described.

The IO-Link technology was described in detail to provide a broad overview of the communication standard and to introduce the problematics. IO-Link communication features were used to introduce a possible way of going beyond the technology specification to support full remote reconfigurability of the Device while keeping all prescribed attributes of a common IO-Link Device. Such reconfigurability is an essential prerequisite for implementing a Device capable of supporting automated Maters testing, which is necessary for adopting agile methodologies in their full range.

IO-Link Device test platform was described in its previous state and introduced as a Device, which can be extended with capabilities needed to support automated Master testing. Design of this extension was then introduced. This design was built based on feature and

coding requirements, which came up from Siemens IO-Link testers, previous implementation of the Device, and from the described way of using IO-Link communication features to support the full remote reconfigurability and the automated testing. The extension design was described in respect to previous implementation of the IOLTD. FW modules already existing in the Device and used in the design were described as they are implemented. Modules, which were already implemented, but needed an adoption were described as they were implemented and how they will be changed. New and redesigned modules were designed and described from the ground. Features and capabilities of the extension design were designed to fulfill all the requirements and their principle was described respectively to previously described modules. Also, an extension of a USB configuration interface of the IO-Link Device test platform was designed to cooperate with the FW extension to support DevOps. Specifically to be able to read an ISDU state of the Device. This extension, however, was not implemented within the scope of this thesis, because all the features needed to support DevOps were successfully implemented using remote reconfiguration via ISDU interface. Thus, this USB interface extension was agreed by the Siemens IO-Link development team to be redundant and not needed to be implemented.

Based on the design, the IOLTD FW extension to support DevOps was implemented. The implementation was done using extended embedded c++ language and the IAR compiler. The extension was built on previous implementation of the IO-Link stack, Gateway application, and the FreeRTOS system. Particular implementation of modules and features of the extension were described and illustrated using flowchart diagrams. The whole extension was designed and written using object-oriented design in an understandable and scalable way to simplify possible future adaptions of the device. Also, proof of concept section with implemented python testing script and simple test station was included within the thesis to provide complete overview, how basic test station can be built and used and how the script can use remote reconfigurability of the IOLTD for automated Master testing.

This thesis can be used as a pattern for adopting modern development methodologies in general IO-Link Masters development and as a guideline for implementation of test partner Device for Masters with the automated testing support. Extension of the USB configuration interface designed within this thesis can be implemented in the future for the IO-Link Device test platform developed in the Prague Siemens IO-Link development department. Also, the current IOLTD state with the DevOps support can be ported on minimal design of the IOLTD in the future, which already exists and offers basic HW support for general IO-Link Device.

Bibliography

[1] *Vendor ID Table* [online]. IO-Link Consortium, 2018 [cit. 2019-01-06]. Available from: https://io-link.com/share/Downloads/Vendor_ID_Table.xml

[2] *Development of a Human Machine Interface and Communication System with a future IO-Link Master Test Device*. Czech Technical University in Prague, 2016. Master-s thesis. Faculty of Electrical Engineering. Vedoucí práce Ing. Pavel Burget, Ph.D.

[3] KUHN, Richard, Raghu KACKER a Yu LEI. *COMBINATORIAL TESTING* [online]. National Institute of Standards and Technology [cit. 2019-01-06]. Available from: https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=910001

[4] *FatFs - Generic FAT Filesystem Module* [online]. ChaN, 2018 [cit. 2019-01-06]. Available from: http://elm-chan.org/fsw/ff/00index_e.html

[5] *Unified Architecture* [online]. OPC foundation [cit. 2019-01-06]. Available from: https://opcfoundation.org/about/opc-technologies/opc-ua/

[6] *JENKINS IN DATANYZE* [online]. Datanyze, 2018 [cit. 2019-01-06]. Available from: https://www.datanyze.com/market-share/ci/jenkins-market-share

[7] *Features and services with Azure DevOps Services or TFS* [online]. Microsoft, 2018 [cit. 2019-01-06]. Available from: https://docs.microsoft.com/en-us/azure/devops/user-guide/services?view=vsts&tabs=new-nav

[8] *How to choose the right DevOps tools* [online]. Sarah Zorah, 2016 [cit. 2019-01-06]. Available from: https://www.atlassian.com/blog/devops/how-to-choose-devops-tools

[9] *The FreeRTOS Kernel* [online]. [cit. 2019-01-06]. Available from: https://www.freertos.org/

[10] *IO-Link Protocol Software for Device* [online]. [cit. 2019-01-06]. Available from: https://www.tmgte.de/en/IO-Link-component/io-link-protokoll-stack-device.html

[11] *The Way of DevOps: DEVOPS PRACTICES* [online]. smartsheet.com, 2018 [cit. 2019-01-06]. Available from: https://www.smartsheet.com/devops

[12] *How to infuse customer feedback into your DevOps* [online]. Ofir Nachmani [cit. 2019-01-06]. Available from: https://techbeacon.com/infuse-customer-feedback-devops

[13] *What is DevOps: The Goals of DevOps* [online]. Andrew Powell-Morse, 2017 [cit. 2019-01-06]. Available from: https://airbrake.io/blog/what-is/devops

[14] *IO-Link — popis digitální komunikace pro senzory* [online]. Vojáček Antonín, 2010 [cit. 2019-01-06]. Available from: https://automatizace.hw.cz/iolink-popis-digitalni-komunikace-pro-senzory

[15] VOLF, Ondřej. *IO-Link Device for testing of IO-Link Masters*. Czech Technical University in Prague, 2017. Master-s thesis. Faculty of Information Technology. Supervisor Ing. Miloš Fenyk.

[16] *IODD Guideline: IO Device Description* [online]. Haid-und-Neu-Str. 7 76131 Karlsruhe Germany, 2013 [cit. 2019-01-06]. Available from: http://io-link.com/share/Downloads/Guide-IODD/IO-Device-Desc-Guideline_10022_V11.zip

[17] *IODD Specification: IO Device Description* [online]. Haid-und-Neu-Str. 7 76131 Karlsruhe Germany, 2011 [cit. 2019-01-06]. Available from: http://io-link.com/share/Downloads/Spec-IODD/IO_Device_Description_V1.1_Specification.zip

[18] *IO-Link Test Specification* [online]. Haid-und-Neu-Str. 7 76131 Karlsruhe Germany, 2014 [cit. 2019-01-06]. Available from: https://io-link.com/share/Downloads/Testspec/IOL-Test-Spec_10032_V112_Jul14.pdf

[19] *IO-Link System Description: Technology and Application* [online]. Haid-und-Neu-Str. 7 76131 Karlsruhe Germany, 2016 [cit. 2019-01-06]. Available from: https://io-link.com/share/Downloads/At-a-glance/IO-Link_Systembeschreibung_engl_2016.pdf

[20] *IO-Link Interface and System Specification: IOL-Interface-Spec_10002_V112_Jul13* [online]. Haid-und-Neu-Str. 7 76131 Karlsruhe Germany, 2013 [cit. 2019-01-06]. Available from: https://io-link.com/share/Downloads/Spec-Interface/IOL-Interface-Spec_10002_V112_Jul13.pdf

Appendix A

Content of included CD

```
ansormicMT.....................................................parent folder
├──LaTeX_sources_pictures
│   └──LaTeX_sources.zip...........archive with latex source codes and figures
└──text
    └──ansormicMT.pdf.............................the thesis text in PDF format
```