Arduino based Framework for Rapid Application Development of a Generic IO-Link interface

Victor Chavez, Jörg Wollert

Department of Mechatronics and Embedded systems
FH Aachen University of Applied Sciences
Goethestraße 1, 52064 Aachen
chavez-bermudez@fh-aachen.de
wollert@fh-aachen.de

Abstract. The implementation of IO-Link in the automation industry has increased over the years. Its main advantage is it offers a digital point-to-point plug-and-play interface for any type of device or application. This simplifies the communication between devices and increases productivity with its different features like self-parametrization and maintenance. However, its complete potential is not always used.

The aim of this paper is to create an Arduino based framework for the development of generic IO-Link devices and increase its implementation for rapid prototyping. By generating the IO device description file (IODD) from a graphical user interface, and further customizable options for the device application, the end-user can intuitively develop generic IO-Link devices. The peculiarity of this framework relies on its simplicity and abstraction which allows to implement any sensor functionality and virtually connect any type of device to an IO-Link master. This work consists of the general overview of the framework, the technical background of its development and a proof of concept which demonstrates the workflow for its implementation.

1 Introduction

The IO-Link specification has expanded the features that sensors and actuators can have. Its features include self-parametrization, maintenance and generic data structures. Its implementation has increased the available options for developers and manufacturers in the automation industry.

However, the development of these devices does not follow a unique workflow and can depend on the manufacturer. This paper proposes a solution to simplify its development and allow rapid-prototyping development. With the creation of an Arduino based framework, all the available features of an IO-Link device are accessible to any developer through an abstraction layer that simplifies its use.

2 Framework overview

The design of the framework was intended to have an easy-to-use approach, such that the end-user didn't need to understand in a detailed manner the IO-Link specification. Its main objective was to create a tool that enabled a rapid prototyping approach and focused more time on to the development of device applications.

To simplify the workflow a graphical user interface (GUI) was developed. It allows the end-user to set up the device parameters and generate the IODD file. Furthermore, it creates a header file to setup options for the firmware application. Another essential part was the hardware interface that communicates with the IO-Link master. The general overview of the interaction between all the described components is shown in Figure 1.

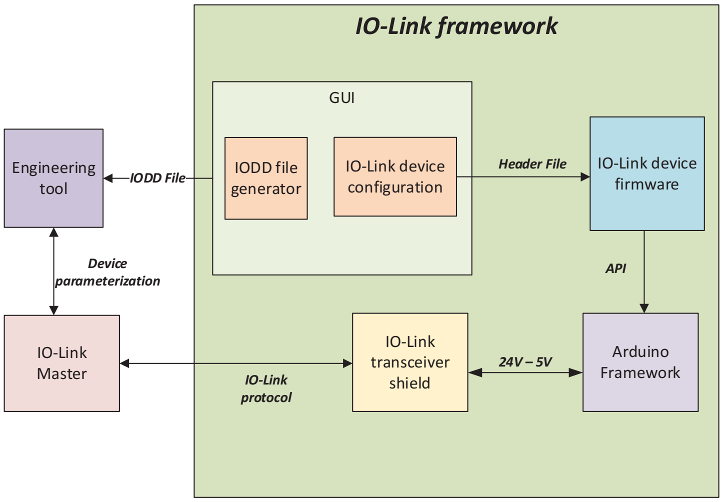


Fig. 1. General overview of framework

3 Firmware

The backbone of the system is the firmware, which runs in the background of the end-user's device application. For the selection of the embedded processor the Arduino Framework was used since it's a well-known low-cost and open-source platform that meets the requirements for its use as an IO-Link device.

The arduino framework consists of different development boards, for this particular case the Arduino Nano (atmega328p) was used due to its small factor size and enough peripherals (i.e. ADC, GPIO's, I2C/SPI). The microcontroller's peripherals used for the firmware's operation were the UART and two I/O pins to synchronize the data exchange according to the IO-Link specification.

Nevertheless, using this board brought certain limitations. The first one is the maximum baud rate supported for IO-Link. This meant that the COM2 mode transmission was used (38,4 kbit/s). In addition, the EEPROM is used to save non-volatile parameters of the device. Since the writing speed is rated to 3,3 ms per byte [1], this limits the maximum amount of writeable parameters without blocking the device application.

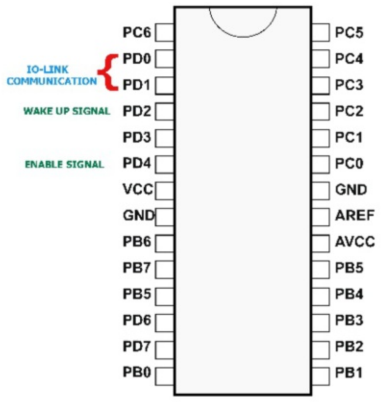
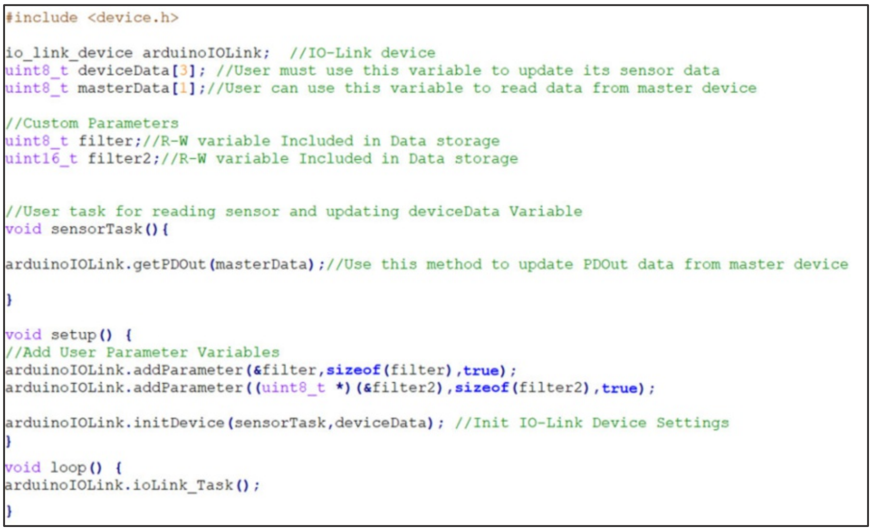


Fig. 2. Microcontroller pinout functions

The firmware is based on the current IO-Link specification described in [2]. It contains the most important features that developers would need to use the main features of the IO-Link technology, which consist of:

- Parameter Manager (PM)
- Data Storage (DS)
- Event Dispatcher (ED)
- Process Data Exchange unit (PDE)

The firmware operates by executing the respective state machines that manage the IO-Link device's features in the background. Each time a new process cycle occurs, the user's device application is called, which, can update its process data and execute any other defined instructions. The initialization of the firmware is done by passing as arguments the user's subroutine and process input data (if available), afterwards, in the main loop, the IO-Link state machines are executed (see Figure 3).



```
#include <device.h>

io_link_device arduinoIOLink;  //IO-Link device
uint8_t deviceData[3]; //User must use this variable to update its sensor data
uint8_t masterData[1];//User can use this variable to read data from master device

//Custom Parameters
uint8_t filter;//R-W variable Included in Data storage
uint16_t filter2;//R-W variable Included in Data storage


//User task for reading sensor and updating deviceData Variable
void sensorTask(){

arduinoIOLink.getPDOut(masterData);//Use this method to update PDOut data from master device

}

void setup() {
//Add User Parameter Variables
arduinoIOLink.addParameter(&filter,sizeof(filter),true);
arduinoIOLink.addParameter((uint8_t *)(&filter2),sizeof(filter2),true);

arduinoIOLink.initDevice(sensorTask,deviceData); //Init IO-Link Device Settings
}

void loop() {
arduinoIOLink.ioLink_Task();

}
```

Fig. 3. Firmware implementation in a sketch file

As aforementioned, to reduce development time, the customizable parameters of the firmware are written to a header file by means of a GUI. This header file contains the identification parameters of the devices as well indicates which IO-Link features the device will have. This implies that the final size of the firmware at compilation time will depend to a great extent on the selected features by the end-user. Table 1 shows a rough estimation of the firmware's size depending on some of these features.

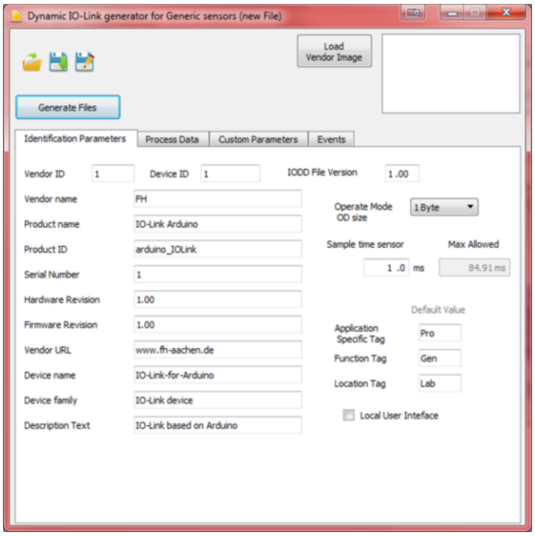
Table 1. Firmware's size estimation at compile time

Firmware Configuration	Flash Memory (KB)	SRAM (KB)
x PM x DS ✓ ED ✓ PDE	9.39	0.78
✓ PM ✓ DS ✓ ED ✓ PDE	13.30	0.98

4 GUI

The main interaction with the framework is done through the GUI. Its main functions are to generate the IODD file and the header file for the firmware. The input parameters are separated by four tabs (i.e. identification parameters, process data, custom parameters and events).

The "Identification Parameters" tab includes the obligatory parameters that an IO-Link device must have described in [2] and [3]. Furthermore, an interactive display shows the maximum allowed sampling time for the user´s device application taking in to account the number of custom parameters due to the EEPROM´s limitation and the time it takes to complete an M-Sequence message considering the on-request data size in operation mode and transmission mode (COM2).


Dynamic IO-Link generator for Generic sensors (new File)
Load Vendor Image
Generate Files
Identification Parameters
Process Data
Custom Parameters
Events

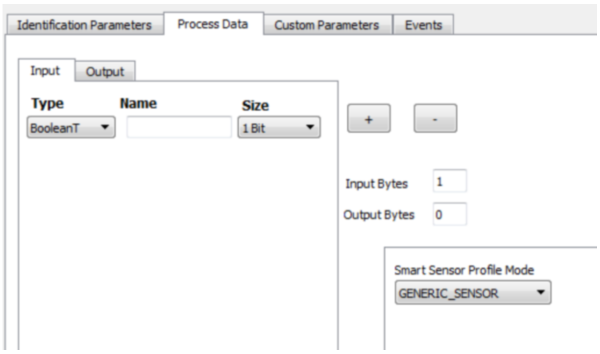
Vendor ID	1
Device ID	1
IODD File Version	1.00
Vendor name	FH
Product name	IO-Link Arduino
Product ID	arduino_IOLink
Serial Number	1
Hardware Revision	1.00
Firmware Revision	1.00
Vendor URL	www.fh-aachen.de
Device name	IO-Link-for-Arduino
Device family	IO-Link device
Description Text	IO-Link based on Arduino
Operate Mode OD size	1 Byte
Sample time sensor	1 .0 ms
Max Allowed	84.91 ms

	Default Value
Application Specific Tag	Pro
Function Tag	Gen
Location Tag	Lab

☐ Local User Inteface

Fig. 4. GUI for the framework

The "Process Data" tab manages the structure of the data that is going to be sent from the IO-Link master to the IO-Link device and vice versa. In addition, since some applications may have common functionalities, it is possible to implement some of the smart sensor profiles described in [4]. The "Custom Parameters" tab enables the user to have extra parameters, which are written and read from the EEPROM of the device. Finally, the "Events" tab is used to indicate if the device has any of the standard events described in [2] or add custom events by the user.



Type: BooleanT
Name:
Size: 1 Bit
Input Bytes: 1
Output Bytes: 0
Smart Sensor Profile Mode: GENERIC_SENSOR

Fig. 5. GUI Process data tab

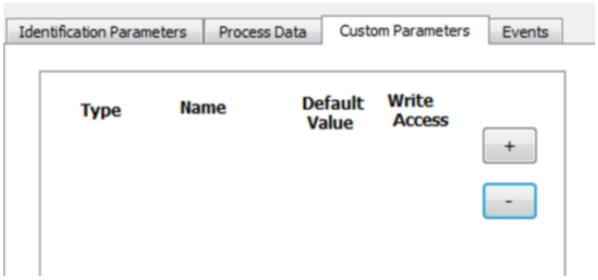
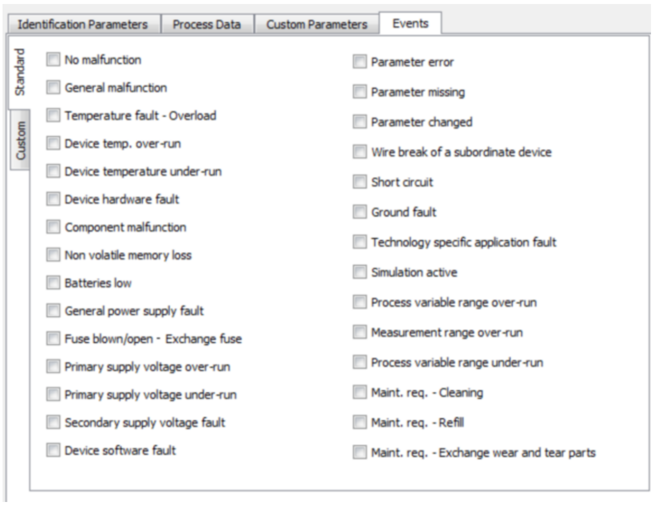


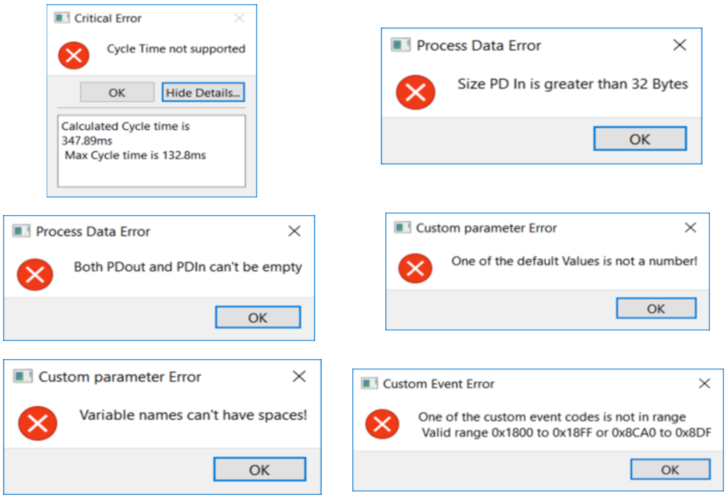
Fig. 6. GUI Custom Parameters tab



- No malfunction
- General malfunction
- Temperature fault - Overload
- Device temp. over-run
- Device temperature under-run
- Device hardware fault
- Component malfunction
- Non volatile memory loss
- Batteries low
- General power supply fault
- Fuse blown/open - Exchange fuse
- Primary supply voltage over-run
- Primary supply voltage under-run
- Secondary supply voltage fault
- Device software fault
- Parameter error
- Parameter missing
- Parameter changed
- Wire break of a subordinate device
- Short circuit
- Ground fault
- Technology specific application fault
- Simulation active
- Process variable range over-run
- Measurement range over-run
- Process variable range under-run
- Maint. req. - Cleaning
- Maint. req. - Refill
- Maint. req. - Exchange wear and tear parts

Fig. 7. GUI Events tab

The generation of the files is done by clicking a button. Considering that the files need to comply with the IO-Link specification, restrictions were implemented to avoid errors from the user. The GUI notifies the user if the files cannot be created through a notification window that describes any specific error (see Figure 8).


Critical Error
Cycle Time not supported
Calculated Cycle time is 347.89ms
Max Cycle time is 132.8ms

Process Data Error
Size PD In is greater than 32 Bytes

Process Data Error
Both PDout and PDIn can't be empty

Custom parameter Error
One of the default Values is not a number!

Custom parameter Error
Variable names can't have spaces!

Custom Event Error
One of the custom event codes is not in range
Valid range 0x1800 to 0x18FF or 0x8CA0 to 0x8DF

Fig. 8. Example of error notifications from GUI

5 Hardware interface

The hardware components of the framework consist of the Arduino Nano development board and the development of an Arduino IO-Link shield. This shield includes the IO-Link transceiver for the communication between the microcontroller and the IO-Link master. Table 2 shows a comparison of different transcievers in the market. The selection of the transceiver was based on a chip that could offer a minimal design setup and at least 200 mA of output current to power up the microcontroller and the sensor application. The MAX14827 falls short on simplicity and the L6362A does not provide enough output current. Therefore, the TIOL11-5 was the best option for this case since it has the lowest number of pins, enough output current and in addition integrated protection. The electric schematic of the IO-Link shield is shown in Figure 9.

Table 2. IO-Link transceivers comparison

Vendor	Maxim Integrated	STMicroelectronics	Texas Instruments
Transceiver	MAX14827	L6362A	TIOL111-5
Control Interface	SPI (serial programming interface)/ Digital Pin	Digital Pin	Digital Pin

Pins	24	12	10
Size	4 x 4 mm	3mm x 3mm	2.5 x 3.0 mm
Output voltage	5V and 3.3V @50mA to 250mA	5V or 3.3V @10mA	5V @50mA to 350mA
Protection functions	• Reverse polarity • Thermal protection	• Reverse polarity • Overload with cut-off function. • Thermal protection • Surge protection • GND and VCC open wire	• Reverse Polarity • EMC Protection • Surge protection • Thermal Protection

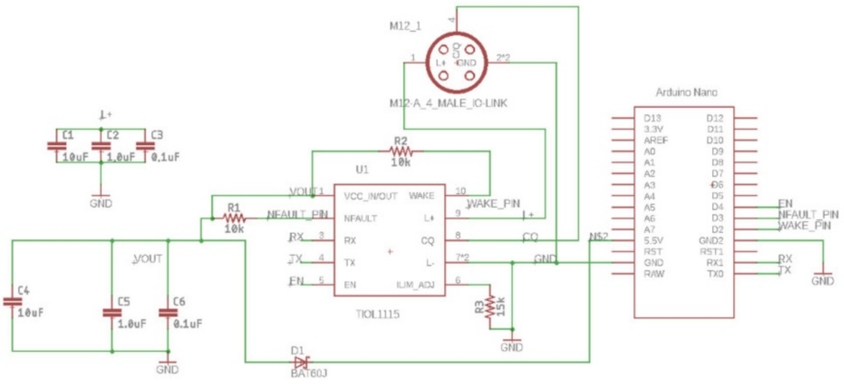


Fig. 9. IO-Link shield electric schematic

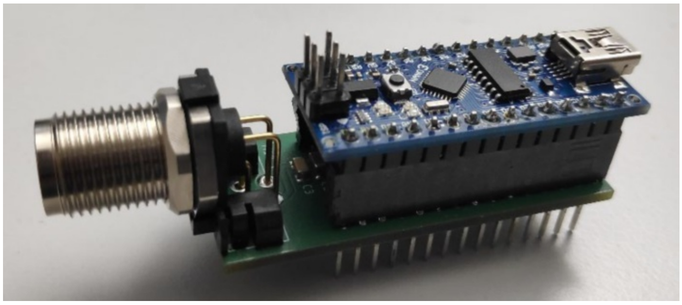


Fig. 10. IO-Link shield prototype mounted to an Arduino Nano

6 Proof of concept

Different features were tested to demonstrate the functionalities of the framework. The general workflow of the framework is shown in Figure 11. These tests were done with the following hardware:

- IO-Link shield PCB described in section 5
- WAGO 4-Channel IO-Link Master 750-657
- WAGO PLC 750-8206
- Arduino NANO development board

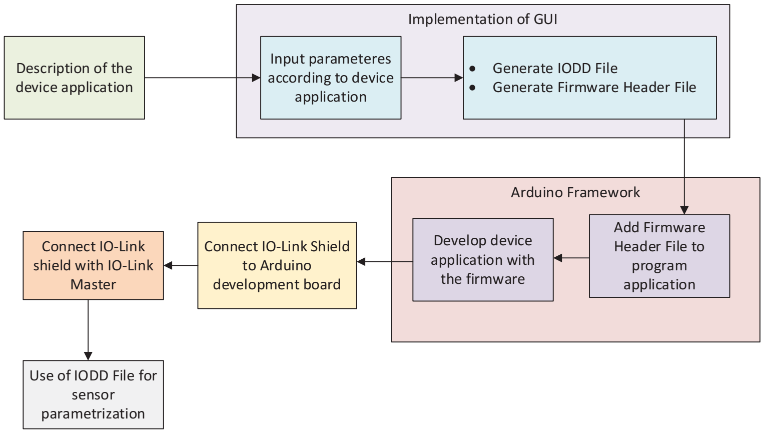

Description of the device application
Implementation of GUI
Input parameteres according to device application
Generate IODD File
Generate Firmware Header File
Arduino Framework
Add Firmware Header File to program application
Develop device application with the firmware
Connect IO-Link Shield to Arduino development board
Connect IO-Link shield with IO-Link Master
Use of IODD File for sensor parametrization

Fig. 11. General workflow of IO-Link device creation

6.1 IO-Link distance sensor

For this example, a distance sensor from Sharp model GP2Y0A21YK0F was implemented. This sensor has an analog output, which corresponds proportionally to the distance measured. Table 3 gives a full description of this device.

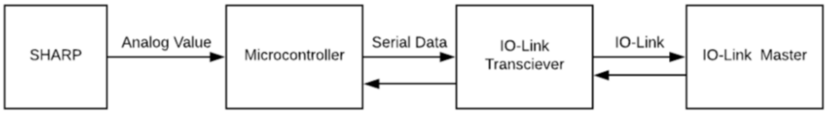
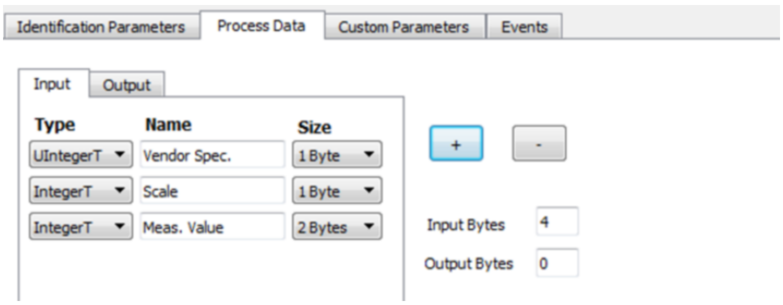


Fig. 12. Data flow of distance sensor application

Table 3. Device description

Characteristics	Description	
Process input data (4 bytes)	Vendor-specific byte (1 byte)	8-bit Value for the specific vendor application.
	Scale (1 byte)	Scale of the measurement value. i.e. Value*10^(scale)
	Measurement value (2 bytes)	16 bit value from sensor

The parameters were set to match the required sensor as seen in Figure 13. In addition, diagnostic events were set as seen in Figure 14. The main routine for the Arduino Nano consisted of reading the analog value of the sensor and converting it to the measured distance in centimeters. Due to a low range and far range limitation from the sensor, standard events from the IO-Link specification were implemented so each time a certain threshold was reached the IO-Link master is notified (see Figure 15).

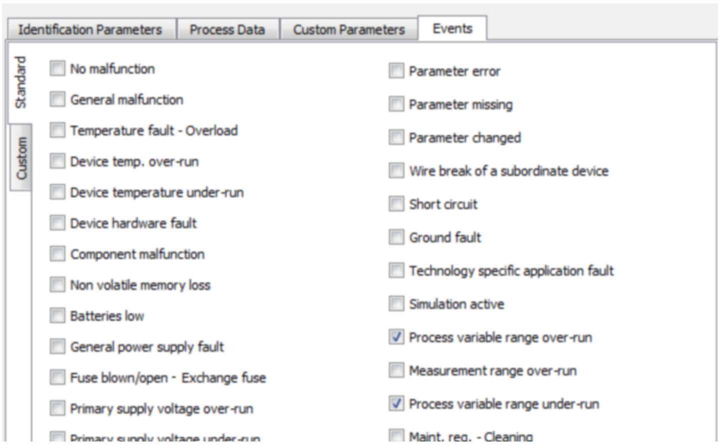


Type	Name	Size
UIntegerT	Vendor Spec.	1 Byte
IntegerT	Scale	1 Byte
IntegerT	Meas. Value	2 Bytes

Input Bytes 4

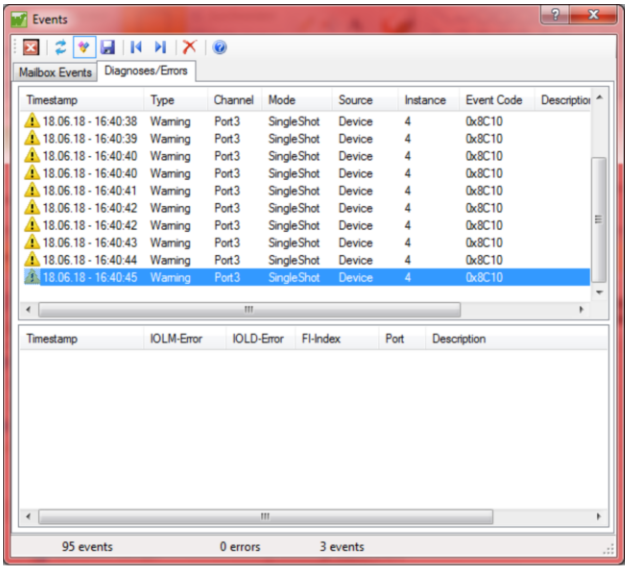
Output Bytes 0

Fig. 13. Process data parameters for the device



- [ ] No malfunction
- [ ] General malfunction
- [ ] Temperature fault - Overload
- [ ] Device temp. over-run
- [ ] Device temperature under-run
- [ ] Device hardware fault
- [ ] Component malfunction
- [ ] Non volatile memory loss
- [ ] Batteries low
- [ ] General power supply fault
- [ ] Fuse blown/open - Exchange fuse
- [ ] Primary supply voltage over-run
- [ ] Primary supply voltage under-run
- [ ] Parameter error
- [ ] Parameter missing
- [ ] Parameter changed
- [ ] Wire break of a subordinate device
- [ ] Short circuit
- [ ] Ground fault
- [ ] Technology specific application fault
- [ ] Simulation active
- [✓] Process variable range over-run
- [ ] Measurement range over-run
- [✓] Process variable range under-run
- [ ] Maint. req. - Cleaning

Fig. 14. Standard event settings for the device



Events

Mailbox Events | Diagnoses/Errors

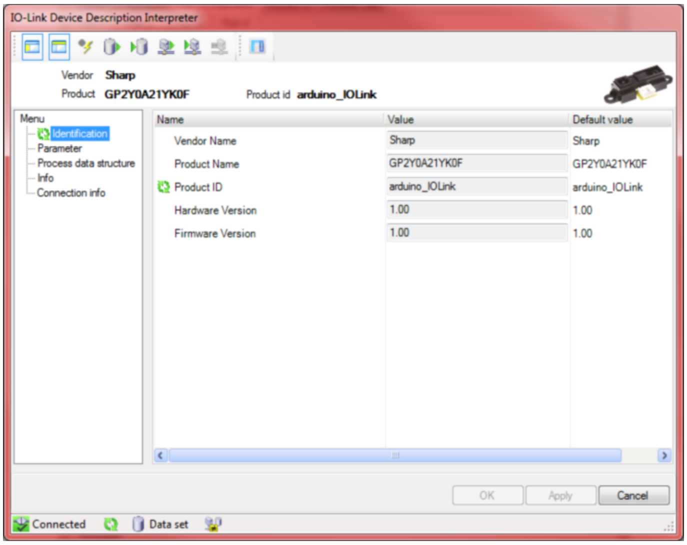
Timestamp	Type	Channel	Mode	Source	Instance	Event Code	Descriptio
18.06.18 - 16:40:38	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:39	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:40	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:40	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:41	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:42	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:42	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:43	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:44	Warning	Port3	SingleShot	Device	4	0x8C10	
18.06.18 - 16:40:45	Warning	Port3	SingleShot	Device	4	0x8C10	

Timestamp	IOLM-Error	IOLD-Error	FI-Index	Port	Description

95 events | 0 errors | 3 events

Fig. 15. IO-Link master detecting events from device

Through the IO-Link master the parameters from the IO-Link device were read (see Figure 16). Afterward, the distance values from the sensor are acquired through the PLC as seen in Figure 17. From the main program of the PLC, it can be seen that the size of the received data in the function block (i.e. "*udiRecievedbytes*") is the same as the one set in the GUI.



IO-Link Device Description Interpreter

Vendor Sharp
Product GP2Y0A21YK0F Product id arduino_IOLink

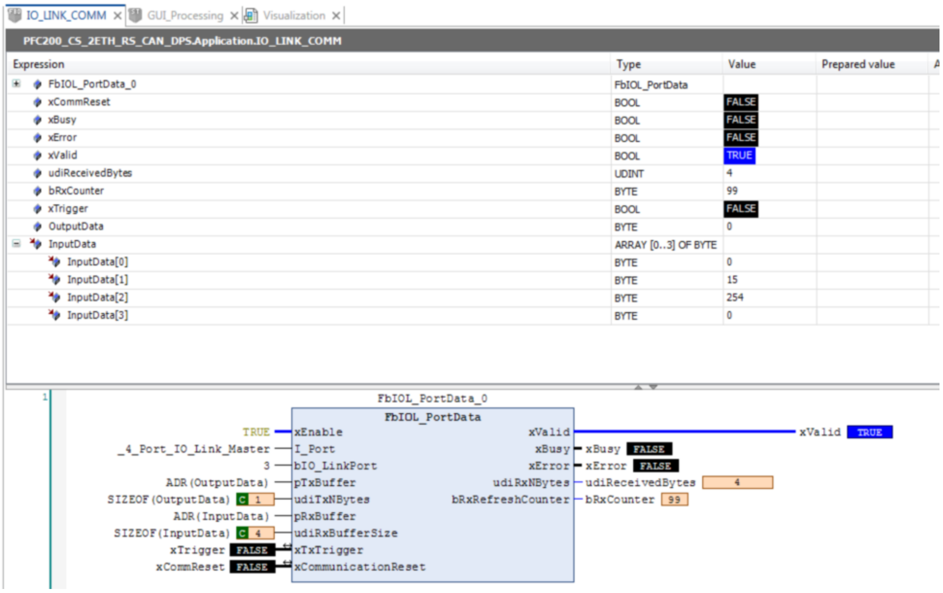
Menu
- Identification
- Parameter
- Process data structure
- Info
- Connection info

Name	Value	Default value
Vendor Name	Sharp	Sharp
Product Name	GP2Y0A21YK0F	GP2Y0A21YK0F
Product ID	arduino_IOLink	arduino_IOLink
Hardware Version	1.00	1.00
Firmware Version	1.00	1.00

OK Apply Cancel

Connected | Data set

Fig. 16. Reading IO-Link parameters of the distance sensor



PFC200_CS_2ETH_RS_CAN_DPS.Application.IO_LINK_COMM

Expression	Type	Value	Prepared value
FbIOL_PortData_0	FbIOL_PortData		
xCommReset	BOOL	FALSE	
xBusy	BOOL	FALSE	
xError	BOOL	FALSE	
xValid	BOOL	TRUE	
udiReceivedBytes	UDINT	4	
bRxCounter	BYTE	99	
xTrigger	BOOL	FALSE	
OutputData	BYTE	0	
InputData	ARRAY [0..3] OF BYTE		
InputData[0]	BYTE	0	
InputData[1]	BYTE	15	
InputData[2]	BYTE	254	
InputData[3]	BYTE	0	

FbIOL_PortData_0
FbIOL_PortData
TRUE xEnable
_4_Port_IO_Link_Master I_Port
3 bIO_LinkPort
ADR(OutputData) pTxBuffer
SIZEOF(OutputData) C 1 udiTxNBytes
ADR(InputData) pRxBuffer
SIZEOF(InputData) C 4 udiRxBufferSize
xTrigger FALSE xTxTrigger
xCommReset FALSE xCommunicationReset
xValid xValid TRUE
xBusy xBusy FALSE
xError xError FALSE
udiRxNBytes udiReceivedBytes 4
bRxRefreshCounter bRxCounter 99

Fig. 17. WAGO PLC program for the distance sensor

7 Conclusions

The results provided by the framework demonstrate a methodology for the creation of rapid prototyping IO-Link devices. The end-user only requires a general overview of the IO-Link specification and the description of its device application. Its advantage is that with just a few steps through a GUI any type of application can be created, modified or retrofitted to IO-Link. Thus, it offers a versatile approach for the development of low-cost IO-Link devices on the fly.

Furthermore, with its high-level abstraction layer it reduces the required development time. At the same time, this opens up the possibilities for developing new interfaces and applications with IO-Link. The aforementioned advantages can contribute to increase the implementation of IO-Link and its key features.

References

[1] Atmel, "ATmega328 / P," *AVR Microcontrollers*, p. 43, 2016.
[2] IO-link Community, "IO-Link Interface and System Specification v1.1.2." IO Link Community, Karlsruhe, 2013.
[3] IO-Link Consortium, "IO-Link Device Description v1.1." Karlsruhe, 2011.
[4] IO-Link Smart sensor profile group, "IO-Link Smart Sensor profile 2nd Edition." IO Link Community, Karlsruhe, 2017.