

PiXtend[®]

PIXTEND V2

kontron

Version History

| Version | Date | Description | Editor |
|---------|------------------|---|--------|
| 1.00 | 13 October 2017 | Document created | RT |
| 1.01 | 13 November 2017 | CODESYS Package Name renamed to PiXtend V2 Professional for CODESYS | RT |
| 1.02 | 20 January 2018 | Signaling for LED L1 added | RT |
| 1.03 | 26 January 2018 | Name of the SD image changed from "Linux Tools" to "Basic". Basic image contains more than just the classic PiXtend Linux tools | TG |
| 1.04 | 13 February 2018 | Information about OpenPLC added | RT |
| 1.05 | 20 February 2018 | CODESYS I/O mapping overview and device parameter description added | RT |
| 1.06 | 26 February 2018 | CODESYS Retain memory usage example created | RT |
| 1.07 | 05 March 2018 | Change Python Library V2 link to version 0.1.1, the previous version had a problem where the GPIO PullUps could not be used as intended. | RT |
| 1.08 | 28 March 2018 | Raspberry Pi 3 B+ added as a compatible model. | RT |
| 1.09 | 18 July 2018 | Information for PiXtend V2 -L- added. | RT |
| 1.10 | 31 July 2018 | - PiXtend V2 -L- Process data + Control and status bytes section added. - UG (limited liability) changed to GmbH. | TG |
| 1.11 | 09 August 2018 | OpenPLC section updated to version 3 of OpenPLC, including information on PiXtend V2 -L-. The CODESYS chapter was expanded to include the use of the PiXtend DAC device. | RT |
| 1.12 | 10 August 2018 | Correction to the text and the font and highlighting instructions for the Raspberry Pi were modified. | RT |
| 1.13 | 09 January 2019 | How to assign a static IP address to the Raspberry Pi. In chapter 7.5, the variables now use the prefix GVL, analog inputs are included in Python overview table, the SPI protocol overview was added to the Appendix for PiXtend V2 -S- and PiXtend V2 -L-. | RT |
| 1.14 | 02 December 2019 | Raspberry Pi 4 is now included, the activation of SPI communication via GPIO24 added, changed to CODESYS Version V3.5 SP15 Patch 1, note added that under Python only one instance is allowed. The following was added or changed: the installation procedure to wiringPi, changing of the serial interface from ttyS0 to ttyAMA0, the topic cyber security (IT security) added to the chapter Basic Knowledge, the PPLv2 also supporting Python 3. | RT |
| 1.15 | 08.06.2021 | Corrections to the layout, changed some URLs to the right addresses, small additions to the serial interfaces in chapter 6 for the PiXtend V2 -L-. | Tur |
| 1.16 | 21.08.2023 | Adjusted company address, CODESYS version, Internet links and source code repositories. Change the source code license for C-, Python and Node-RED from GPL v3 to MIT, except the CODESYS drivers and FHEM. Logo change. | Tur |
| 1.17 | 17.04.2024 | Adjustment to CODESYS versions, Internet addresses and source code repositories | Tur |

Table of Contents

| | | |
|---------|--|----|
| 1. | About this Documentation | 9 |
| 1.1. | Scope of Application | 9 |
| 1.2. | Copyright | 9 |
| 1.3. | Trademarks | 9 |
| 1.4. | Symbols..... | 10 |
| 2. | Important Information..... | 11 |
| 2.1. | Subject to Change | 11 |
| 2.2. | Intended Use | 11 |
| 2.3. | Technical Condition | 12 |
| 2.3.1. | PiXtend V2-S- model..... | 12 |
| 2.3.2. | PiXtend V2-L- model..... | 13 |
| 2.4. | Certifications..... | 14 |
| 2.5. | Safety information..... | 15 |
| 2.6. | Disclaimer | 16 |
| 2.7. | Contact Information | 16 |
| 2.8. | Assistance | 16 |
| 3. | Overview | 17 |
| 4. | Requirements | 17 |
| 5. | Licenses | 18 |
| 6. | Basic knowledge | 19 |
| 6.1. | Definition "Safe State" | 19 |
| 6.2. | SPI Communication, Data Transmission and Cycle Time | 19 |
| 6.3. | Activate SPI communication | 21 |
| 6.4. | Retain memory | 22 |
| 6.5. | Preparing a SD card for your Raspberry Pi | 23 |
| 6.6. | Determining the network address (IP address) of the Raspberry Pi | 25 |
| 6.7. | Raspberry Pi- Assigning a static IP address | 27 |
| 6.8. | Raspbian login information (login) | 28 |
| 6.9. | Turning off the Raspberry Pi | 29 |
| 6.10. | Compatibility of the software component..... | 30 |
| 6.11. | Serial interface..... | 30 |
| 6.12. | PiXtend V2- Notes on the serial interface..... | 31 |
| 6.13. | Information on security and cyber security..... | 33 |
| 6.13.1. | Changing the password..... | 33 |
| 6.13.2. | Raspberry Pi pin headers | 34 |
| 6.13.3. | Check commands and scripts | 34 |
| 6.13.4. | Creating system backups | 34 |
| 6.13.5. | Installing updates | 34 |
| 7. | CODESYS..... | 35 |
| 7.1. | Note | 36 |
| 7.1.1. | Copyright of texts and images..... | 36 |
| 7.1.2. | Compatibility | 36 |
| 7.2. | Requirements | 37 |
| 7.2.1. | System requirements for CODESYS | 37 |
| 7.2.2. | Required hardware | 37 |
| 7.3. | Installation of the required software components..... | 38 |
| 7.3.1. | CODESYS development environment | 38 |
| 7.3.1.1 | Introduction of CODESYS | 38 |
| 7.3.1.2 | CODESYS Download..... | 39 |
| 7.3.1.3 | CODESYS installation | 40 |
| 7.3.2. | Installing CODESYS Control for Raspberry Pi | 40 |
| 7.3.2.1 | CODESYS Control for Raspberry Pi - Download | 40 |
| 7.3.2.2 | CODESYS Control for Raspberry Pi – Installation | 41 |
| 7.3.3. | Creating a bootable SD card for the Raspberry Pi..... | 42 |
| 7.3.4. | Installing PiXtend V2 CODESYS device driver | 42 |

| | | |
|---------|--|-----|
| 7.3.4.1 | PiXtend V2 CODESYS device driver - download | 42 |
| 7.3.4.2 | PiXtend V2 CODESYS device driver– installation | 42 |
| 7.4. | Further steps | 42 |
| 7.5. | CODESYS – Creating a project..... | 43 |
| 7.5.1. | Step by step to create your first PiXtend V2 CODESYS program | 43 |
| 7.5.1.1 | Create CODESYS standard project for PiXtend V2 | 43 |
| 7.5.1.2 | Add SPI device | 45 |
| 7.5.1.3 | Add PiXtend V2 -S- device | 47 |
| 7.5.1.4 | Creating Global Variable List..... | 48 |
| 7.5.1.5 | Mapping variables | 50 |
| 7.5.1.6 | Creating the main program..... | 53 |
| 7.5.1.7 | Connection to the PiXtend V2 -S- and program download | 57 |
| 7.5.1.8 | Further steps..... | 59 |
| 7.5.2. | Step by step to create your first CODESYS Webvisu..... | 60 |
| 7.6. | Step by step to create your first DAC program | 64 |
| 7.6.1. | Create CODESYS standard project for PiXtend | 64 |
| 7.6.2. | Add SPI device..... | 66 |
| 7.6.3. | Creating Global Variable List | 69 |
| 7.6.4. | Mapping variables | 70 |
| 7.6.5. | PiXtend V2 Add DAC device..... | 72 |
| 7.6.6. | Task configuration | 74 |
| 7.6.7. | Creating the main program | 74 |
| 7.6.8. | Connection to the PiXtend V2 and program download | 75 |
| 7.6.9. | Creating the CODESYS Webvisu | 76 |
| 7.7. | CODESYS serial communication..... | 81 |
| 7.7.1. | Requirements..... | 81 |
| 7.7.2. | RS232 interface – Cable connection | 81 |
| 7.7.3. | RS485 interface – Cable connection (only PiXtend V2-L-) | 82 |
| 7.7.4. | Preparing Linux | 83 |
| 7.7.4.1 | Adjusting the interface settings..... | 83 |
| 7.7.4.2 | Activating the interface in the CODESYS configuration..... | 83 |
| 7.7.4.3 | Reboot the Raspberry Pi | 84 |
| 7.7.5. | Terminal Program..... | 85 |
| 7.7.5.1 | Installing a terminal program | 85 |
| 7.7.5.2 | Open the terminal and adjust the settings | 85 |
| 7.7.6. | CODESYS | 86 |
| 7.7.6.1 | Start the CODESYS project “PiXtendSerialTest” | 86 |
| 7.7.6.2 | Visualization | 86 |
| 7.8. | CAN communication | 88 |
| 7.8.1. | Introduction | 88 |
| 7.8.1.1 | Requirements | 89 |
| 7.8.1.2 | Restrictions | 89 |
| 7.8.2. | Hardware connection to CAN bus..... | 89 |
| 7.8.3. | Preparation and tests under Linux..... | 90 |
| 7.8.3.1 | Configuration of the Device Tree Overlay..... | 91 |
| 7.8.3.2 | Edit blacklist | 91 |
| 7.8.3.3 | Creating a CAN script | 92 |
| 7.8.3.4 | Activating CAN | 93 |
| 7.8.3.5 | Installing and using CAN utilities | 93 |
| 7.8.4. | Preparations for CODESYS | 94 |
| 7.8.4.1 | Creating the start script | 94 |
| 7.8.4.2 | Creating the baud rate script..... | 94 |
| 7.8.4.3 | Starting the CODESYS Control with PiXtend V2 -L- CAN support..... | 95 |
| 7.8.5. | CODESYS Project with CANopen | 96 |
| 7.8.5.1 | PiXtend V2 -L- as CAN slave | 97 |
| 7.8.5.2 | PiXtend V2 -L- as CANopen master | 106 |
| 7.8.5.3 | Program download and test | 110 |

| | | |
|----------|--|-----|
| 7.9. | CODESYS – PiXtend Retain memory | 113 |
| 7.9.1. | Preparation | 113 |
| 7.9.2. | Creating a program | 114 |
| 7.9.3. | Further Information | 117 |
| 7.9.3.1 | Improving data security | 117 |
| 7.9.3.2 | Working with structures | 117 |
| 7.10. | CODESYS – Shutting down the Raspberry Pi | 118 |
| 7.11. | PiXtend V2-S- Micro-controller 70 | 120 |
| 7.11.1. | SPI device parameter | 120 |
| 7.11.1.1 | 5 volt/10 volt jumper setting | 121 |
| 7.11.1.2 | GPIO Configuration..... | 122 |
| 7.11.1.3 | Micro-controller settings | 122 |
| 7.11.2. | I/O error | 123 |
| 7.12. | PiXtend V2-L- as master | 126 |
| 7.12.1. | SPI device parameter | 126 |
| 7.12.1.1 | 5-volt/10-volt jumper setting | 126 |
| 7.12.1.2 | GPIO Configuration..... | 127 |
| 7.12.1.3 | Micro-controller settings | 127 |
| 7.12.2. | I/O overview | 128 |
| 7.13. | FAQs – Frequently asked questions and troubleshooting | 132 |
| 7.13.1. | CODESYS Raspberry Pi Runtime and PiXtend V2..... | 132 |
| 7.13.2. | Serial communication | 133 |
| 7.13.2.1 | Not all characters are transmitted or arrive one after the other | 133 |
| 7.13.2.2 | Why are the GPIOs 18 and 22 switched in the program? | 133 |
| 7.13.2.3 | Scrolling the table does not work properly..... | 134 |
| 7.13.2.4 | The page/user interface is not displayed correctly | 134 |
| 7.13.2.5 | The first message is not transmitted correctly | 134 |
| 7.13.2.6 | Why is the “Auto send” function not available with RS485? | 134 |
| 7.13.2.7 | Not all baud rates are displayed | 134 |
| 8. | pxdev – Linux Tools and Library | 135 |
| 8.1. | Note | 136 |
| 8.1.1. | Usage of pixtendtool2(s/l)- Single commands for PiXtend V2 | 136 |
| 8.1.2. | Usage of pxauto2(s/l)- GUI for PiXtend V2 | 136 |
| 8.1.3. | Usage of the PiXtend/PiXtend V2 C-Library..... | 137 |
| 8.2. | Requirements | 137 |
| 8.3. | Using the PiXtend V2 basic image | 138 |
| 8.4. | Manuel installation of pxdev | 139 |
| 8.4.1. | Preparation and installation | 139 |
| 8.4.2. | Activating the SPI bus | 140 |
| 8.5. | Using pixtendtool2(s/l)..... | 141 |
| 8.5.1. | Example: | 142 |
| 8.5.2. | Further steps | 143 |
| 8.6. | Using pxauto2(s/l) | 144 |
| 8.6.1. | Navigation | 146 |
| 8.6.2. | Editing fields..... | 146 |
| 9. | C-Library “pxdev” | 148 |
| 9.1. | Requirements | 149 |
| 9.2. | Preparation | 149 |
| 9.2.1. | Creating a new folder and C file | 149 |
| 9.2.2. | Adding libraries | 150 |
| 9.3. | Programming | 150 |
| 9.4. | Example program for PiXtend V2-S- | 158 |
| 9.5. | Example program for PiXtend V2-L- | 160 |
| 9.6. | Compile and run the program | 162 |
| 9.7. | Frequently Asked Questions (FAQs) | 163 |
| 10. | FHEM..... | 164 |
| 10.1. | Requirements | 164 |
| 10.2. | Installation | 165 |

| | | |
|---------------|---|------------|
| 10.2.1. | Installation of FHEM..... | 165 |
| 10.2.2. | Integration of the module | 165 |
| 10.2.3. | Configuration of the system | 166 |
| 10.3. | Description of the module..... | 166 |
| 10.3.1. | Creating a new PiXtend V2 device..... | 166 |
| 10.3.2. | Internals..... | 166 |
| 10.3.3. | Set commands..... | 166 |
| 10.3.4. | Get commands | 168 |
| 10.3.5. | Readings | 169 |
| 10.3.6. | Attribute | 171 |
| 10.4. | Example: | 171 |
| 10.4.1. | Display of sensor values..... | 172 |
| 10.4.2. | Debouncing and combining of inputs and outputs..... | 174 |
| 10.4.3. | Controlling servo motors | 174 |
| 10.5. | Frequently Asked Questions (FAQs) | 175 |
| 11. | PiXtend Python Library V2 (PPLV2) | 176 |
| 11.1. | Requirements | 176 |
| 11.2. | Installation with the PiXtend V2 image | 176 |
| 11.3. | Installation on the original Raspbian | 177 |
| 11.3.1. | Preparation | 177 |
| 11.3.2. | PiXtend Python Library V2 Installation | 178 |
| 11.4. | Programming | 178 |
| 11.4.1. | Example directory | 178 |
| 11.4.2. | Executing an example | 178 |
| 11.4.3. | Creating your own program..... | 180 |
| 11.4.4. | Short Overview | 181 |
| 11.4.5. | Starting Python automatically | 182 |
| 11.4.6. | Using the serial interface | 184 |
| 11.4.7. | Frequently Asked Questions (FAQs) | 184 |
| 12. | Appendix | 186 |
| 12.1. | PiXtend V2-S-..... | 187 |
| 12.1.1. | Process data | 187 |
| 12.1.1.1 | Overview of the process data | 187 |
| 12.1.1.1.1 | Process data transmitted from the Raspberry Pi to the PiXtend | 187 |
| 12.1.1.1.2 | Process data transmitted from the Raspberry Pi to the PiXtend DAC | 187 |
| 12.1.1.1.3 | Process data transferred from the PiXtend to the Raspberry Pi | 187 |
| 12.1.1.2 | SPI bus protocol overview..... | 188 |
| 12.1.1.3 | SPI bus configuration..... | 189 |
| 12.1.1.4 | Output data | 190 |
| 12.1.1.4.1 | DigitalOut | 190 |
| 12.1.1.4.2 | RelayOut..... | 190 |
| 12.1.1.4.3 | GPIOWrite | 191 |
| 12.1.1.4.4 | PWM0X (L/H) – 16-bit resolution | 191 |
| 12.1.1.4.5 | PWM0Ctrl – for 16-bit PWMs | 191 |
| 12.1.1.4.5.1. | Frequency mode | 192 |
| 12.1.1.4.5.2. | Servo mode | 193 |
| 12.1.1.4.5.3. | Duty cycle mode | 194 |
| 12.1.1.4.5.4. | Universal mode | 195 |
| 12.1.1.4.6 | PWM1X (L/H) – 8-bit resolution | 195 |
| 12.1.1.4.6.1. | Servo mode | 196 |
| 12.1.1.4.6.2. | Duty cycle mode | 197 |
| 12.1.1.4.6.3. | Universal mode | 198 |
| 12.1.1.4.6.4. | Frequency mode | 199 |
| 12.1.1.4.7 | RetainDataOutX | 200 |
| 12.1.1.5 | Output data – DAC | 200 |
| 12.1.1.5.1 | AnalogOutX (L/H) | 200 |
| 12.1.1.6 | Input data..... | 202 |

| | | |
|---------------|--|-----|
| 12.1.1.6.1 | DigitalIn..... | 202 |
| 12.1.1.6.2 | AnalogInX (L/H) | 203 |
| 12.1.1.6.3 | GPIOIn | 204 |
| 12.1.1.6.4 | TempX (L/H) | 205 |
| 12.1.1.6.5 | HumidX (L/H)..... | 206 |
| 12.1.1.6.6 | RetainDataInX | 206 |
| 12.1.2. | Control and status bytes | 207 |
| 12.1.2.1 | Overview of the control bytes | 207 |
| 12.1.2.2 | Overview of the status bytes | 208 |
| 12.1.2.3 | Description of the control bytes | 209 |
| 12.1.2.3.1 | ModelOut | 209 |
| 12.1.2.3.2 | UCMode..... | 209 |
| 12.1.2.3.3 | UCCtrl..... | 210 |
| 12.1.2.3.3.1. | UCCtrl0 | 210 |
| 12.1.2.3.3.2. | UCCtrl1 | 212 |
| 12.1.2.3.4 | DigitalInDebounce | 212 |
| 12.1.2.3.5 | GPIOCtrl..... | 215 |
| 12.1.2.3.6 | GPIODebounce | 216 |
| 12.1.2.3.7 | PWM0Ctrl – for 16-bit PWMs..... | 216 |
| 12.1.2.3.7.1. | PWM0Ctrl0 | 217 |
| 12.1.2.3.7.2. | PWM0Ctrl1 (L/H) | 219 |
| 12.1.2.3.8 | PWM1Ctrl – for 8-bit PWMs | 220 |
| 12.1.2.3.8.1. | PWM1Ctrl0..... | 220 |
| 12.1.2.3.8.2. | PWM1Ctrl1 (L/H) | 222 |
| 12.1.2.4 | Description of the status bytes | 224 |
| 12.1.2.4.1 | Firmware | 224 |
| 12.1.2.4.2 | Hardware..... | 224 |
| 12.1.2.4.3 | ModelIn | 224 |
| 12.1.2.4.4 | UCState..... | 225 |
| 12.1.2.4.5 | UCWarnings..... | 226 |
| 12.2. | PiXtend V2-L- | 227 |
| 12.2.1. | Process data | 227 |
| 12.2.1.1 | Overview of the process data | 227 |
| 12.2.1.1.1 | Process data transmitted from the Raspberry Pi to the PiXtend | 227 |
| 12.2.1.1.2 | Process data transmitted from the Raspberry Pi to the PiXtend DAC..... | 227 |
| 12.2.1.1.3 | Process data transferred from the PiXtend to the Raspberry Pi..... | 227 |
| 12.2.1.2 | SPI bus protocol overview | 228 |
| 12.2.1.3 | SPI bus configuration..... | 230 |
| 12.2.1.4 | Output data | 231 |
| 12.2.1.4.1 | DigitalOutX | 231 |
| 12.2.1.4.2 | RelayOut | 232 |
| 12.2.1.4.3 | GPIOOut | 233 |
| 12.2.1.4.4 | PWMYX – 16-bit resolution | 233 |
| 12.2.1.4.4.1. | Servo mode | 234 |
| 12.2.1.4.4.2. | Duty cycle mode | 235 |
| 12.2.1.4.4.3. | Universal mode | 237 |
| 12.2.1.4.4.4. | Frequency mode | 238 |
| 12.2.1.4.5 | RetainDataOutX | 239 |
| 12.2.1.5 | Output data – DAC | 240 |
| 12.2.1.5.1 | AnalogOutX (L/H) | 240 |
| 12.2.1.6 | Input data..... | 242 |
| 12.2.1.6.1 | DigitalInX | 242 |
| 12.2.1.6.2 | AnalogInX (L/H) | 243 |
| 12.2.1.6.3 | GPIOIn..... | 244 |
| 12.2.1.6.4 | TempX (L/H) | 245 |

| | | |
|---------------|--|-----|
| 12.2.1.6.5 | HumidX (L/H) | 246 |
| 12.2.1.6.6 | RetainDataInX | 247 |
| 12.2.2. | Control and status bytes | 247 |
| 12.2.2.1 | Overview of the control bytes | 248 |
| 12.2.2.2 | Overview of the status bytes | 248 |
| 12.2.2.3 | Description of the control bytes | 249 |
| 12.2.2.3.1 | ModelOut | 249 |
| 12.2.2.3.2 | UCMode | 249 |
| 12.2.2.3.2.1. | UCCtrl | 250 |
| 12.2.2.3.2.2. | UCCtrl0 | 250 |
| 12.2.2.3.2.3. | UCCtrl1 | 252 |
| 12.2.2.3.3 | DigitalInDebounce | 253 |
| 12.2.2.3.4 | GPIOCtrl | 255 |
| 12.2.2.3.5 | GPiodebounce | 256 |
| 12.2.2.3.6 | PWMYCtrl – for 16-bit PWMs | 257 |
| 12.2.2.3.6.1. | PWMYCtrl0 | 257 |
| 12.2.2.3.6.2. | PWMYCtrl1 (L/H) | 259 |
| 12.2.2.4 | Description of the status bytes | 260 |
| 12.2.2.4.1 | Firmware | 260 |
| 12.2.2.4.2 | Hardware | 260 |
| 12.2.2.4.3 | ModelIn | 260 |
| 12.2.2.4.4 | UCState | 261 |
| 12.2.2.4.5 | UCWarnings | 262 |
| 12.3. | Error LED “L1” Signals | 263 |
| 13. | List of Figures | 264 |
| 14. | List of Tables | 266 |

About this Documentation

Keep this documentation in a safe place for future reference!

This documentation is part of the product and is to be kept for the entire duration of the product's usage. If the product is passed on or sold, this document must be handed over to the next user; this also includes any updates and/or changes to this documentation.

1.1. Scope of Application

This documentation applies exclusively to the software components listed in the table of contents and to the following PiXtend V2 models:

- PiXtend V2 -S- Extension Board (Article: 50199 004)
- PiXtend V2 -S- ePLC Basic (Article: 50199 005 + 50199 013)
- PiXtend V2 -S- ePLC Pro (Article: 50199 006 + 50199 014)
- PiXtend V2 -S- ePLC Basic Pi4 (Article: 50199 02005 + 50199 02113)
- PiXtend V2 -S- ePLC Pro Pi4 (Article: 50199 02406 + 50199 02514)
- PiXtend V2 -L- Extension Board (Article: 50199 001)
- PiXtend V2 -L- ePLC Basic (Article: 50199 002 + 50199 011)
- PiXtend V2 -L- ePLC Pro (Article: 50199 003 + 50199 012)
- PiXtend V2 -L- ePLC Basic Pi4 (Article: 50199 018 + 50199 019)
- PiXtend V2 -L- ePLC Pro Pi4 (Article: 50199 022 + 50199 023)

Note:

If only the name PiXtend V2 is used in this documentation, the information mentioned applies equally to both products (PiXtend V2 -S- and PiXtend V2 -L-).

1.2. Copyright

This documentation, including all texts and pictures, is protected by copyright. The written approval of Kontron Electronics GmbH, D-72636 Frickenhausen, Germany, must be obtained for any other use, translation into other languages, archiving or other alteration.

Copyright 2024 © Kontron Electronics GmbH

1.3. Trademarks

- "Raspberry Pi" and its logo are registered trademarks of the Raspberry Pi Foundation - www.raspberrypi.org.
- "CODESYS" and its logo are registered trademarks of the 3S-Smart Software GmbH - www.codesys.com.
- "PiXtend", "ePLC" and its logo are registered trademarks of the Kontron Electronics GmbH – www.kontron-electronics.de.
- "AVR", "ATmega" and its logo are registered trademarks of the www.atmel.com Microchip Technology Corporation www.microchip.com.
- "Debian" and "Raspbian" are registered trademarks of the Debian Project – www.debian.org.
- "I2C" and "I²C" are registered trademarks of NXP Semiconductors – www.nxp.com.
- "Arduino" is a registered trademark of the Arduino AG – www.arduino.cc.

The rights of all companies and company names mentioned herein as well as products and product names lie with the respective companies.

1.4. Symbols

NOTICE

NOTICE indicates a particular characteristic.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, will result in death or serious injury.

Important Information

This chapter contains information on the reservation of the right to make changes, the intended use, the technical condition of the PiXtend V2 models, the exclusion of liability, the safety instructions and where you can obtain assistance.

2.1. Subject to Change

Kontron Electronics GmbH reserves the right to revise or amend this documentation in whole or in part if this serves the technical progress or if existing software components are changed or new ones have been created. The latest version of this documentation is always available at <https://www.pixtend.de>.

2.2. Intended Use

PiXtend V2 The devices are used in combination with the single-board computer "Raspberry Pi" (Raspberry Pi Foundation, UK registered charity 1129409), which is already included in the product (PiXtend V2 ePLC Basic/Pro, Basic Pi4/Pro Pi4) or has been bought by the customer as an accessory (PiXtend V2 Extension Board).

The PiXtend V2 system fulfills the function of a programmable logic controller (PLC) or an electrical measuring, control, regulating and laboratory device. It can read and evaluate sensors and control actuators. The logic programming of inputs and outputs can be done, among other things, with the software "CODESYS V3" from the company 3S Smart Software Solutions GmbH. Kontron Electronics also provides components for other programming languages and systems from the areas of IT and home automation that customers can use. Instructions and examples have been created by Kontron Electronics for this purpose.

PiXtend V2 devices are designed for dry indoor environments - protection classes IP20 (ePLC Pro) and IP00 (Extension Board and ePLC Basic). Operation outdoors and in humid/wet rooms is not permitted, except when PiXtend devices are installed in a suitable housing. The devices are not designed for hazardous areas or safety critical systems or installations.

PiXtend V2 devices can be used in industrial and commercial environments, in educational facilities and in residential areas.

PiXtend V2 offers the possibility, under certain conditions, of switching dangerous voltages. Working on dangerous voltages is only permitted for qualified personnel (the requirements may differ from country to country). If in doubt, the use of dangerous voltages is prohibited.

Apart from this, PiXtend is suitable for all persons aged 14 and over who have read and understood the safety data sheet and the manuals. Use in educational facilities must be supervised by qualified and authorized personnel. Power supplies and accessories used must be approved for the country in which the PiXtend V2 system is to be installed and used.

2.3. Technical Condition

2.3.1. PiXtend V2 -S- model

PiXtend V2 -S- is supplied with a pre-defined configuration, independent of its model:

- “SPI_EN” switch activated → Communication between PiXtend and Raspberry Pi is activated.
- “PI_5V” switch activated → PiXtend and Raspberry Pi are supplied by the same voltage regulator on the PiXtend (input voltage 24V DC $\pm 20\%$). No separate power supply needs to be connected to the Raspberry Pi.
- All digital inputs are configured for 24V (no jumper is set).
- All analog inputs are configured for 0 to 10V (no jumper is set).
- The micro-controller firmware is always the latest version released by Kontron Electronics. The current version can be found on our website.

The ePLC models include additional configurations:

ePLC Basic, Pro, ePLC Basic Pi4 & Pro Pi4

- Contents of the SD card
 - The contents are stated in the order, e.g., “CODESYS Control Demo” or “PiXtend Basic” (C/Python / Node-RED).
 - Our dealers always receive and ship the SD card “CODESYS Control Demo.”

ePLC Pro & ePLC Pro Pi4

- Housing → stainless-steel cover and DIN rail housing pre-assembled.

If you need another version or a different hardware and software combination, please send your request directly to us (info@pixtend.de).

2.3.2. PiXtend V2 -L- model

PiXtend V2 -L- is supplied with a pre-defined configuration, independent of its model:

- “SPI_EN” switch activated → Communication between PiXtend and Raspberry Pi is activated.
- “PI_5V” switch activated → PiXtend and Raspberry Pi are supplied by the same voltage regulator on the PiXtend (input voltage 24V DC $\pm 20\%$). No separate power supply needs to be connected to the Raspberry Pi.
- All digital inputs are configured for 24V (no jumper is set).
- All analog inputs are configured for 0 to 10V (no jumper is set).
- “CAN” / “AO” Jumper is set to “AO” → analog outputs active, CAN inactive.
- RS485 “A” / “M” Jumper is set to “A” → automatic send/receive switching.
- Termination resistors for RS485 and CAN are not set (no jumper).
- The micro-controller firmware is always the latest version released by Kontron Electronics. The current version can be found on our website.

The ePLC models include additional configurations:

ePLC Basic & Pro, ePLC Basic Pi4 & Pro Pi4

- Contents of the SD card
 - The contents are stated in the order, e.g., “CODESYS Control Demo” or “PiXtend Basic” (C / Python / Node-RED).
 - Our dealers always receive and ship the SD card “CODESYS Control Demo.”

ePLC Pro & ePLC Pro Pi4

- Housing → stainless-steel cover and DIN rail housing pre-assembled.

If you need another version or a different hardware and software combination, please send your request directly to us (info@pixtend.de).

2.4. Certifications



This product has been designed and manufactured in accordance with applicable European directives and is therefore marked with the CE symbol. The intended use is described in this document. A safety data sheet is included with each product in paper form (multilingual).

Warning:

Changes and modifications to the product, as well as a non-compliance with the information contained in the manuals and safety data sheets, will lead to the loss of certification.



The symbol of the crossed-out waste bin (WEEE symbol) means that this product must be recycled separately from any household waste as electrical waste. Ask your local municipal administration to find the nearest recycling station.

2.5. Safety information

⚠ CAUTION

PiXtend V2 devices are not designed for safety critical systems. Before use, check the suitability of Raspberry Pi and PiXtend V2 for your application. The default settings have been selected to meet the requirements of most applications.

⚠ DANGER

Caution is required when handling and, especially, when experimenting with the process data. The connected sensors and actuators may assume undefined states or output incorrect values if handled incorrectly.

If a machine, a device or a process is controlled or regulated by the PiXtend V2, dangerous conditions can occur. Make yourself aware of possible sources of danger before starting.

If in doubt, disconnect the connections to the devices, sensors, motors, etc. from the power supply in order to minimize dangers to people and the machine.

The control bytes are not saved permanently. After a reset or power cycle, all previous settings are erased and the next action is only executed by the PiXtend V2 again when the onboard micro-controller receives a command.

NOTICE

It is recommended not to start the PiXtend V2 automatically during the development of a control program.

2.6. Disclaimer

The information contained in this documentation has been compiled, checked and tested with the greatest possible care with the software and hardware described herein. Nevertheless, discrepancies cannot be ruled out completely. Kontron Electronics GmbH is not liable for any damages that may result from the use of the software, software components, hardware or the steps described in this documentation.

2.7. Contact Information

Our postal address:

Kontron Electronics GmbH

Max-Planck-Str. 6
D-72636 Frickenhausen, Germany

How to reach us:

Telephone: +49 7022 4057-0
info@kontron-electronics.de
www.kontron-electronics.de

2.8. Assistance

If you have any questions, please feel free to contact us by e-mail (support@pixtend.de). You will receive an answer as soon as possible.

The latest versions of all documents and software components can be found in the download section of our website
<https://www.pixtend.de>.

Overview

PiXtend is a programmable logic controller (PLC) used in combination with the Raspberry Pi single-board computer. The wide range of digital and analog inputs and outputs makes it possible to connect a wide variety of sensors and actuators. Other devices, controls and computer systems can be connected via standard serial interfaces (RS232/RS485, Ethernet, WiFi, CAN). All interfaces and I/Os have a robust design and comply with the PLC standard (IEC 61131-2).

Requirements

The exact hardware and software requirements are listed separately at the beginning of each chapter, as these may vary depending on the software component used.

In general, the following is required for most applications:

- System requirements for development computers (PC)
 - Microsoft Windows 7/8/10/11 (32/64 bit)
 - Suitable PC hardware for the corresponding Microsoft Windows platform
 - Sufficient disk space for installing new programs
 - The necessary rights (e.g., administrator rights) that allow installation of a new program
- Required hardware
 - PiXtend V2 Board (www.pixtend.de)
 - Raspberry Pi Model B+/2 B/3 B/3 B+/4 B
 - Standard RJ45 network cable
- For initial installation of a new Raspberry Pi:
 - SD card reader, internal or external
 - SD memory card, minimum 4 GB (recommended 8 GB)
micro SD for Raspberry Pi model B+/2 B/3 B/3 B+/4 B
 - Optional: HDMI-compatible monitor
 - Optional: USB keyboard
 - Optional: USB mouse

Licenses

Linux Tools:

All examples and programs for using the PiXtend V2 system were published by Kontron Electronics GmbH under the MIT license and can be used and modified freely. The use in one's own projects as well as for commercial use, in connection with PiXtend products, is permitted.

Our Linux programs and drivers are based on the WiringPi library by Gordon Henderson. Please note the respective license terms.

CODESYS:

The "PiXtend V2 Professional for CODESYS Package" for CODESYS V3.5 is provided free of charge by Kontron Electronics GmbH and can be used for private and commercial purposes. The package can be freely distributed and used on any number of computers.

It is not permitted to modify the "PiXtend V2 Professional for CODESYS Package" or any of the PiXtend V2 driver libraries. If you need a customized version of one of the PiXtend V2 drivers, please contact Kontron Electronics GmbH.

The CODESYS examples and the RC-Plug library, which are also included in the package, were published under the MIT license and can be used and modified freely. The use in one's own projects as well as for commercial use is permitted.

Our CODESYS programs, devices and libraries are based on the "CODESYS Control for Raspberry Pi" and other software components from CODESYS GmbH. The corresponding license conditions of CODESYS GmbH and CODESYS apply.

FHEM:

The source code provided for the FHEM system has been published by Kontron Electronics GmbH under the GNU GPL v3 license. Our program and driver uses the WiringPi library from Gordon Henderson and FHEM modules from Rudolf König. The license terms of the respective rights holder apply.

Basic knowledge

6.1. Definition "Safe State"

The PiXtend V2 system has a user-enabled watchdog timer. If the user has activated it and there is communication failure or the SPI bus is interrupted, the Watchdog triggers after the user-defined waiting time. When the Watchdog is triggered, the micro-controller stops its task and switches to a defined state. This state is referred to as the "safe state" and has the following characteristics:

- All digital outputs and relays are switched off, put into the idle state
- PWM outputs are switched to high impedance (tri-state)
- Retain data is stored when Retain option has been activated
- The status LED "L1" flashes depending on the cause of the error (Error LED "L1" Signals) when the LED is activated
- The micro-controller or the PiXtend V2 device has to be restarted (power cycle)

If switching off the digital outputs does not correspond to the safe state of your application, external measures must be taken to prevent any critical or dangerous situations.

6.2. SPI Communication, Data Transmission and Cycle Time

Communication between the Raspberry Pi and the PiXtend V2 (micro-controller) is carried out by using the SPI interface and with a standard transmission speed of 700 kHz. Just taking the time required for the data transmission alone into consideration is not sufficient to be able to determine how fast the communication with the micro-controller is. The cycle time also includes how long it takes for the next transmission as well as further transmissions after the first SPI data transmission.

After each data transmission the micro-controller needs some time to process the data received. Only then may the next data transmission take place; otherwise, it is possible that the micro-controller will not accept the new data.

The following cycle times apply to the PiXtend V2 -S- device:

| Name | Time | Comment |
|-----------|--------|---|
| Minimal | 2.5 ms | Fastest possible cycle time may not be exceeded. No DHT11/22 sensors can be used. |
| Optimized | 10 ms | Recommended cycle time when no DHT11/22 sensors are used. |
| Standard | 30 ms | Recommended cycle time when all functions are used. |

The following cycle times apply to the PiXtend V2 -L- device:

| Name | Time | Comment |
|-----------|-------|---|
| Minimal | 5 ms | Fastest possible cycle time may not be exceeded. No DHT11/22 sensors can be used. |
| Optimized | 10 ms | Recommended cycle time when no DHT11/22 sensors are used. |
| Standard | 30 ms | Recommended cycle time when all functions are used. |

The default cycle time has been preset in all software components so that all functions are available to the user without restrictions.

The cycle time may be changed by the user, but the steps necessary for such a change depend on the particular software component used.

The Linux tools for PiXtend V2, for example, must be changed in the source code, as well as the PiXtend Python Library V2.

All CODESYS users, however, can simply change the interval of the corresponding task and the changes are applied immediately after the next login.

NOTICE

The CODESYS driver of the PiXtend V2 has a protective mechanism which prevents the driver from communicating with the micro-controller too fast.

If the Cycle Guard detects a bus cycle interval that is too short, a “BusCycleError” is shown in the PiXtend V2 IO mapping, and no communication with the micro-controller takes place. In this case, select a longer (slower) interval for the bus cycle task.

6.3. Activate SPI communication

The SPI communication between the Raspberry Pi and the PiXtend V2 board requires a release (or enable) signal from the GPIO24; otherwise, no communication can take place. This device was installed so that the user can decide if and when a data exchange between the Raspberry Pi and PiXtend V2 is allowed to take place.

The GPIO24 has other names for the Raspberry Pi, which are used depending on the software component used. The following table shows the names used in the WiringPi software.

| Physical pin | wiringPi name | wiringPi number | BCM connection name |
|--------------|---------------|-----------------|---------------------|
| 18 | GPIO. 5 | 5 | 24 (GPIO24) |

Table 1: Raspberry Pi GPIO24 names

The example in the languages C and Python, as well as the tool CODESYS, are shown below on how to use the GPIO24.

Example with C:

- `wiringPiSetup();`
- `pinMode(5, OUTPUT);`
- `digitalWrite(5, 1);`

Note:

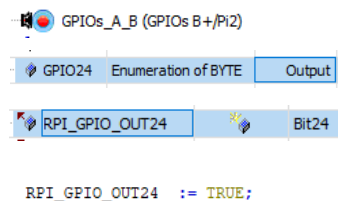
For the programming languages C and Python, libraries are available to the user which already utilize this function. It is recommended to use these libraries when ever possible. Further information can be found in the corresponding chapters of this manual.

Example Python:

- `GPIO.setmode(GPIO.BCM)`
- `GPIO.setup(24, GPIO.OUT)`
- `GPIO.output(24, True)`

Example CODESYS V3.5:

- Open the GPIO device
- GPIO Parameter, set GPIO24 as output
- GPIO I/O mapping, add a variable name for bit 24 in channel digital outputs
- Set the variable in program to TRUE



6.4. Retain memory

Retain Memory is a special memory on the PiXtend V2 that allows the user to maintain values of variables in the event of a power failure and to resume a task after a restart. A technology developed by Kontron Electronics allows the Raspberry Pi to utilize this special memory through the PiXtend V2. This storage technology is a well-known function in automation technology; it is available to every PiXtend V2 application, regardless of which software component is used.

The PiXtend micro-controller monitors the voltage at the supply input ("VCC"). If the Retain system is activated and the supply voltage falls below 19V DC, the controller changes to the safe state and stores the Retain data. In the time between the detection of the undervoltage and the complete storage of the data, the PiXtend V2 and Raspberry Pi receive power from an internal capacitor. The Retain system can only be activated and used if a nominal supply voltage of 24V DC is used. If, for example, PiXtend is operated with 12V DC, the Retain memory cannot be used and is disabled.

| Property | Value | Comment |
|--------------------------|--|--|
| Type of memory | Flash EEPROM | in the PiXtend micro-controller |
| Number of storage cycles | min. 10,000 | |
| Memory size | - 32 bytes (PiXtend V2 -S-) - 64 bytes (PiXtend V2 -L-) | |
| Voltage threshold | 19V DC (+/- 0.6V) | between VCC and GND |
| Rated supply voltage | 24V DC | to be able to use Retain |
| Max. input power via VCC | 10W | for the guaranteed storage of all data |

Table 2: Technical data – Retain system

Refer to the PiXtend V2 Software Manual for more information on using the Retain system.

Typical applications for retain memory include:

- Operation-hour counter
- Device configuration/parameters
- Language settings for the user interface
- Motor position
- Component or workpiece counter
- Data flow protection
- Record keeper of the current work step
- and much more

6.5. Preparing a SD card for your Raspberry Pi

If you have ordered a pre-installed SD card from our shop, you can start right away and preparing your own SD card is not necessary. However, if you want to prepare an SD card for your Raspberry Pi nevertheless, you will find information on how to do that in this chapter.

We provide various SD card images in our download section at <https://www.pixtend.de>. Depending on the type of image you want, the images always include a current version of the Raspbian operating system and also one of the following pre-configured options:

- PiXtend V2 Basis Image with PiXtend Linux Tools including the C-library (pxdev), PiXtend Python Library V2 (PPLV2) including examples and Node-RED
- CODESYS Control Demo with CODESYS V3 Runtime Extension for the Raspberry Pi and PiXtend

Always use an SD card or SD image from Kontron Electronics for your first tests. Our images are tested extensively and are pre-configured with all required settings. Our cards will ensure that the device is up and running quickly.

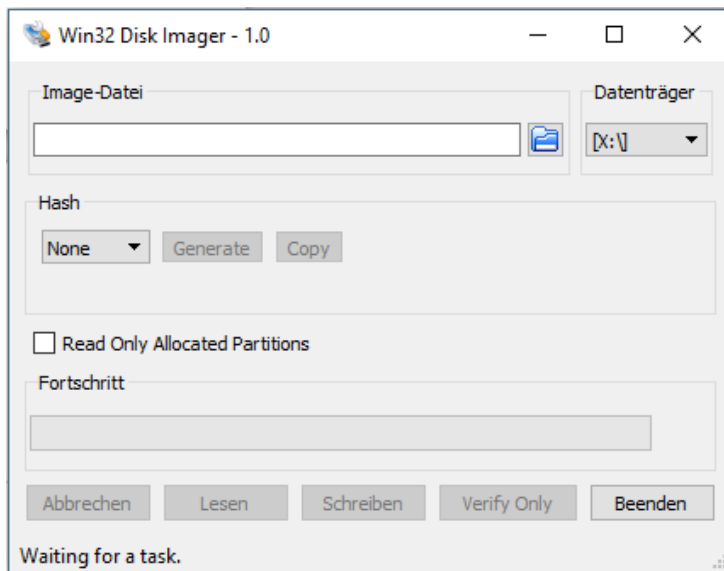


Figure 1: Software - Win32DiskImager

A PC with an SD card reader is required to transfer one of our images to a blank SD card. We recommend the open source program “Win32DiskImager” as software for writing to SD cards. The program runs under Microsoft Windows and can be downloaded free of charge from <http://sourceforge.net/projects/win32diskimager/>

NOTICE

To run or access the drive, the Win32DiskImager program must be started with administrator rights or with an administrator account. If this is not the case, it is possible that the SD card is not written or is written incorrectly!

To prepare an SD card or transfer one of our images to your own SD card, perform the following steps:

- Insert the SD card into the card reader.
- Wait for Windows to recognize the media.
- Open the workspace/This PC or Windows File Explorer and note the drive letters of the SD card – this must be specified later on in the Win32DiskImager program.
- Open and extract the zip file with the SD card image from our download section.
- Start the Win32DiskImager program with administrator rights.
- Select the extracted image file (*.img) in Win32DiskImager under “Image file”.
- Select the drive letter of the SD card under disk.

⚠ CAUTION

If you select the wrong disk drive, all data can be lost as soon as you have started the writing process in the Win32DiskImager program.

It is best to remove all removable media (such as USB sticks or USB hard disks) before you start the writing process, maybe even before you start the Win32DiskImager program.

- Once you have completed all settings, start the writing process by clicking on “Write”.
- Wait until the writing process has been completed.

After a successful transfer of the image to the SD card, it can be ejected from the PC and inserted into the SD card slot of the Raspberry Pi.

If you see a window opening during or after the creation of the SD card, with or without content, or an error message appears that says that a drive cannot be accessed, or you are prompted to format the media, click on the "Cancel" button and close any windows that may have appeared on these dialogs.

This is no error in such situations. You get these messages because Microsoft Windows is not able to read the SD card because it is intended for the Raspbian operating system. As mentioned, remove the SD card from the PC and insert it into the SD card slot of the Raspberry Pi. The next step is to connect the power supply before starting the Raspbian operating system.

6.6. Determining the network address (IP address) of the Raspberry Pi

After creating a bootable SD card for the Raspberry Pi and successfully booting it, the question often arises, which network address (IP address) does the Raspberry Pi have? The IP address is important for users who want to operate the Raspberry Pi without a screen and keyboard.

The default network settings use:

DHCP Dynamic Host Configuration Protocol

With this setting, the IP address is automatically assigned by the DHCP server in the network.

There are several ways to determine the IP address of the RPi.

1. For everyone:

Even if only temporarily, connect a screen and keyboard and enter the command:

```
ifconfig
```

This checks the IP address of the Raspberry Pi (connection "eth0").

Most DHCP server save the MAC address of the RPi and assign the same IP address every time to this MAC address.

Now the screen and keyboard can be removed, and the Raspberry Pi can be connected to remotely from a PC with SSH (e.g., Putty).

2. For CODESYS developers:

If you want to use the Raspberry Pi and the PiXtend V2 as a controller (PLC), you can use the Raspberry Pi search function built in by 3S Smart Software Solutions in the Raspberry Pi Update window. This search function scans the network to find Raspberry Pi devices. The IP addresses of all of the Raspberry Pi devices found are displayed in a list. In this list, you can either select the desired Raspberry Pi and then note the IP address or, for example, retrieve information about the system.

Please note: The Raspberry Pi update window can only be opened if the “CODESYS Control for Raspberry Pi” package is installed in CODESYS. Refer to chapter 7.3.2 Installing CODESYS Control for Raspberry Pi

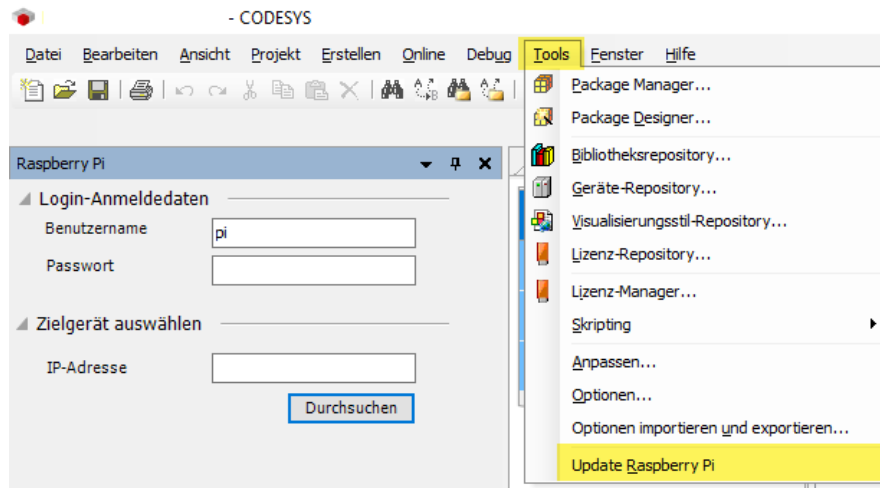


Figure 2: Basic knowledge – Determining the IP address of the Raspberry Pi in CODESYS

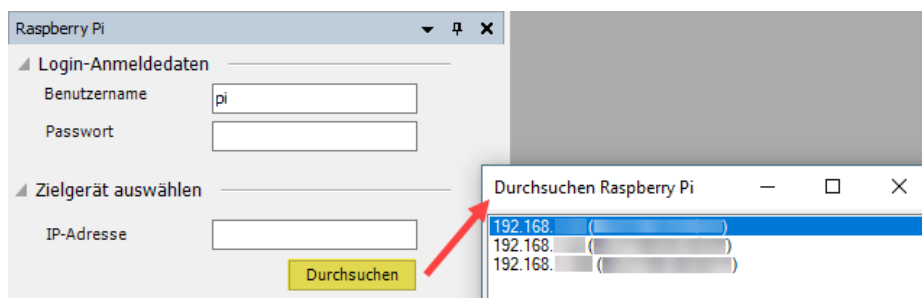


Figure 3: Basic knowledge – List of all Raspberry Pi devices found in the network

If it is not clear which Raspberry Pi is which, then it is best to switch off all Raspberry Pi devices except the desired one. Then close CODESYS, reboot the gateway and perform the search again. Sometimes it takes a while until CODESYS forgets the inactive Raspberry Pi devices; in these cases, restart the PC and start the search again. Only one device should appear in the list.

3. If nothing else helps:

Check out the Raspberry Pi Foundation website to help to determine the IP address of a Raspberry Pi device

<https://www.raspberrypi.org/documentation/remote-access/ip-address.md>.

The website lists various options, including programs for Windows or macOS; even smartphone apps are mentioned.

6.7. Raspberry Pi - Assigning a static IP address

The Raspberry Pi typically receives its network settings from the DHCP on the local network. The DHCP option is very practical, but it is better to have a static IP address for the Raspberry Pi device in some cases.

The following information is from the Raspberry Pi Learning Resources website. We recommend that you always check this website first for the most detailed information, as the Raspberry Pi Foundation makes changes to operating system settings from time to time.

Connect to the Raspberry Pi either directly, with a monitor and keyboard, or via a network using an SSH client program (such as Putty).

1. Edit the file “/etc/dhcpd.conf”:

```
sudo nano /etc/dhcpd.conf
```

2. Add the following at the end of the file:

```
interface eth0

static ip_address=192.168.0.2/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```

Insert a valid IP addresses for your network instead of the IP address 192.168.0.XXX. The entry “ip_address” is the static IP address of the Raspberry Pi device, “routers” is the IP address of the router and “domain_name_servers” is the IP address of the DNS server, which is usually the same IP address as the router. The 24 after the IP address is the network mask and corresponds to the value 255.255.255.0.

3. Save the changes:

The key combination “CTRL + O” saves the changes and “CTRL-X” closes the nano editor.

4. Perform a restart:

```
sudo reboot
```

After rebooting, the Raspberry Pi has applied the specified static IP address and can now be reached under this new IP address.

6.8. Raspbian login information (login)

NOTICE

Our images have the default Raspbian login settings as defined by the Raspberry Pi Foundation.

The login information is:

- User: pi
- Password: raspberry

6.9. Turning off the Raspberry Pi

With Raspberry Pi the question always comes up:
"Can I just unplug the power to turn off the Raspberry Pi?"

To answer this question, it is important to note that the Raspberry Pi is a "full-fledged" computer on a circuit board and like any other computer with an operating system, the sudden loss of power can lead to data loss.

Therefore, while pulling the "plug" is possible at any time, it should be avoided to ensure the integrity of the file system and data on the SD card. Similar to other computers, it can happen that the Raspberry Pi will not boot and the SD card may need to be reloaded in order to use it again.

To restart the Raspberry Pi, use the following command:

```
sudo reboot
```

Use this command to turn the Raspberry Pi off:

```
sudo shutdown -h now
```

After a few seconds, the operating system stops all services and brings the operating system to a safe state. Now the power supply can be disconnected; all data remains on the SD card.

Remark:

Pulling the plug or interrupting the power supply to the Raspberry Pi does not necessarily lead to data loss or an unusable SD card. The information on this page is intended to be an indication that, as with any other computer, there is a possibility of failure, but no certainty that it will occur.

If you have more stringent requirements, please contact us so we can help you to optimize your PiXtend application and achieve a higher data security.

6.10. Compatibility of the software component

There are many software components for the PiXtend V2, which allow the platform to be used in many different ways. The following must be observed when using these components.

NOTICE

The programs and software components listed in this document cannot be operated in parallel!

The “SPI bus” used for the communication with the micro-controller must only be used from one single program at a time. If the communication with the micro-controller is done simultaneously from different programs, and also with different commands, this may lead to defective parts of the hardware, e.g., the relays.

It is possible to use different programs “one after the other” as long as it is the “pxdev-Linux Tools & Library” or the “PiXtend Phyton Library V2 (PPLV2)”. These are programs that terminate themselves after execution and must be restarted manually.

If the CODESYS Runtime extension has been installed on the SD card or you are using our CODESYS SD card image, no other program can be used to send additional commands to the PiXtend V2 or to query information. This use is reserved exclusively for CODESYS; furthermore, CODESYS is automatically started every time the Raspberry Pi boots.

If you use the FHEM system and control your PiXtend V2 device via FHEM, the same restrictions apply as for CODESYS. FHEM is also started automatically after its installation; otherwise, problems may occur. A parallel use of FHEM and CODESYS is not possible, since there is overlap in the usage of the SPI bus.

6.11. Serial interface

To use the serial port of the Raspberry Pi computer for your own purposes, it must first be activated and configured. This means that if you use the serial interface for other purposes, the following commands will activate or change the serial interface. From now on it does not send any data by itself; access to the Linux console via the serial interface is deactivated by this procedure.

The serial interface of the Raspberry Pi can be activated or adjusted using the raspi-config program:

```
sudo raspi-config
```

Switch to the following in the program:

Interfacing Options --> Serial --> <No> --> <Yes> --> <Ok>

The following entry can then be found in the file /boot/config.txt:

```
enable_uart=1
```

This activates the UART interface. The program “raspi-config” does all the work for us.

Then restart the Raspberry Pi computer and use the serial port exclusively in your own applications.

A restart can be performed with the following command:

```
sudo reboot
```

6.12. PiXtend V2 - Notes on the serial interface

In order to work with our PiXtend eIO devices as well as with other serial devices without restrictions, we recommend using the full serial interface of the Raspberry Pi. This is also called PL011 or serial0 and has the device name `ttyAMA0`. Only this interface allows the parity bit to be used in serial communication.

All our SD card images from version 2.1.6.0 for the Basic image and the PiXtend V2 -S- CODESYS image or version 2.1.1.1L for the PiXtend V2 -L- CODESYS image use the `ttyAMA0` interface for communication. The `ttyS0` interface is no longer used.

If you want to check if you already use a modified image, it is best to look in the `config.txt` file in the Raspbian boot partition.

If you insert the SD card into a card reader, then open the SD card with the Windows Explorer and look in the `config.txt` file to see if it contains the entry `dtoverlay=pi3-disable-bt` (starting with Raspberry Pi OS Bullseye use `dtoverlay=disable-bt` instead) at the very end.

To check this directly on the Raspberry Pi, use the following command:

→ `sudo nano /boot/config.txt`

Go to the end of the file and look there for the entry `dtoverlay=pi3-disable-bt` (starting with Raspberry Pi OS Bullseye use `dtoverlay=disable-bt` instead). If it is available, then you are already using the device `ttyAMA0` as serial interface and do not need to do anything else.

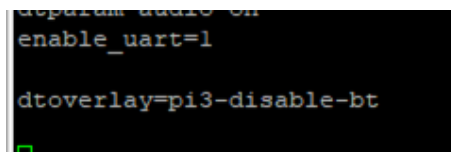


Figure 4: RPi - Turning off Bluetooth®

If this entry does not exist, insert it and carry out the following steps:

- `sudo nano /boot/config.txt`
- Go to the end of the file
- and enter: `dtoverlay=pi3-disable-bt` (starting with Raspberry Pi OS Bullseye use `dtoverlay=disable-bt` instead)
- Save the changes by pressing `control+O` → Enter
- Exit the editor by pressing `control+x`
- Perform a restart: `sudo reboot`

If you are working with CODESYS V3.5 and want to control serial devices, then after the previous modification a further adjustment is necessary so that CODESYS V3.5 uses the correct serial interface.

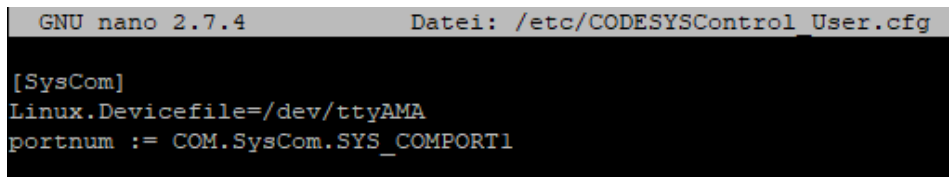
- Execute the following command: `sudo nano /etc/CODESYSControl_User.cfg`
- In the file `CODESYSControl_User.cfg` search for the section `[SysCom]`
- Change the section as follows from:

```
[SysCom]
;Linux.Devicefile=/dev/ttyS
;portnum := COM.SysCom.SYS_COMPORT1
```

- to

```
[SysCom]
Linux.Devicefile=/dev/ttyAMA
portnum := COM.SysCom.SYS_COMPORT1
```

- remove the two semicolons, change the device name from `ttyS` to `ttyUSB`; do not add a number at the end of the device name, CODESYS will do this later itself.
- Save the changes by pressing `control+O` → Enter
- Exit the editor by pressing `control+x`
- Reboot the Raspberry Pi with this command: `sudo reboot`



```
GNU nano 2.7.4      Datei: /etc/CODESYSControl_User.cfg

[SysCom]
Linux.Devicefile=/dev/ttyAMA
portnum := COM.SysCom.SYS_COMPORT1
```

Figure 5: CODESYSControl_User.cfg - serial connection ttyAMA

Remember to deactivate the login shell on the serial0 / ttyAMA0 interface.

NOTICE

Bluetooth® can no longer be used after this change!

Usage of the RS485 interface on the PiXtend V2-L-:

To use the RS485 interface on the PiXtend V2 -L-, the “GPIO 18” auf the Raspberry Pi has to be activated as an output and set to logical 1 or True. This switches the serial interface of the Raspberry Pi to the RS485 chip on the PiXtend V2 -L- board. The RS232 interface can not be used after this change.

6.13. Information on security and cyber security

Anyone running a computer system, their own computer or a Raspberry Pi computer, nowadays should be concerned about the security of the system and take precautions. This applies in particular to devices that have a network connection and/or a WLAN module. It makes no difference whether the system in question is operated in a private household, in the office of a company or as part of a system in a production facility.

Below we give recommendations and food for thought in regard to what you should do when using a Raspberry Pi in terms of IT security to increase or improve system security. Please note, this is only a small selection of possibilities. We recommend that all users continue to obtain detailed information via the Internet. Our recommendations are intended as a starting point for your search. For industrial use of PiXtend devices, please consult your internal IT or cyber security department. They are usually very familiar with Linux systems, including the Raspberry Pi computer. If this is not the case, it is advisable to consult external companies or consultants on these topics. If PiXtend is operated in an isolated network, no connection to a network is established and the wireless interfaces are deactivated, the implementation of a secure IT solution is much easier.

6.13.1. Changing the password

The first and probably most important point is changing the password of the “pi” user. The default login information is always the same for each newly installed Raspbian; it is available on the Internet and we provide it in this documentation. Use the Linux command “passwd” immediately after the first start to change the password of the “pi” user.

For example, if the RPi is to be accessible via SSH, the login can be switched from password authentication to certificate authentication. In addition, a new user, assigned the same rights as the “pi” user, can be created and the “pi” user can be deleted. Further information can be found on the Internet.

6.13.2. Raspberry Pi pin headers

If the Raspberry Pi is accessible via the Internet, extreme caution is required. Regardless of whether the Raspberry Pi is operated in a home network or in a company network, it is best to avoid direct Internet access. If it is still necessary, for example, use the port forwarding function of your router or ask your IT department what options are available in your company. This solution is only recommended for very advanced users and most companies will not allow direct accessibility.

Instead, rely on a VPN connection and use it to access your Raspberry Pi when remote access is needed. Residential routers often have an easy setup for VPN access.

6.13.3. Check commands and scripts

The Internet offers information and tips about Raspberry Pi, not least because of the ever-growing community. Unfortunately, among all these offers there are also untrustworthy sources. When experimenting and trying out new things with the Raspberry Pi, one tends to copy commands from a web page or other sources without questioning what this command does. If the command also requires superuser rights (root rights), i.e., you have to use "sudo", something can quickly go wrong.

In the case of unknown commands from unknown sources, it should first be checked which command is involved and what effects are associated with the use of the command. If you are asked to download something, e.g., with wget or curl, one should consider whether you can trust the source or not.

6.13.4. Creating system backups

Backups of the Raspberry Pi's SD card can be created quite easily using the Win32 Disk Imager (Windows) program, see Chapter 6.5 Preparing a SD card for your Raspberry Pi. Since the SD card contains all the information that the Raspberry Pi needs to operate, it is easy to restore the device using a replacement card. This procedure is a simple and quick way to get the system back into operation after an incident.

We recommend to document finished projects at key points and to record all changes and the installed software. This gives you the opportunity to track changes and you can rebuild the system at any time.

6.13.5. Installing updates

An important point of cyber security is the installation of updates; there is an active development of the Linux kernel and the Raspbian distribution with continuous updates. This is done in order to permanently improve the system and to close any security gaps found.

The question arises as to how and under what circumstances updates can be installed to achieve maximum security. Instructions on the Internet show how to install updates automatically, similar to Windows.

Remark:

Installing updates is important. However, updates can be critical with regard to system stability and function in some cases, especially when it comes to control applications.

Updating programs can usually be carried out without any risks. However, updating parts of the operating system or the entire operating system may lead to incompatibilities with existing or self-written programs. If, after an update, programs do not work as expected or the update fails, the device must be serviced and repaired. Refer to the section 6.13.4 Creating system backups.

We recommend that you do not install updates on the production system, but first carry out tests with a test device to determine whether everything works as expected after an update.

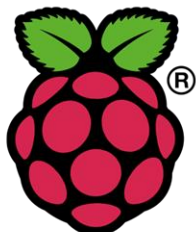
CODESYS

This chapter describes the installation of all necessary software components to program the PiXtend V2 (www.pixtend.de) with CODESYS (www.codesys.de).



The CODESYS development environment is designed for the professional hardware-independent programming of controllers according to IEC 61131-3 and is developed by CODESYS GmbH.

In order to program a Raspberry Pi in CODESYS the Raspberry Pi Runtime extension for CODESYS, also developed by CODESYS GmbH is required.



Special device drivers for PiXtend V2 are required to get direct access to the I/O hardware and interfaces of the PiXtend V2 board using CODESYS. The PiXtend V2 drivers are provided free of charge by Kontron Electronics GmbH.



This chapter is intended for those who want to create a project for PiXtend V2 using CODESYS.

NOTICE

The use of CODESYS is recommended for PiXtend V2 -S- and PiXtend V2 -L- is the same, only the appropriate SPI driver must be selected for the control of the I/O hardware (also called I/O mapping), since this is where there are differences between devices.

7.1. Note

7.1.1. Copyright of texts and images

Texts and images marked with the abbreviation "3S" are from the company 3S-Smart Systems GmbH - www.codesys.com.

Texts and images marked with the abbreviation "RPI" are from the company Raspberry Pi Foundation – www.raspberrypi.org.

Text and images which are not marked or marked with the abbreviation "QS" or "KED" are from the company Kontron Electronics GmbH – www.pixtend.de.

7.1.2. Compatibility

CODESYS projects and programs written for or on other CODESYS controllers can often be transferred to other CODESYS compatible controllers with little effort.

The CODESYS Runtime extension for the Raspberry Pi computer turns it into a full-fledged CODESYS PLC; all compatible CODESYS programs can be run.

Only the hardware compatibility has to be considered; it has to be checked if there are matches or differences. It may be necessary to make adjustments to the hardware configuration in the program so that the correct signals arrive at the correct position in the CODESYS project.

If you are unsure whether your existing CODESYS PLC project works with PiXtend V2, please contact us. We will be happy to advise you on necessary changes and can help to transfer the program.

7.2. Requirements

7.2.1. System requirements for CODESYS

- Microsoft Windows 7/8/10/11 (32/64 bit)
- Suitable PC hardware for the corresponding Microsoft Windows platform

7.2.2. Required hardware

- PiXtend V2 Board (www.pixtend.de)
- Raspberry Pi Model B+/2B/3B/3B+/4B
- Standard RJ45 network cable
The network settings of the Raspberry Pi are set to DHCP by default. In most cases, the IP address is assigned by the DHCP server on the router.
- For initial installation of a new Raspberry Pi:
 - SD card reader, internal or external
 - SD memory card, minimum 4GB (recommended 8GB)

microSD for Raspberry Pi model B+/2B/3B/3B+/4B
 - Optional: HDMI-compatible monitor
 - Optional: USB keyboard
 - Optional: USB mouse

7.3. Installation of the required software components

7.3.1. CODESYS development environment

7.3.1.1 Introduction of CODESYS

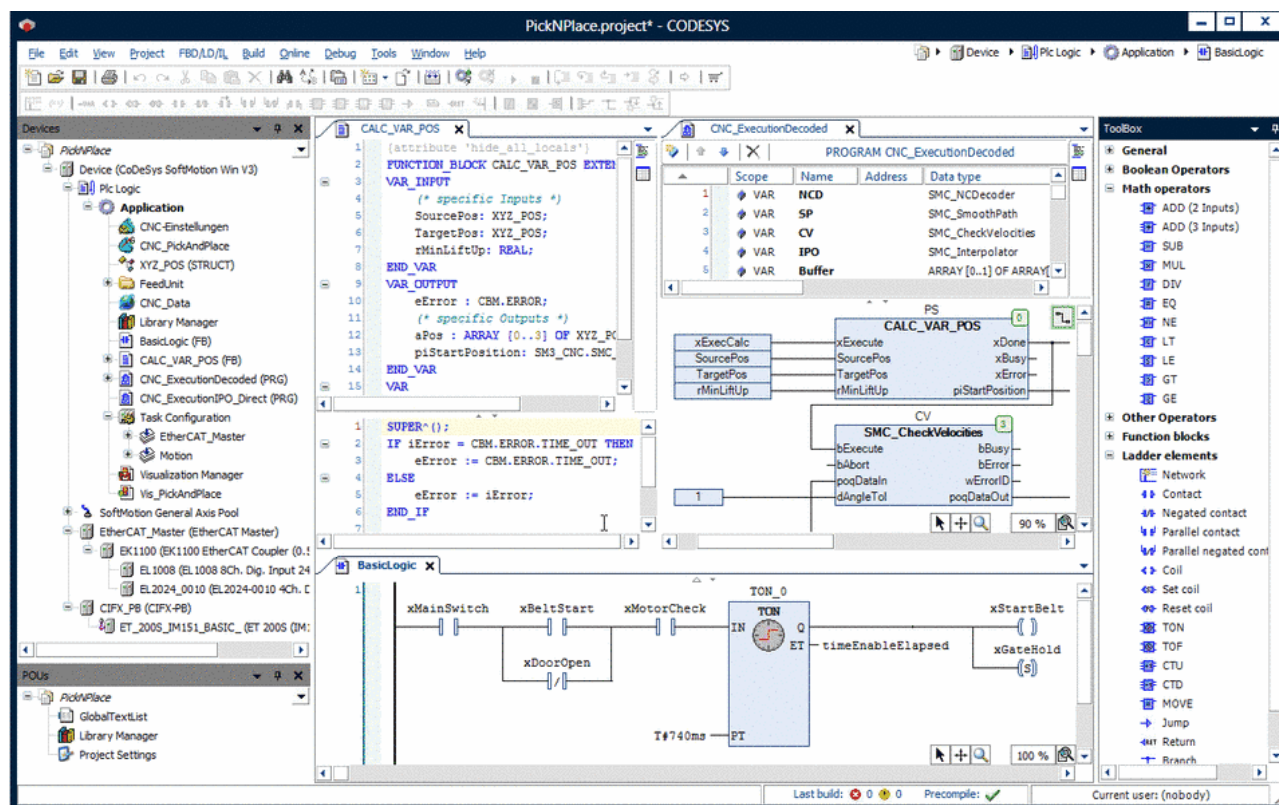


Figure 6: CODESYS Development System (source: 3S)

CODESYS is a software platform for tasks in industrial automation technology. The basis is the IEC 61131-3 programming tool CODESYS Development System V3. The tool offers the user integrated solutions - with the aim of providing practical support in implementing tasks. (Source: 3S)

The hardware-independent programming system CODESYS from the company CODESYS GmbH is ideally suited for use with the Raspberry Pi and PiXtend V2. In addition to programming in all languages for programmable logic controllers (PLC) according to the IEC 61131-3 standard, it is also possible to create web visualizations. With the "Webvisu" you can easily transfer the contents and controls of your program to a website (CODESYS web server running on the Raspberry Pi). You do not need any knowledge of web programming (HTML, PHP, CGI, etc.) to create and work with Webvisu.

With the modern graphic interface of CODESYS, you are well equipped for the programming of the control system and webvisu. The Webvisu can be accessed with any current smartphone, tablet or PC/MAC. Only a current web browser is needed. We recommend the latest versions of browsers like Google Chrome, Microsoft Edge or Mozilla Firefox.

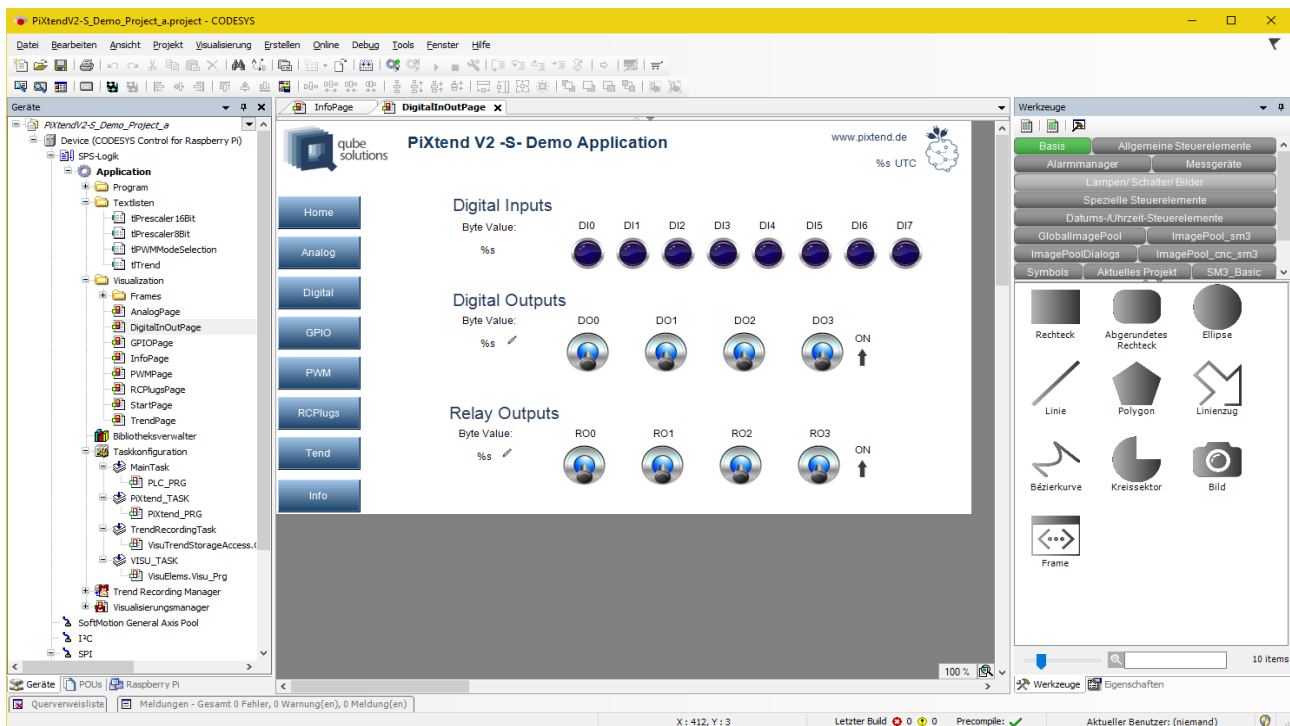


Figure 7: CODESYS - Visualization for the PiXtend V2 demo project

7.3.1.2 CODESYS Download

The CODESYS development environment can be downloaded free of charge from CODESYS (registration required). It is easy and free to register.

- To do this, go the website <https://store.codesys.com>.
- Click on the Login button and register as a new customer in the "CODESYS Store" (create user account).
- Fill out the form completely and note that a correct e-mail address is required.
- Click on "Register Now" and follow the instructions on the store page or the instructions you receive by email; you may need to confirm your email address and wait until your account is activated.
- With your self-assigned password and your username, you can log into the CODESYS Store and download the latest CODESYS version.

7.3.1.3 CODESYS installation

Install the CODESYS development environment with the default settings and as administrator user or perform the installation with administrator rights. CODESYS itself can be started without administrator rights; a "normal" Windows user is sufficient. Now a fully-fledged CODESYS development environment is available with which you can program the PiXtend V2 PLC according to the IEC 61131-3 industrial standard.

7.3.2. Installing CODESYS Control for Raspberry Pi

7.3.2.1 CODESYS Control for Raspberry Pi - Download

To use a Raspberry Pi with CODESYS, you need the free CODESYS Control Runtime extension for the Raspberry Pi from the CODESYS Store. Starting with CODESYS version V3.5 SP18 you can use the CODESYS Installer to install the CODESYS Raspberry Pi runtime package in CODESYS, no need to visit the store.

The CODESYS Store includes both free and paid products, which can be integrated into CODESYS as additional features.

Login to the CODESYS Store to download the files. You have already created a corresponding account according to the specifications in a previous section or use the CODESYS Installer starting with version V3.5 SP18.

- Directly after login you can download all products marked with "Free" without any payment.
- Click on the product "CODESYS Control for Raspberry Pi SL"
- Click the download button and wait for the download to complete.
- A "First Steps" guide can be found in the CODESYS Online help at:
https://content.helpme-codesys.com/en/CODESYS%20Control/_rtsl_install_runtime_on_controller.html
-

Notes on the demo version:

- The free version is limited to two hours of running time. After that, the runtime environment automatically ends until it is restarted. After each restart, the Raspberry Pi can be used again for two hours without restrictions. This is usually sufficient for initial tests.
- A license for a time-unlimited version is available. You can perform the licensing at a later time in the CODESYS Store.
- We recommend everyone who is starting out with PiXtend V2 and CODESYS for the first time, not to install the first CODESYS license on the Raspberry Pi but to purchase an additional CODESYS key (CODESYS license dongle) and save the license there. This has the advantage that on the one hand the license is transferrable, should the worst come to worst and the Raspberry Pi has to be replaced, and on the other hand the SD card can simply be exchanged without having to worry about the CODESYS license.

7.3.2.2 CODESYS Control for Raspberry Pi – Installation

Start CODESYS and open the Package Manager (since SP18 use the CODESYS Installer) via the Tools menu → Package Manager/CODESYS Installer.

Click on “Install” in the Package Manager (CODESYS Installer: Browse for Package, select and then install) and first select the „CODESYS Edge Gateway for Linux 4.X.X.X.package“ package and install this file. Perform the installation using the default options. You might have to close CODESYS in order for the installation to start. For the CODESYS Installer to run CODESYS has to be closed. Starting with SP18 use the CODESYS Installer to install the CODESYS Raspberry Pi package and all needed other packages in one go.

Now click on “Install” in the Package Manager / CODESYS Installer again and select the „ CODESYS Control for Raspberry Pi 4.X.X.X.package “ file and install it. Perform the installation using the default options. You might have to close CODESYS in order for the installation to start. Using the CODESYS Installer CODESYS must be closed.

Note: With newer versions of CODESYS V3.5, starting with SP18, use the CODESYS Installer.

NOTICE

For the installation you need the administrator rights on the computer; otherwise, the installation may fail. If in doubt, start CODESYS with a right click and select “Run as Administrator” and then start the “Package Manager” to install the packages. The CODESYS Installer has a button/clickable link at the bottom of the screen which allows the restart of the app with administrative rights.

The package includes:

- CODESYS Extension for the Raspberry Pi (drivers and libraries)
- Example programs (by default in the user directory under “CODESYS Control for Raspberry Pi”)
- Debian Package codesyscontrol_arm_raspberry_V4.X.X.X.deb for the installation of the Runtime on the Linux system of the Raspberry Pi

NOTICE

A manual with more detailed technical information and installation instructions can be found in the CODESYS online help under Addons: https://content.helpme-codesys.com/en/CODESYS%20Control/_rtsl_product_overview.html

Select the desired language at the top.

The document also explains the licensing options for the CODESYS Raspberry Pi Runtime extension.

7.3.3. Creating a bootable SD card for the Raspberry Pi

Download the latest PiXtend V2“CODESYS” SD card image from the download section <https://www.pixtend.de>.

The image is based on the Raspberry Pi OS Linux operating system and is already pre-configured for use with CODESYS and PiXtend V2.

Alternatively, you can download and install the latest version of Raspberry Pi OS as described at

<https://www.raspberrypi.com/software>.

Follow the installation instructions, which can be found in the CODESYS Online Help under Add-on “https://content.helpme-codesys.com/en/CODESYS%20Control/_rtsl_install_runtime_on_controller.html”

For beginners, we strongly recommend using our PiXtend V2 image “CODESYS Control” for a simple and trouble-free start.

7.3.4. Installing PiXtend V2 CODESYS device driver

To access the PiXtend V2 hardware in CODESYS, you need the PiXtend V2 CODESYS device drivers. This driver is available from our Git server (git.kontron-electronics.de) or in the CODESYS store.



7.3.4.1 PiXtend V2 CODESYS device driver - download

The package “PiXtend V2 Professional for CODESYS” contains the CODESYS device drivers, sample programs and a detailed manual for the use of CODESYS with PiXtend V2.

7.3.4.2 PiXtend V2 CODESYS device driver– installation

Install the file “PiXtend_V2_Professional_for_CODESYS_V2_X_X.package” using the CODESYS package manager (starting with SP18 use the CODESYS Installer, Internet connection required) to perform the installation of PiXtend V2 device drivers, example projects and PiXtend V2 documentation for CODESYS. By default, the contents are stored in the user directory under “PiXtend V2 Professional for CODESYS”.

NOTICE

For the installation you need the administrator rights on the computer; otherwise, the installation may fail. If in doubt, start CODESYS with a right click and select “Run as Administrator” and then start the “Package Manager” or “CODESYS Installer” to install the PiXtend V2 Package.

7.4. Further steps

Your CODESYS development environment is now prepared for use with PiXtend V2.

To create an empty CODESYS project with PiXtend V2 support, see “Chapter 7.5 CODESYS - Creating Projects”.

7.5. CODESYS – Creating a project

PiXtend This chapter describes all necessary steps to create a new PiXtend V2 project (www.pixtend.de) in CODESYS (www.codesys.com). You will learn to use V2 as a CODESYS device and to create a simple visualization for your project.

7.5.1. Step by step to create your first PiXtend V2 CODESYS program

7.5.1.1 Create CODESYS standard project for PiXtend V2

Start CODESYS.

Create a new project by clicking on File → New Project in the main menu (shortcut Ctrl+N)

Select “Standard Project” from the category “Projects”, give the project a name (here “PiXtendTest”) and confirm with “OK”

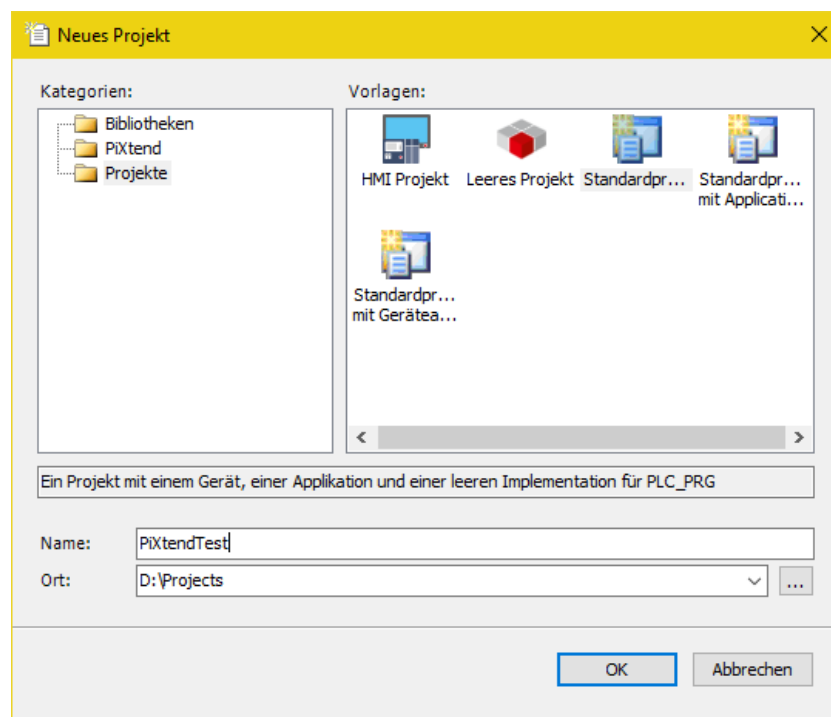


Figure 8: CODESYS - New project

Select "CODESYS Control for Raspberry Pi" as the device and select "Continuous Function Chart (CFC)" as the programming language for the main program PLC_PRG.

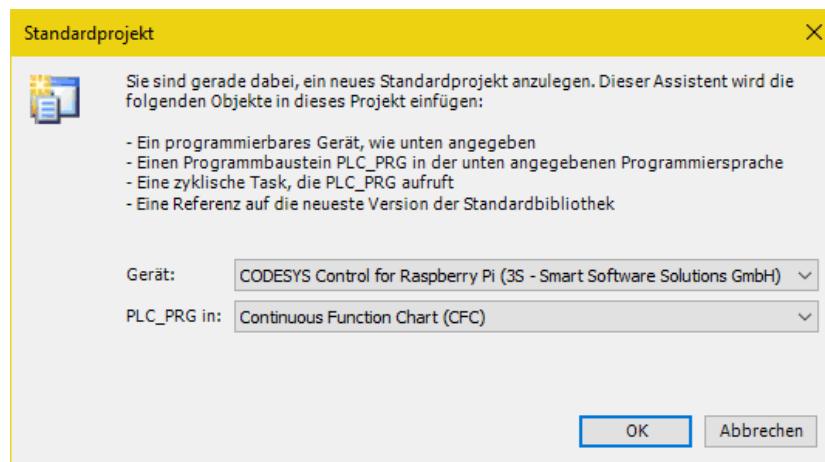


Figure 9: CODESYS - New project - Device selection

After CODESYS has created the standard project, the project has the following structure:

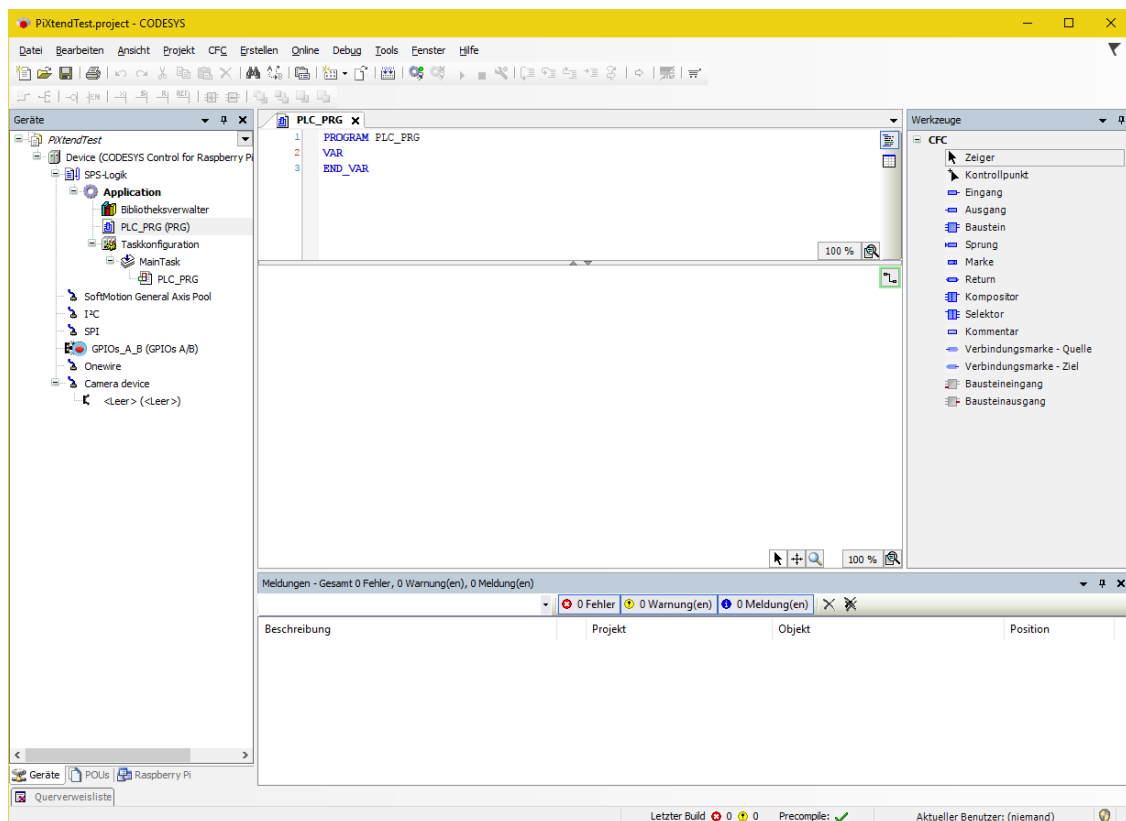


Figure 10: CODESYS - New project

7.5.1.2 Add SPI device

In the project tree right, click on the entry “SPI” and select “Add device”.

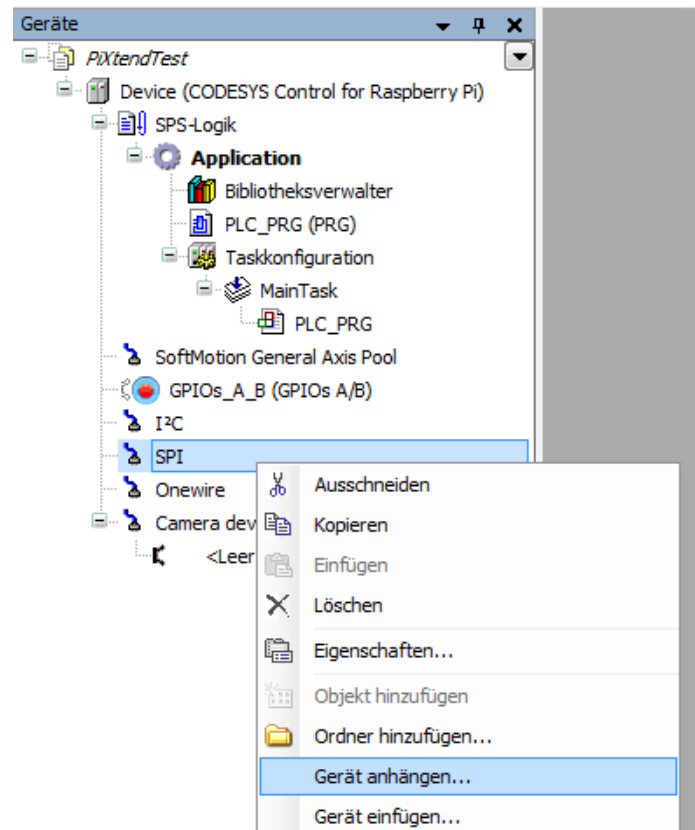


Figure 11: CODESYS - Add device

Select "SPI master" as the device and click on "Add device". Leave the window open.

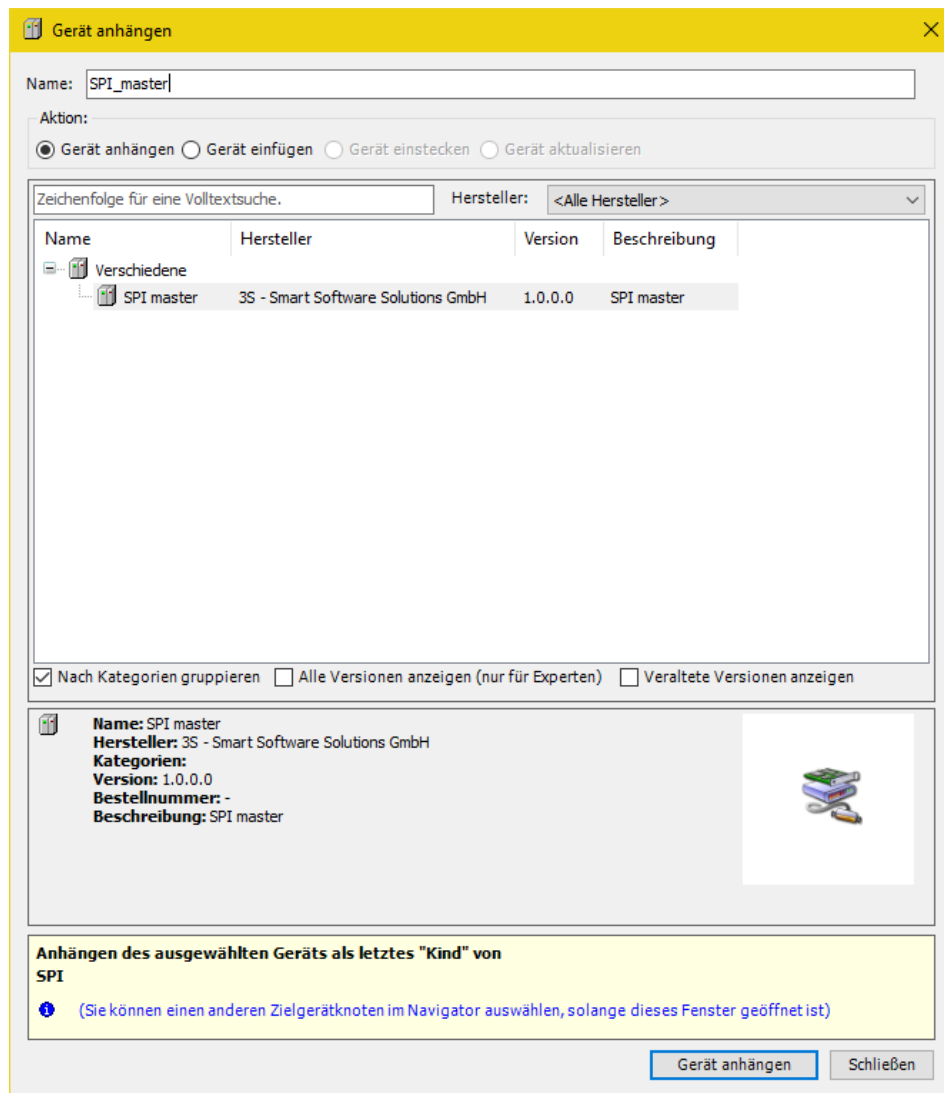


Figure 12: CODESYS - Add device - Modbus master

There is now a new entry "SPI_master (SPI Master)" in the project tree under SPI.

7.5.1.3 Add PiXtend V2 -S- device

While the window is open, click with the left mouse button on the SPI Master that was just created. The contents of the “Add device” is updated.

Select “Kontron Electronics GmbH” from the Device Manufacturer drop-down menu, then select the device from “PiXtend V2 -S-”* (not PiXtend V2 -S- DAC) and click on the “Add device” button at the bottom right and close the window.

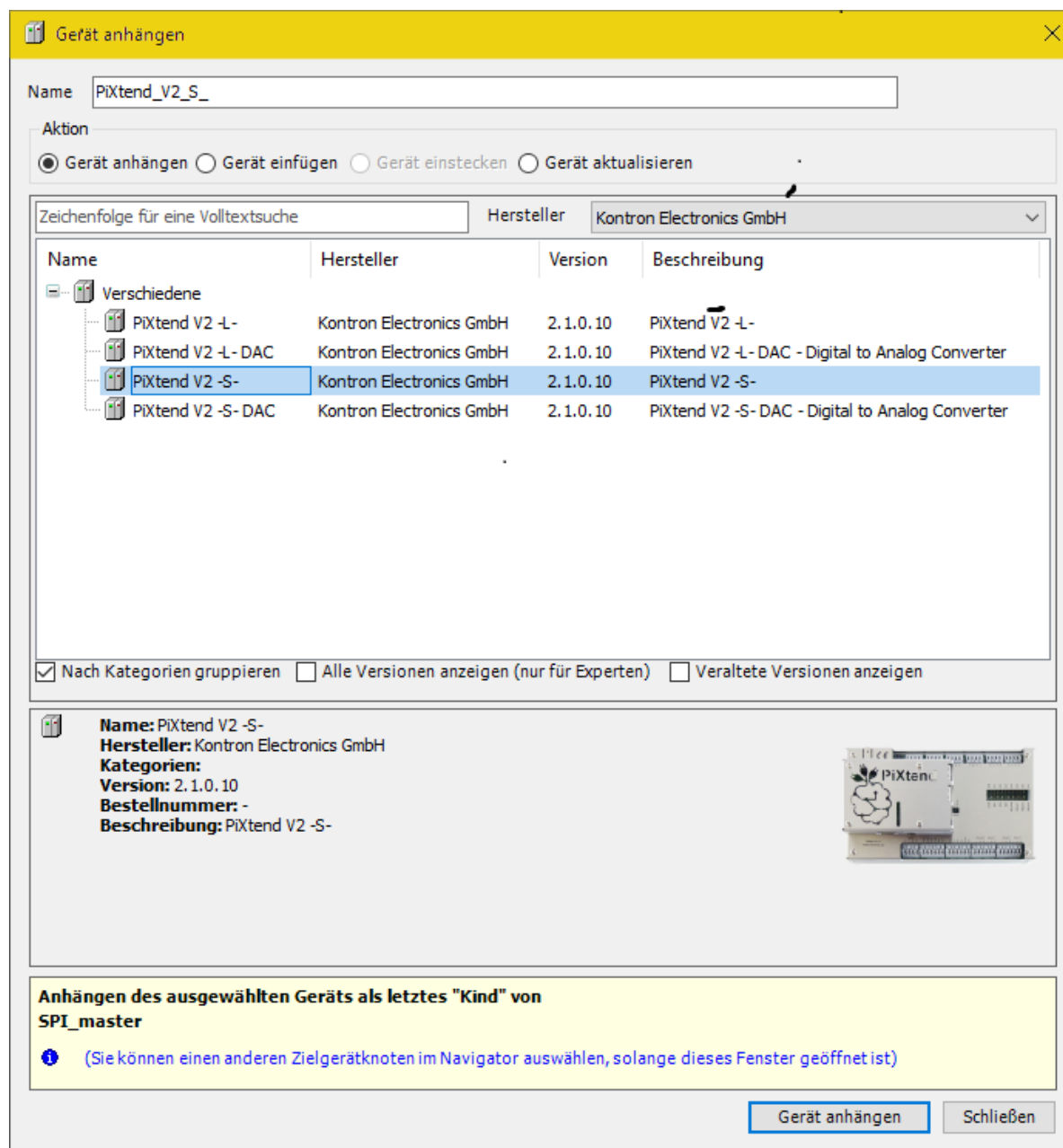


Figure 13: CODESYS - Add device - PiXtend V2 -S-

Now PiXtend_V2_S_ appears as a device under “SPI_master (SPI Master)”.

NOTICE

Alternatively you can use a PiXtend V2 -L- device instead of PiXtend V2 -S- at this point. The example described here is equally suitable for both PiXtend V2 devices. Note that the device names may be different.

Double-click on PiXtend_V2_S_ in the project tree to open the configuration page for the device. The "SPI devices I/O Mapping" tab shows which inputs and outputs of the PiXtend V2 -S- are available for use in CODESYS. During operation ("Online to controller"), the entire process image of the PiXtend V2 -S- can be monitored in this window. Input values can be monitored, and output values can be set directly. Since the values are to be used in our main program as well as in the visualization, we will create a "Global Variable List" to assign variables to the inputs and outputs.

7.5.1.4 Creating Global Variable List

Right-click on the "Application" entry in the project tree and use "Add Object" -> "Global Variable List" to add a new Global Variable List with the name "GVL":

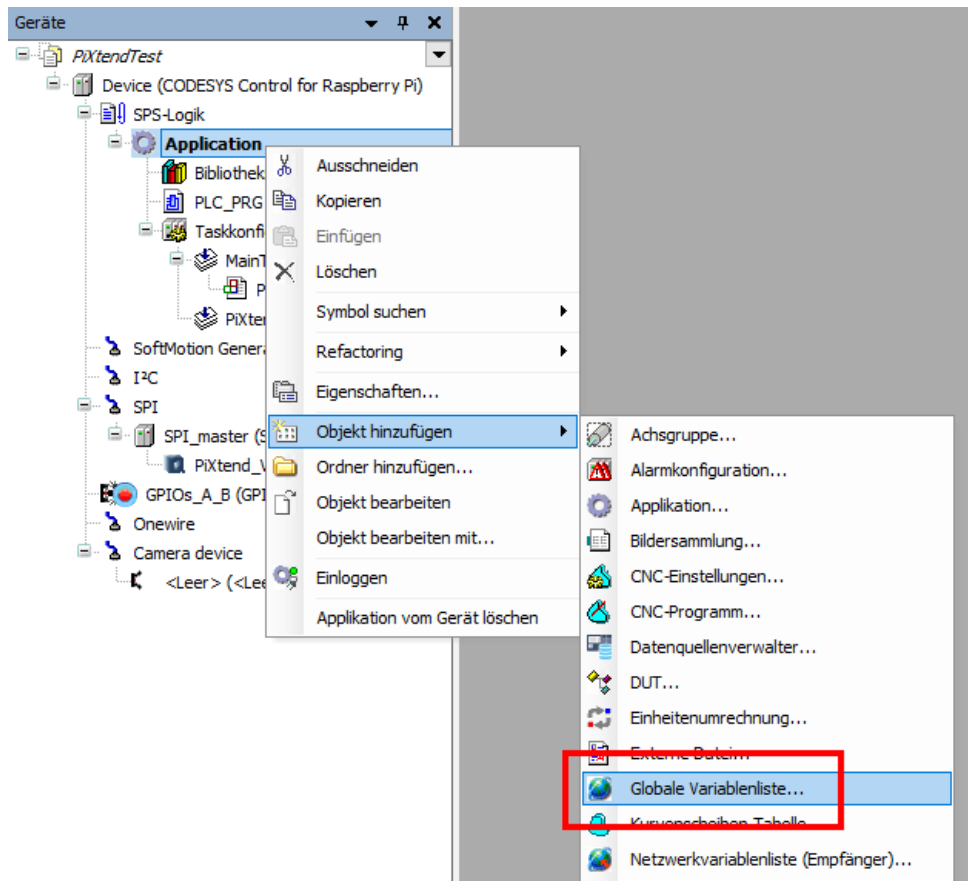


Figure 14: CODESYS - Adding Global Variable List

Open the GVL and add the following entries:

```
//Digital Inputs
xDI0: BOOL;
xDI1: BOOL;
//Digital Outputs
xDO0: BOOL;
xDO1: BOOL;
//Status
xRun: BOOL;
byHardware: BYTE;
byFirmware: BYTE;
```

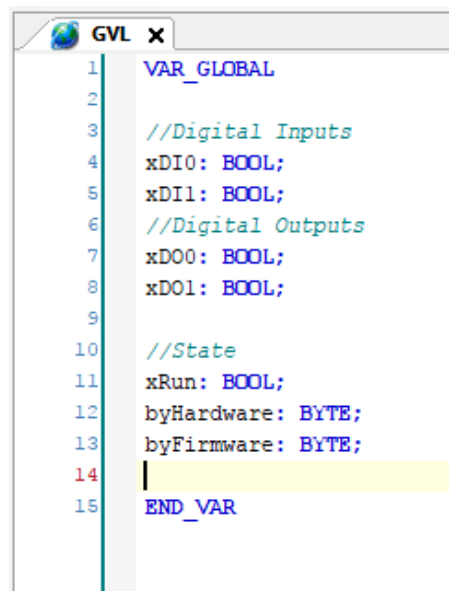


Figure 15: CODESYS - GVL - Declarations

To keep this chapter simple, we will limit ourselves to the use of two digital inputs and two digital outputs for the time being.

xDI0 stands for the first digital input, i.e. digital input 0, and xDO0 for digital output 0. Note the difference between O (letter) and 0 (number).

Prefixes for variable names have proven particularly useful in larger projects.

We recommend using the prefix x for BOOL variables (true or false) to prevent confusion with byte (prefix by) variables.

More prefixes:

x → BOOL
 by → BYTE
 w → WORD
 dw → DWORD
 r → REAL
 s → STRING

The status bit xRun provides information on the status of the micro-controller.

The bytes byHardware and byFirmware provide information on the hardware revision of the PiXtend V2 -S- and the firmware version of the micro-controller.

The naming of the variables in the GVL is up to you, we use the same conventions as in the PiXtend V2 demo project to help you get started.

7.5.1.5 Mapping variables

After you have created the required variables in the GVL, they must be mapped accordingly, i.e. assigned to the respective PiXtend V2 -S- hardware inputs and outputs.

Open the previously used "SPI devices I/O mapping" by double-clicking on the PiXtend V2 -S- device. Select the folder "Digital Inputs" and the channel "DigitalInputs" (in the picture of type byte with address %IB22):

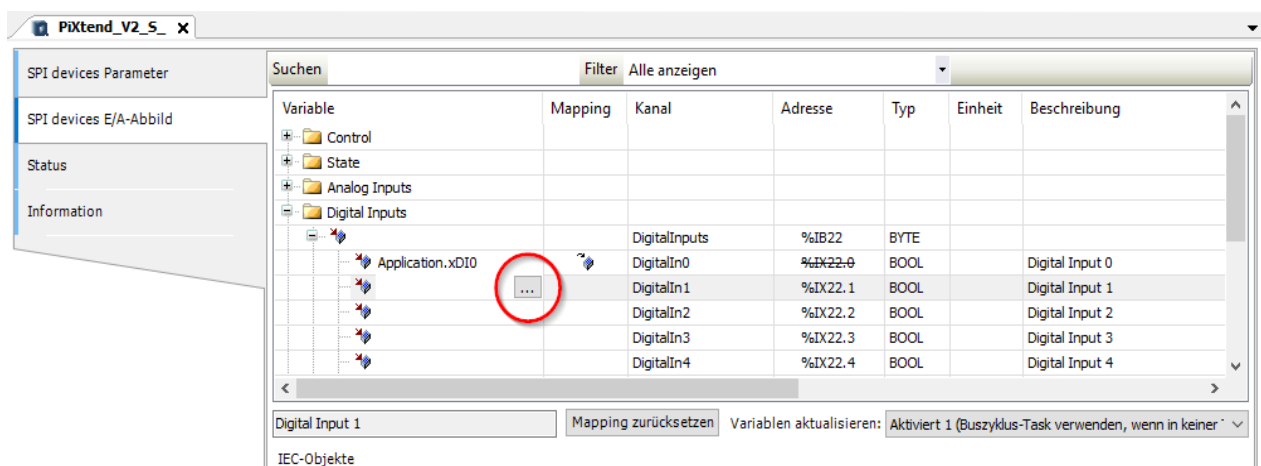


Figure 16: CODESYS - PiXtend V2 -S- I/O mapping

Now the existing input variables can be mapped bit by bit. If you double-click on the still empty left column, three dots (...) appear. A click on it opens the Input assistance, with which the desired variable is selected.

Select the variable Application → GVL → xDI0 for DigitalIn bit 0

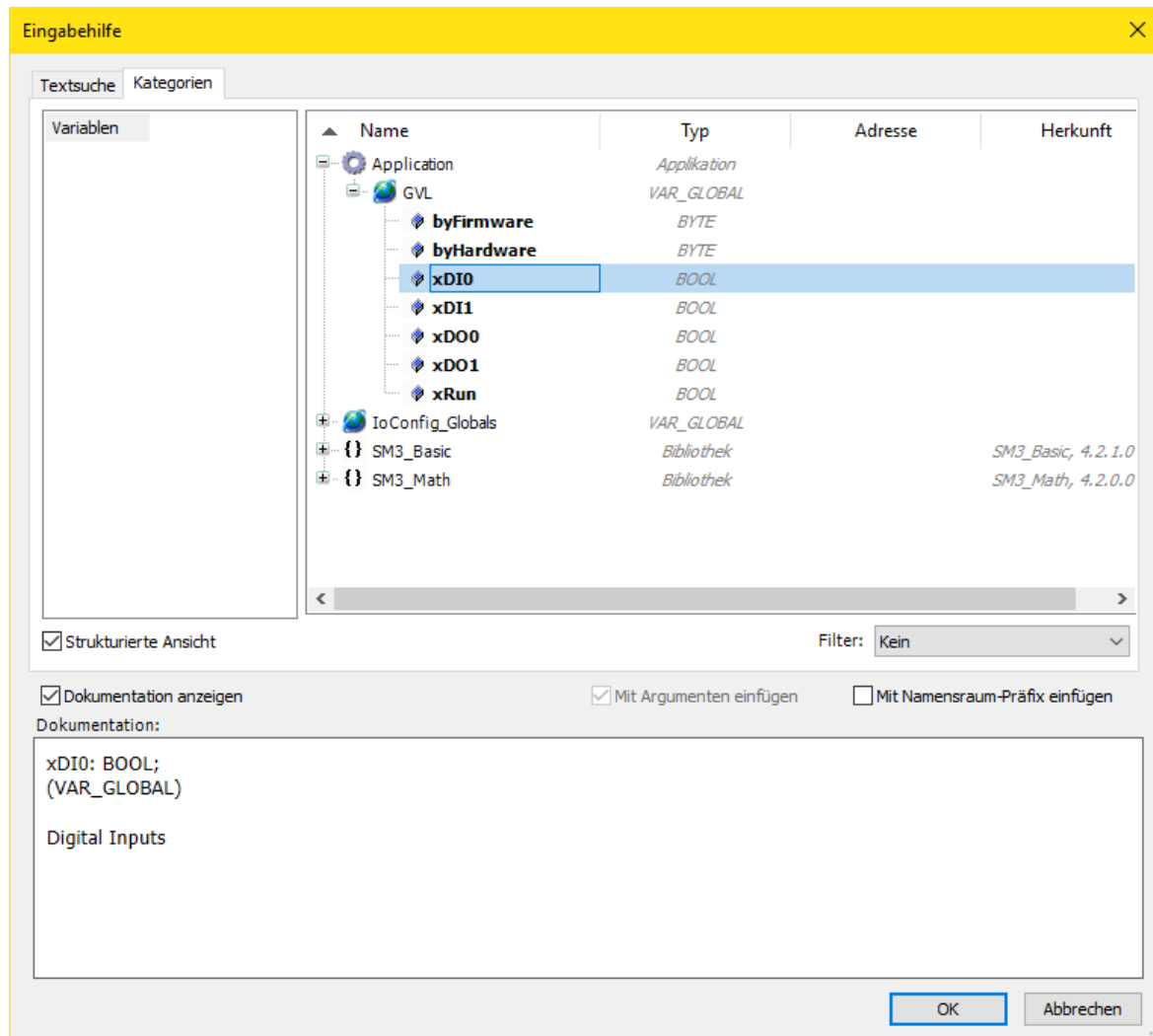


Figure 17: CODESYS - Input assistance

Repeat the process for the remaining variables:

Application → GVL → xDI0 for Digital Input → DigitalIn → Bit 0

Application → GVL → xDI1 for Digital Input → DigitalIn → Bit 1

Application → GVL → xDO0 for Digital Out → DigitalOut → Bit 0

Application → GVL → xDO1 for Digital Out → DigitalOut → Bit 1

Bytes and bits (BOOLs) can be mapped according to the same principle:

| | | | | | |
|------------------------|--|--------------------|--------|------|--|
| State | | | | | |
| Application.byFirmware | | Firmware | %IB0 | BYTE | |
| Application.byHardware | | Hardware | %IB1 | BYTE | |
| | | ModelIn | %IB2 | BYTE | |
| | | RetainCRCErr | %IX3.0 | BIT | |
| | | RetainVoltageError | %IX3.1 | BIT | |
| | | Error0 | %IX3.2 | BIT | |
| | | Error1 | %IX3.3 | BIT | |
| | | Error2 | %IX3.4 | BIT | |
| | | Error3 | %IX3.5 | BIT | |
| Application.xRun | | Run | %IX3.6 | BIT | |

Figure 18: CODESYS - PiXtend V2 -S- I/O mapping - State

Application → GVL → xRun for Status → Run

Application → GVL → byFirmware → Firmware

Application → GVL → byHardware → Hardware

Lastly, the GPIO bit 24 of the Raspberry Pi must be configured as an output. Open the Raspberry Pi GPIO configuration by double clicking on "GPIOs_A_B" in the project tree. Select "Output" for GPIO24:

PLC_PRG

PiXtend_V2_S_

GVL

GPIOs_A_B x

GPIOs Parameter

GPIOs E/A-Abbild

Status

Information

| Parameter | Typ | Wert | Standar... | Einheit | Beschreibung |
|-----------|---------------------|----------|------------|---------|-------------------------|
| GPIO4 | Enumeration of BYTE | not used | not used | | configuration of GPIO4 |
| GPIO17 | Enumeration of BYTE | not used | not used | | configuration of GPIO17 |
| GPIO18 | Enumeration of BYTE | not used | not used | | configuration of GPIO18 |
| GPIO22 | Enumeration of BYTE | not used | not used | | configuration of GPIO22 |
| GPIO23 | Enumeration of BYTE | not used | not used | | configuration of GPIO23 |
| GPIO24 | Enumeration of BYTE | Output | not used | | configuration of GPIO24 |
| GPIO25 | Enumeration of BYTE | not used | not used | | configuration of GPIO25 |
| GPIO27 | Enumeration of BYTE | not used | not used | | configuration of GPIO27 |
| GPIO28 | Enumeration of BYTE | not used | not used | | configuration of GPIO28 |

Figure 19: CODESYS - GPIO parameter - GPIO24

Then go to the “GPIO I/O Mapping” tab and click on “Outputs”. Enter “rpi_gpio24” as a variable for bit 24.

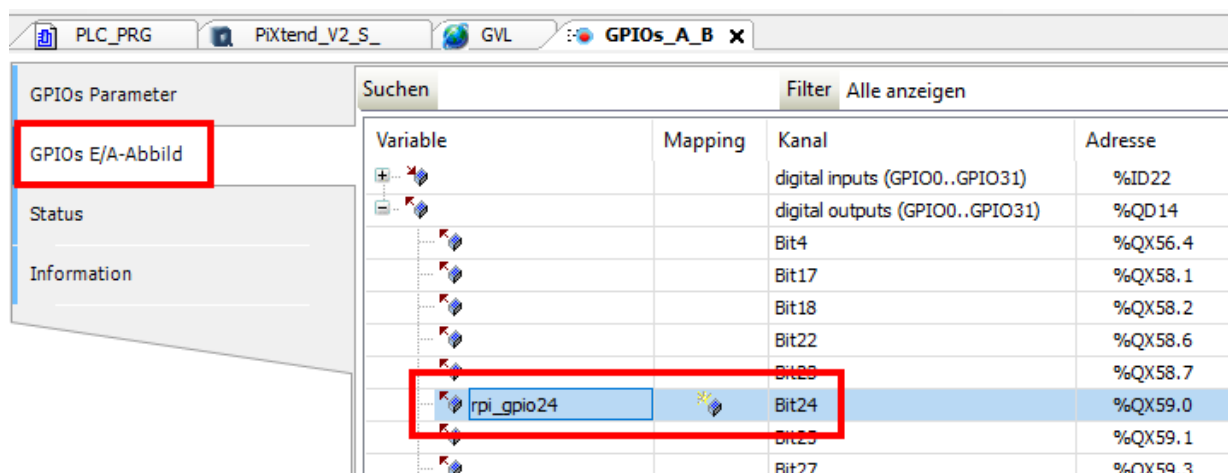


Figure 20: CODESYS - GPIO I/O mapping - rpi_gpio24 variable

7.5.1.6 Creating the main program

Now it is time to create a program which monitors the two inputs inputs DI0 and DI1 and sets the two outputs DO0 and DO1 according to our program logic.

Sequential programs (for example in C or Python) are usually executed only once, from top to bottom. With programmable logic controllers (PLC), it is necessary that certain program parts are executed cyclically at fixed intervals. This checks whether input variables have changed and whether the outputs must be reset or reset according to the program logic.

A task is used to determine the order in which a program is executed and how often.

The typical sequence within a PLC cycle is:

- Retrieve and store all hardware input values
- Run the program logic to calculate new output values
- Assign all hardware output values

Click on Task Configuration → MainTask to set the cycle time to t # 100ms. This cycle time is independent of the hardware used, since it has its own task. The automatically created PiXtend task has a cycle time of 30 ms and should not be changed. For more information, refer to the chapter 6.2 SPI Communication, Data Transmission and Cycle Time.

The notation t# symbolizes that the value is a time specification and is usually used when programming controllers according to IEC 61131-3.

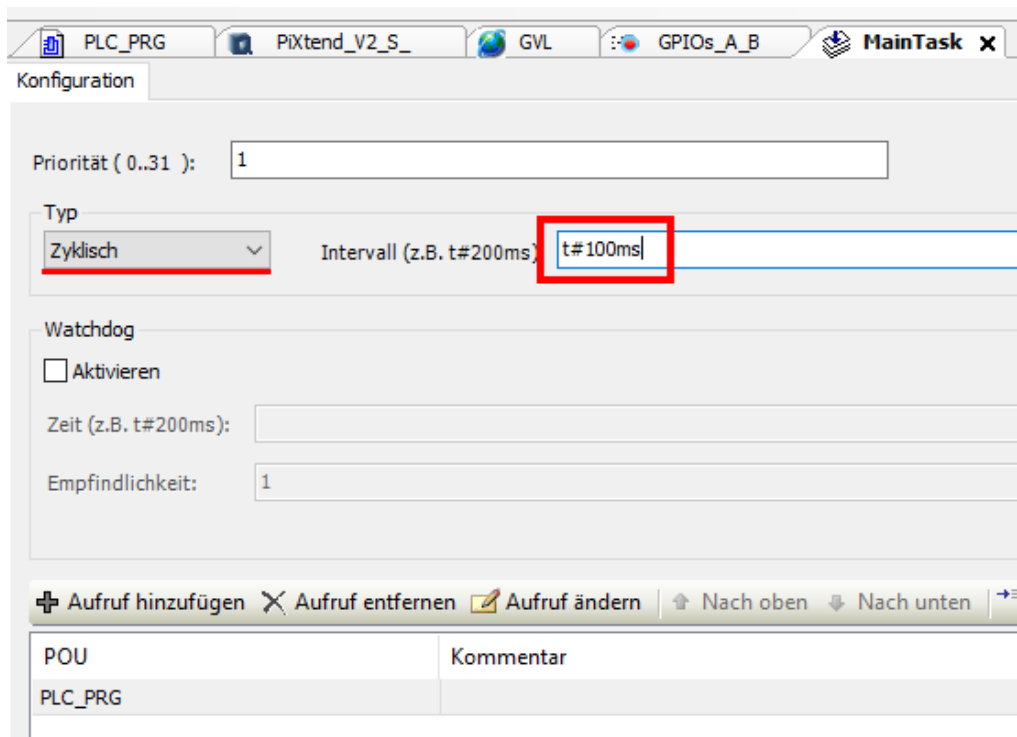


Figure 21: CODESYS - Task Configuration - Main Task

The settings made mean: In the “MainTask” task, the PLC_PRG program is executed every 100 ms and then runs sequentially (from top to bottom).

In our example the main program is named "PLC_PRG". It is created in the programming language "Continuous Function Chart", since this setting was selected when the CODESYS project was created.

Complex PLC programs are usually distributed over several POU (Program Organization Units) in order to improve the maintainability and clarity of the code.

For the time being we restrict ourselves to one program unit, namely PLC_PRG, which was automatically generated by CODESYS when the standard project was created.

Double-click on PLC_PRG to open the editor and begin entering the program code.

CFC is graphically programmed. First drag and drop three "input" blocks from the toolbox (right) and place them under each other in the workspace:

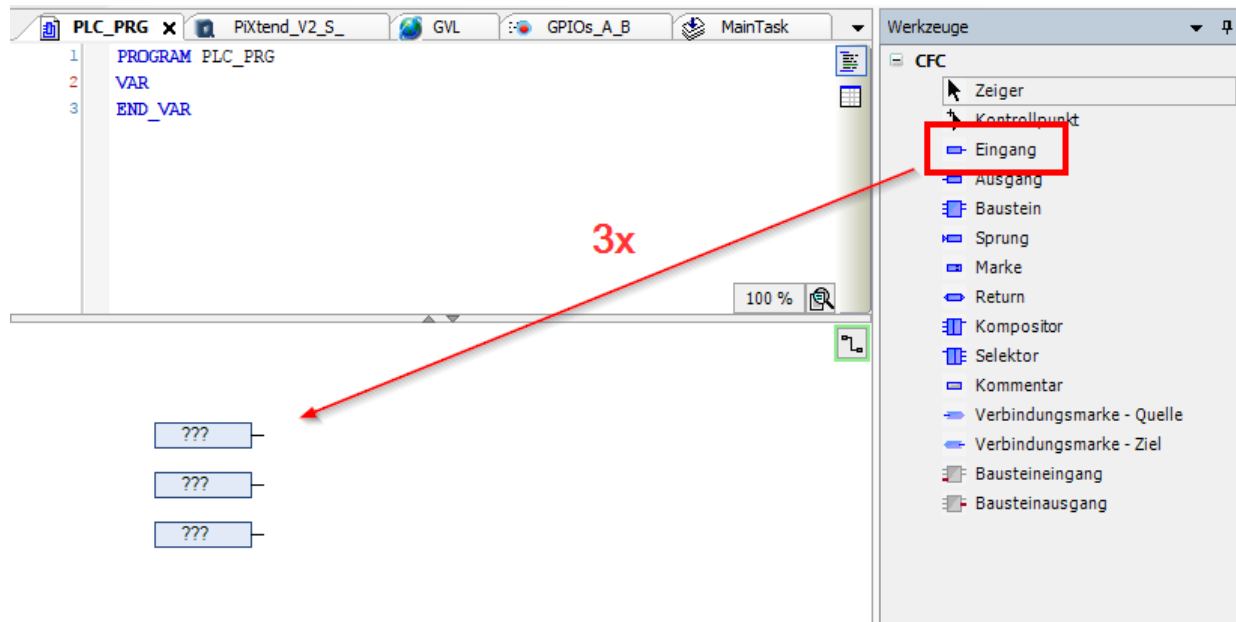


Figure 22: CODESYS - PLC_PRG (CFC) - Input

Repeat this with three “output” blocks and drag and drop a line from an input with to an output block.

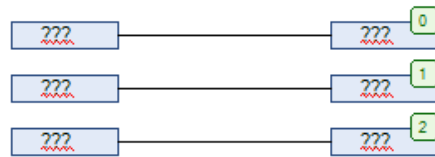
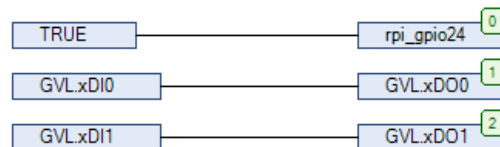


Figure 23: CODESYS - CFC - Three inputs connected to three inputs

Click in the middle of a block to assign variables or constants as shown in the figure. Click on the three points ... to open the previous Input assistance for variables. You can either type in the variable name and use the auto complete feature of CODESYS or select it from the list. When using the direct input, make sure to use the prefix “GVL”, the name of the Global Variable List.



You have just created your first program. Here is a brief explanation:

- The Raspberry Pi GPIO bit 24 is set to TRUE to enable SPI communication with the PiXtend V2 -S- micro-controller.
- Input GVL.xDI0 is written to output GVL.xDO0
- Input GVL.xDI1 is written to output GVL.xDO1

From the menu bar, click on Create → Compile to compile the program.

7.5.1.7 Connection to the PiXtend V2 -S- and program download

After the program has been compiled without errors, double-click on ""Device (CODESYS Control for Raspberry Pi)" in the project tree and then on the "Scan Network" button in the "Communication" tab.

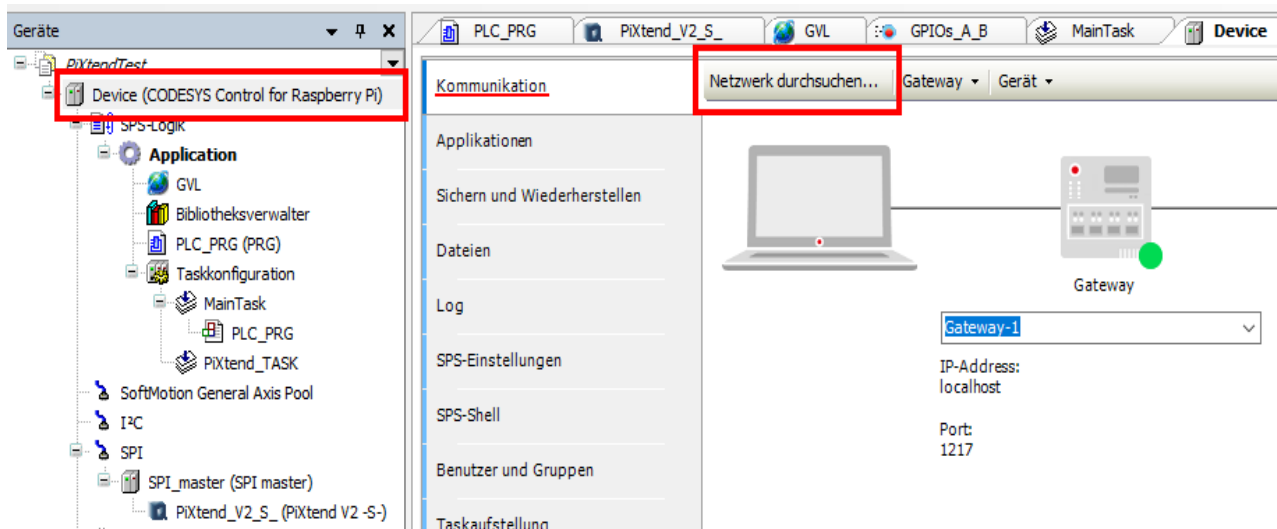


Figure 25: CODESYS - Communication - Scan Network

CODESYS is now searching in your local network for a Raspberry Pi on which the CODESYS Runtime extension is running.

If no device is found, check the following:

- Is the correct SD card with the image for the CODESYS runtime extension inserted into the device? Pre-installed images are available in the download section (www.pixtend.de). To create your own CODESYS image for the Raspberry Pi, please refer to the PDF guide from the CODESYS Store on the download section of the CODESYS Control for Raspberry Pi SL.
- Is the Raspberry Pi switched on and does it have a valid IP that you can ping from your PC? (via the ping command from the Windows command line)
- The IP of your Raspberry Pi can be found with the command "ifconfig" in the Raspberry Pi command line. If you do not know the IP, you will need a screen and keyboard to check directly on the Raspberry Pi.
- If you see a valid IP here, but the ping is not successful, check the network connection to your Raspberry Pi.
- If the Raspberry Pi has been switched on for more than two hours, the CODESYS Runtime extension will automatically terminate and the Raspberry Pi must be restarted. You can execute the command conveniently via the Linux console:

```
sudo shutdown -r now
```

If the Raspberry Pi has been found and selected, click on Online → Login on the main menu and download the program to the Raspberry Pi.

As soon as it is online, you can switch to the “Run Mode” with the F5 key and see the live values in the PiXtend V2 -S- I/O Mapping tab. Click PiXtend V2 -S- in the device tree and select the “SPI devices I/O Mapping” tab.

Under State → “Firmware” and “Hardware” you will see the firmware version of the micro-controller and the revision of the hardware:

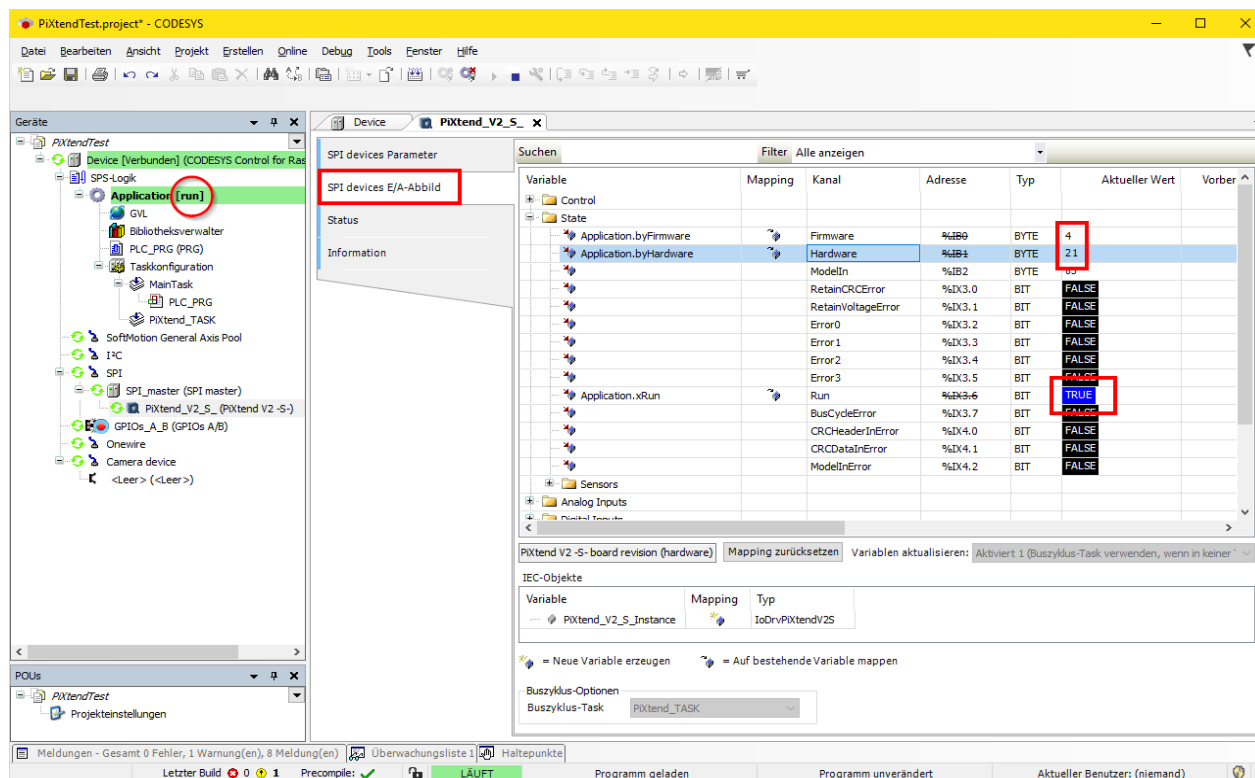


Figure 26: CODESYS - PiXtend V2 -S- I/O mapping online

This is a PiXtend V2 -S- board revision 1 (hardware = 21) and the micro-controller firmware version 4. If you have a revised PiXtend V2, for example, after an improvement or extension, the versions can differ.

The “Run” channel should be “True” for normal operation. This means that the micro-controller on PiXtend V2 is in “Run Mode”.

7.5.1.8 Further steps

Test the functionality of your program by applying a HIGH level (24V) to the digital input 0 and 1 (for example by means of a button, switch, or a wire jumper). Please observe the instructions for installation and the technical data sheet in the hardware manual.

If everything works, you can modify your PLC_PRG program as you wish. Try the following:

Add a module from the toolbar and assign it the type "AND" by clicking on the three question marks (???) and typing AND.

Connect the two inputs of the AND block with GVL.xDI0 and GVL.xDI1. The output of the AND block is therefore only TRUE if both inputs are simultaneously TRUE.

To make things a bit more interesting, add a "Timer On" block "TON" and give it an instance name (here "timer_delay"). The timer expects a time (here T#2s - two seconds) and an input signal.

Connect the IN signal to the output of the AND block. The output Q is assigned to the digital output GVL.xDO1.

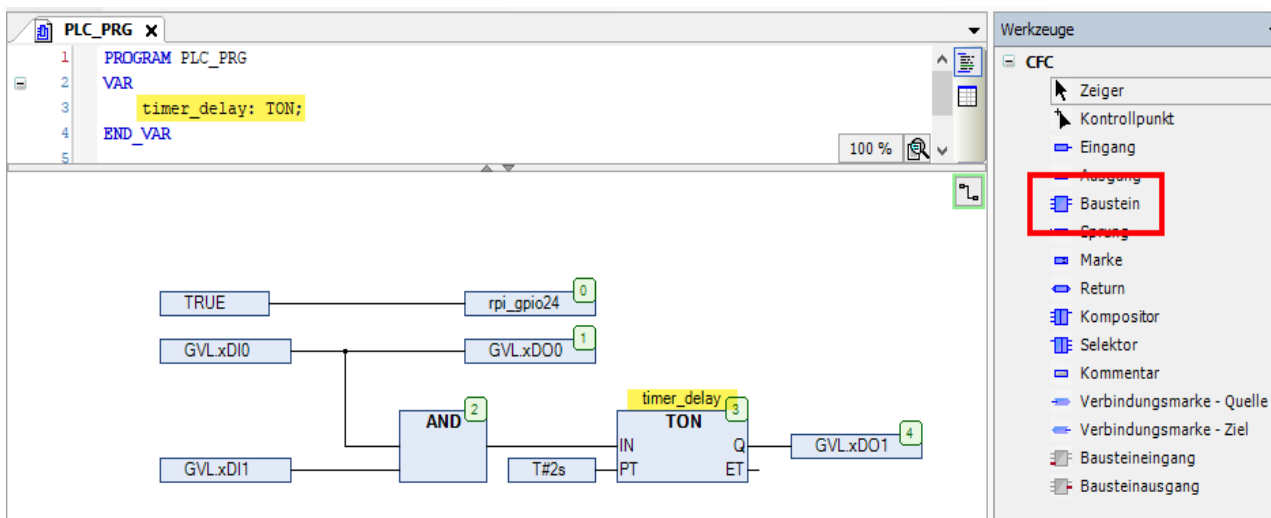


Figure 27: CODESYS - PLC_PRG (CFC) - Demo Program

A high level on both inputs GVL.xDI0 and GVL.xDI1 for at least two seconds thus results in the output GVL.xDO1 being set. If only one input is HIGH, the output remains LOW. Similarly, the output remains LOW as long as both inputs are HIGH and the two seconds of the timer have not elapsed.

7.5.2. Step by step to create your first CODESYS Webvisu

We now want to add a visualization to the project so that you can monitor your control from any PC (also smartphone / tablet). Right-click in the project tree on the Application → Add Object → Visualization.

CODESYS now automatically creates the “Visualization Manager” and a empty visualization called “Visualization”.

In the manager, parameters for the Webvisu can be set, e.g., the name of the visualization and preferred resolution. We will leave the default settings.

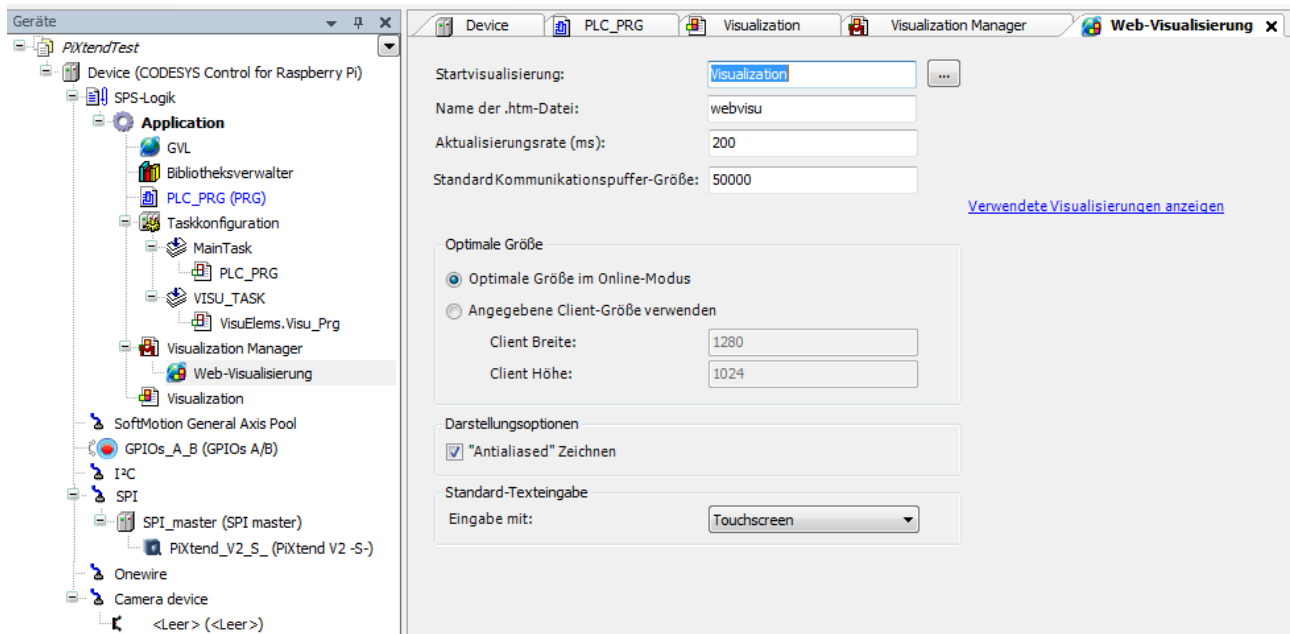


Figure 28: CODESYS - Web Visualization - Configuration

Double-click on “Visualization” to open the editor for the visualization. CODESYS already has a number of pre-made controls elements. Open the group Switches/Lamps/Images in the toolbar and place four lamps in the workspace.

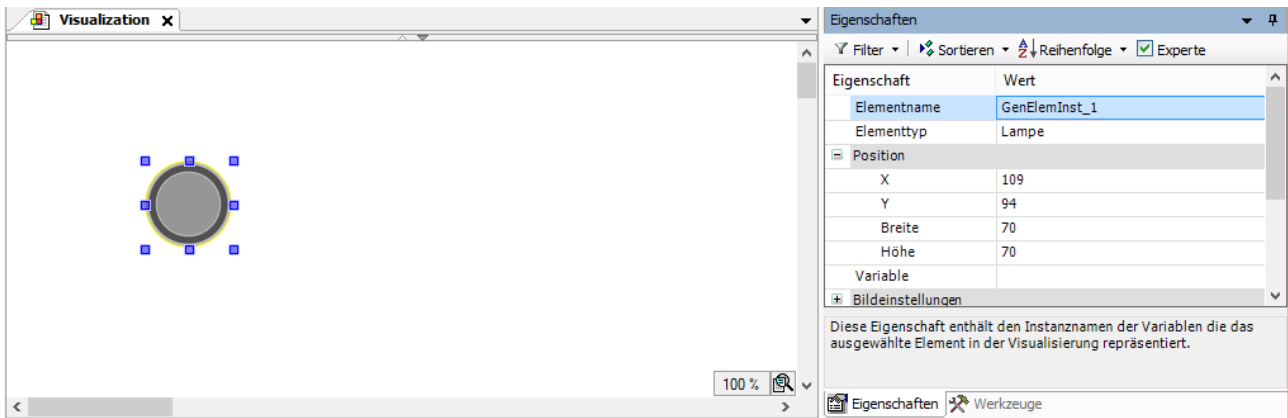


Figure 29: CODESYS - Visualization

Add four labels and assign the variables to the lamps under the “Variable” property (xDI0, xDI1, xDO0, xDO1).

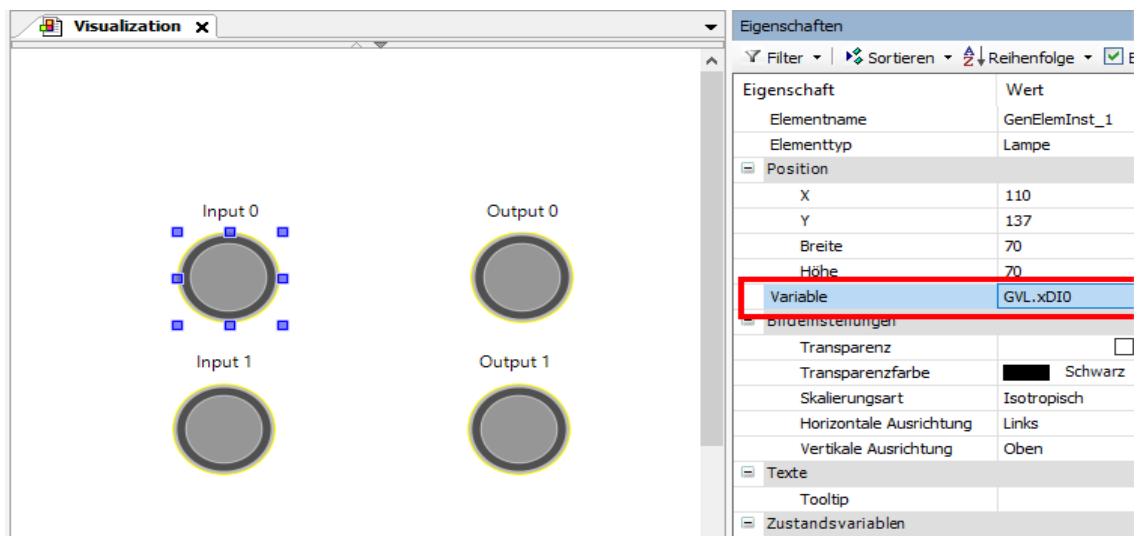


Figure 30: CODESYS - Visualization - Element properties

Labels are always static. To display contents of variables, CODESYS uses text fields (rectangles).

Drag two new rectangles into the workspace, enter “Firmware: %s” for the first one and “Hardware: %s” for the second one as the “Text” property. This creates placeholders for two-byte variables.

Check the box next to “Expert”. Now insert the two variables for firmware version and hardware revision, byFirmware and byHardware, under “Text variables” for the respective rectangle.

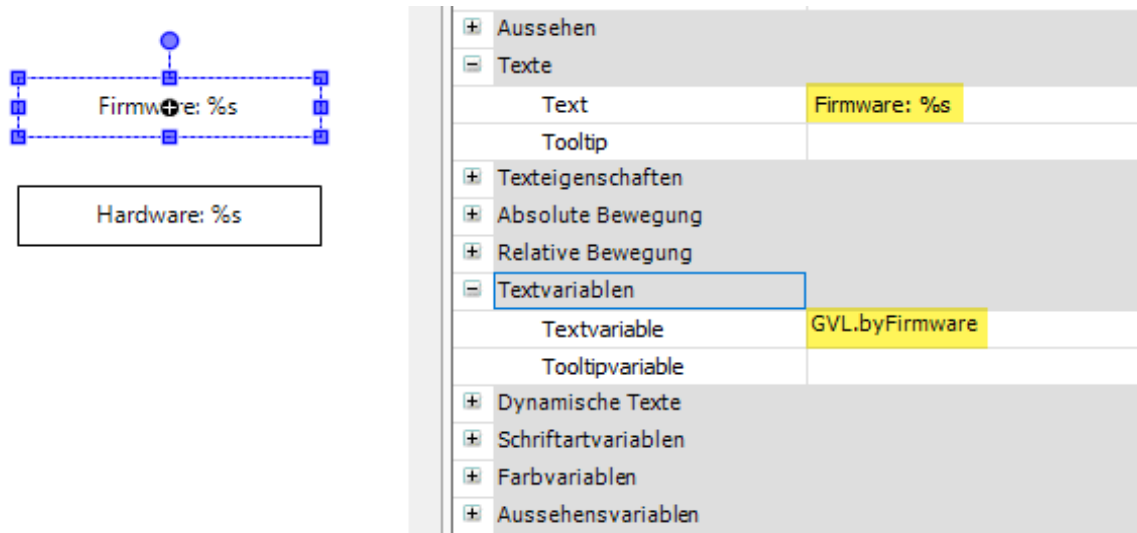


Figure 31: CODESYS - Visualization - Configuration of a rectangle

In live mode, the two placeholders %s are filled with the values from the variables. Further placeholders are %d for numbers, %f for floating point numbers (REAL, LREAL). Formatting options as in the C programming language are possible.

Finally, we add a headline and an animated CODESYS logo (special controls - waiting symbol cube) to visually enhance the whole thing:

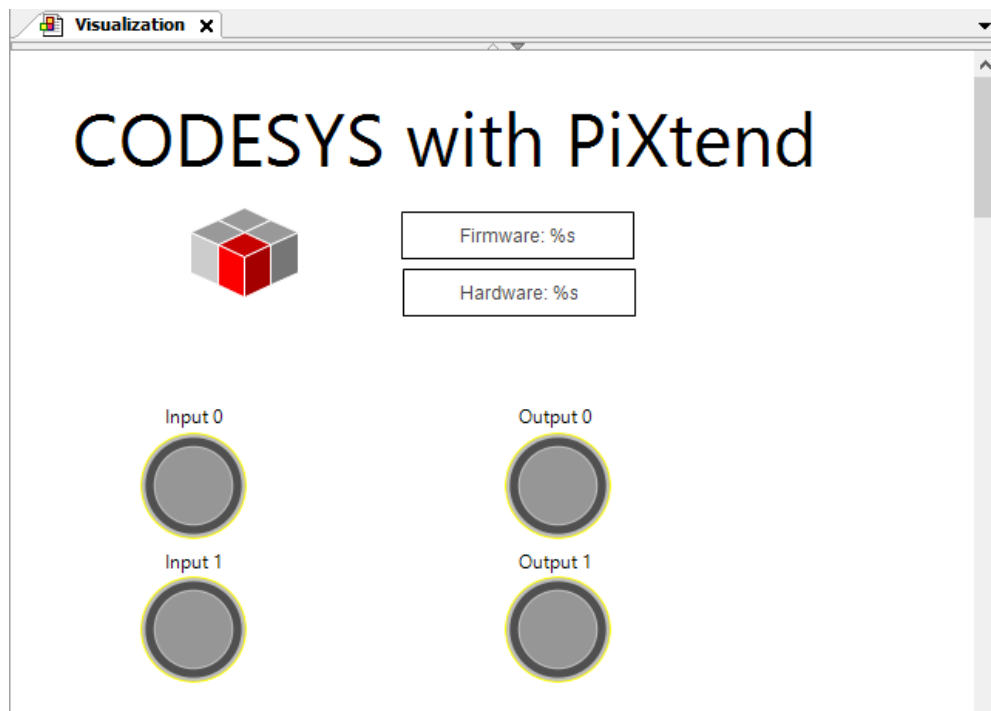


Figure 32: CODESYS - Visualization - Completed

Recompile the project Create → Compile and perform a complete download.

Open a browser of your choice on your PC or smartphone/tablet and enter the IP of your Raspberry Pi, followed by ":8080/webvisu.htm", for example

<http://192.168.1.99:8080/webvisu.htm>

7.6. Step by step to create your first DAC program

7.6.1. Create CODESYS standard project for PiXtend

→ Start CODESYS.

Create a new project by clicking on File → New Project in the main menu (shortcut Ctrl+N)

Select “Standard Project” from the category “Projects” give the project a name (here “PiXtendDACTest”) and confirm with “OK”.

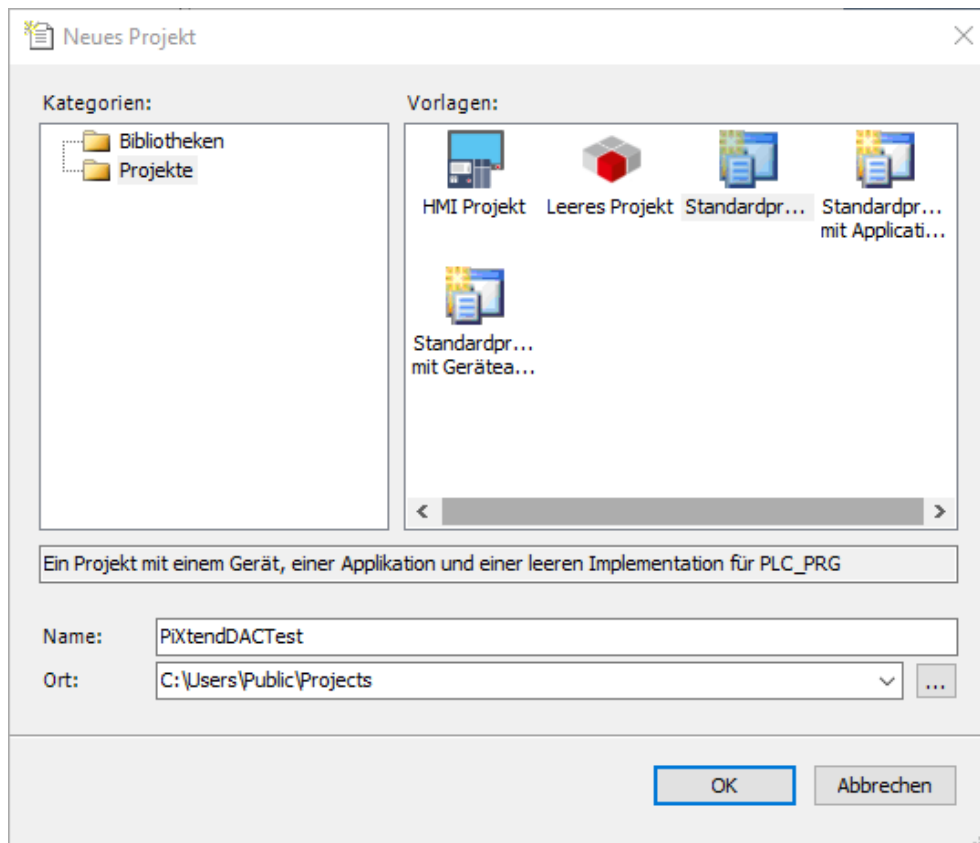


Figure 33: CODESYS - DAC - New project

Select “CODESYS Control for Raspberry Pi” as the device and select “Continuous Function Chart (CFC)” as the programming language for the main program PLC_PRG.

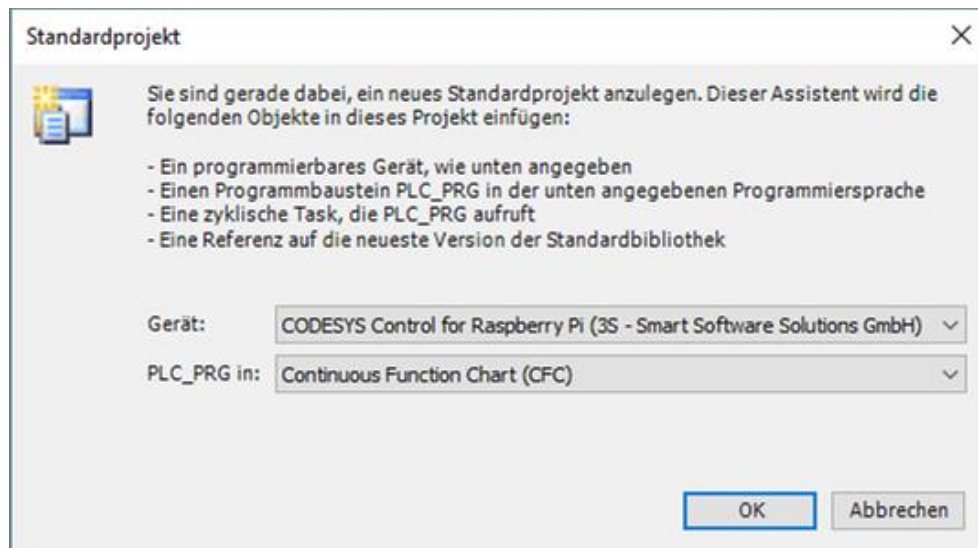


Figure 34: CODESYS - DAC - Selecting a device

After CODESYS has created the standard project, the project has the following structure:

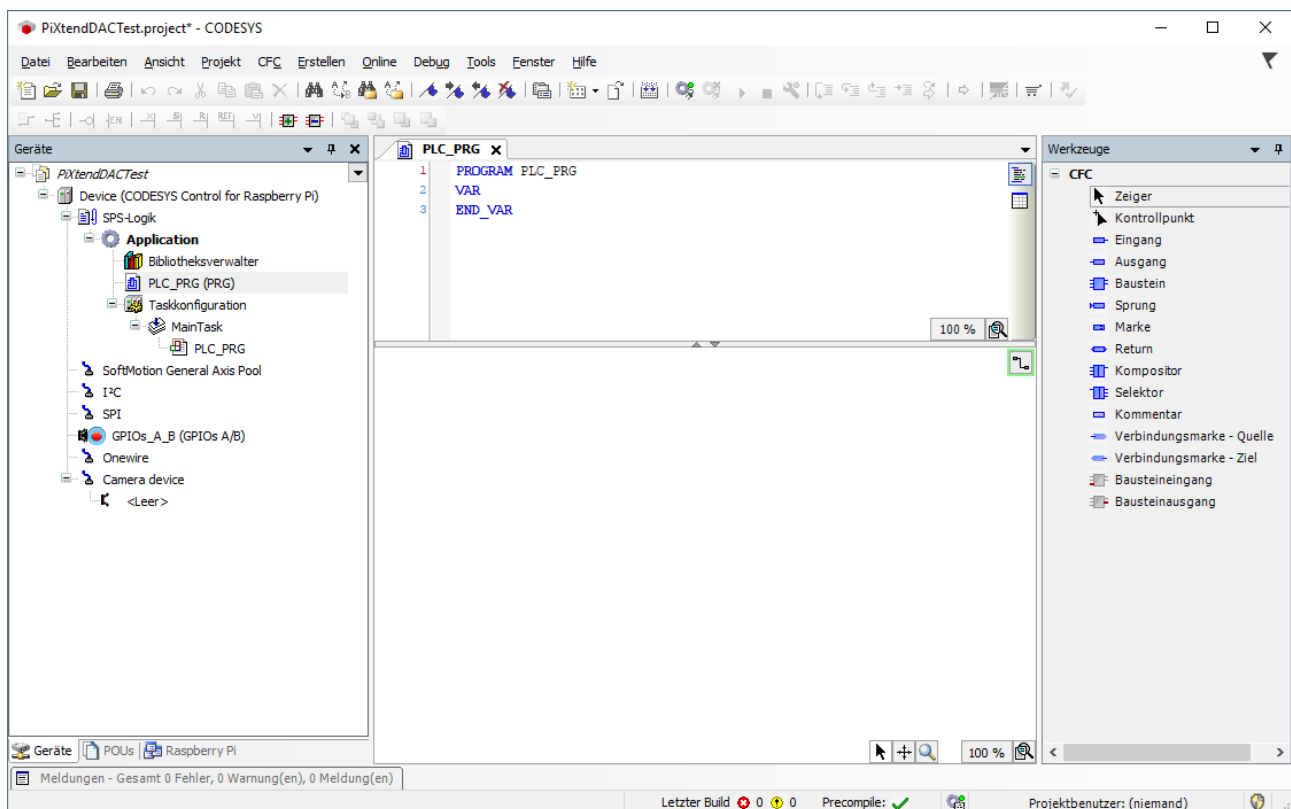


Figure 35: CODESYS - DAC - Project overview

7.6.2. Add SPI device

In the project tree right, click on the entry “SPI” and select “Add device”:

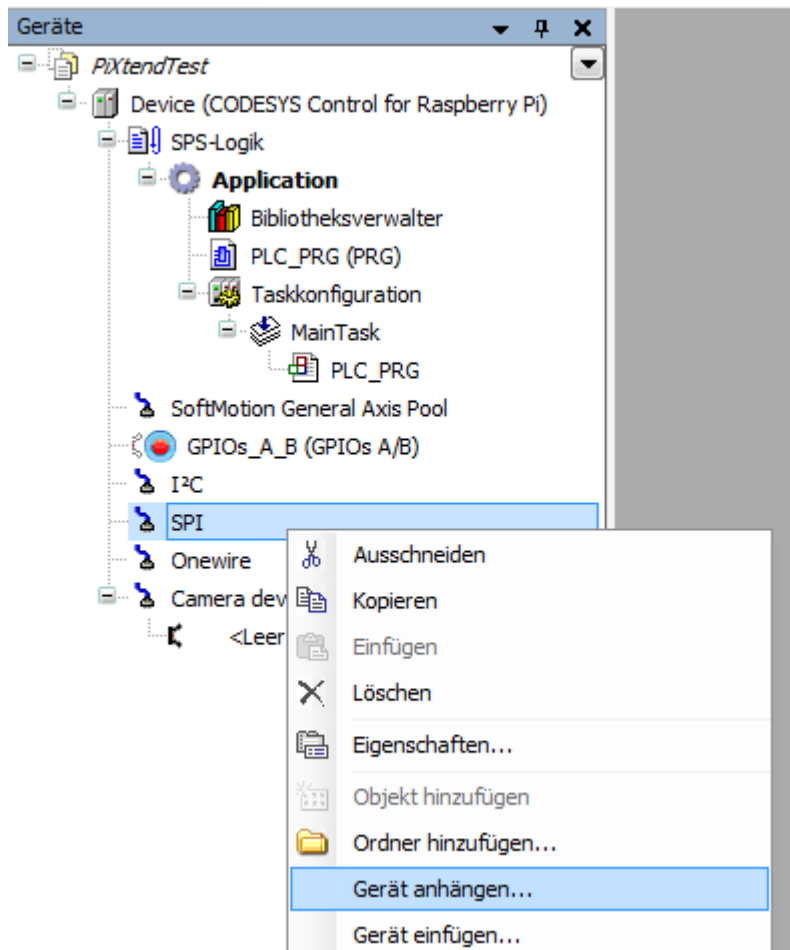


Figure 36: CODESYS - DAC - Adding SPI device

Select "SPI master" as the device and click on "Add device".

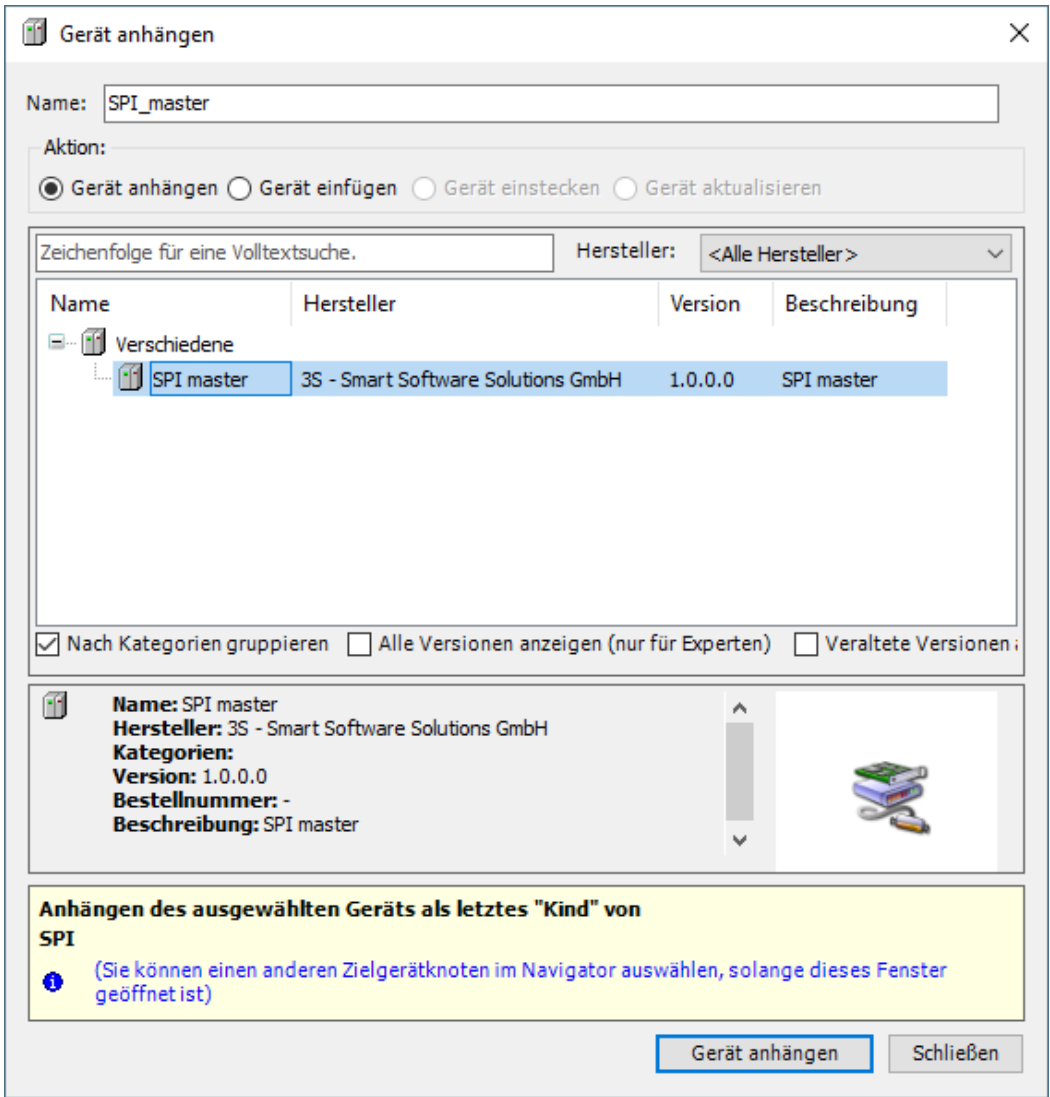


Figure 37: CODESYS - DAC - Adding SPI master

There is now a new entry “SPI_master (SPI Master)” in the project tree under SPI.

Since the PiXtend V2 DAC is connected to the Raspberry Pi via SPI port 0.1, you have to adjust the parameters of the SPI master. Double-click on the entry “SPI_master” and change the SPI port to “/dev/spidev0.1”:

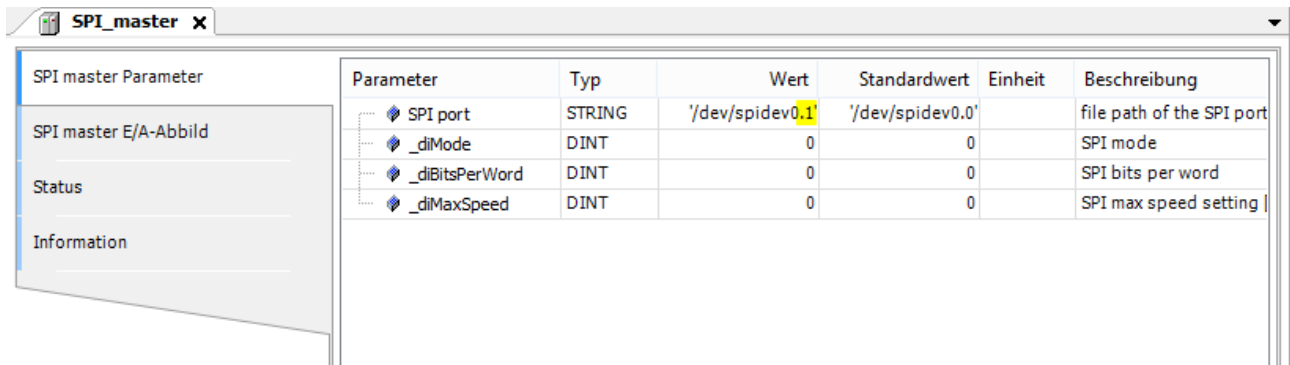


Figure 38: CODESYS - DAC - Configuring SPI master

NOTICE

The “SPI port” must be set to the device path

/dev/spidev0.1; otherwise, there is no communication with the DAC on the PiXtend V2 board.

7.6.3. Creating Global Variable List

Click on the “Application” entry on the right in the project tree and use “Add Object” -> “Global Variable List” to add a new Global Variable List with the name “GVL”.

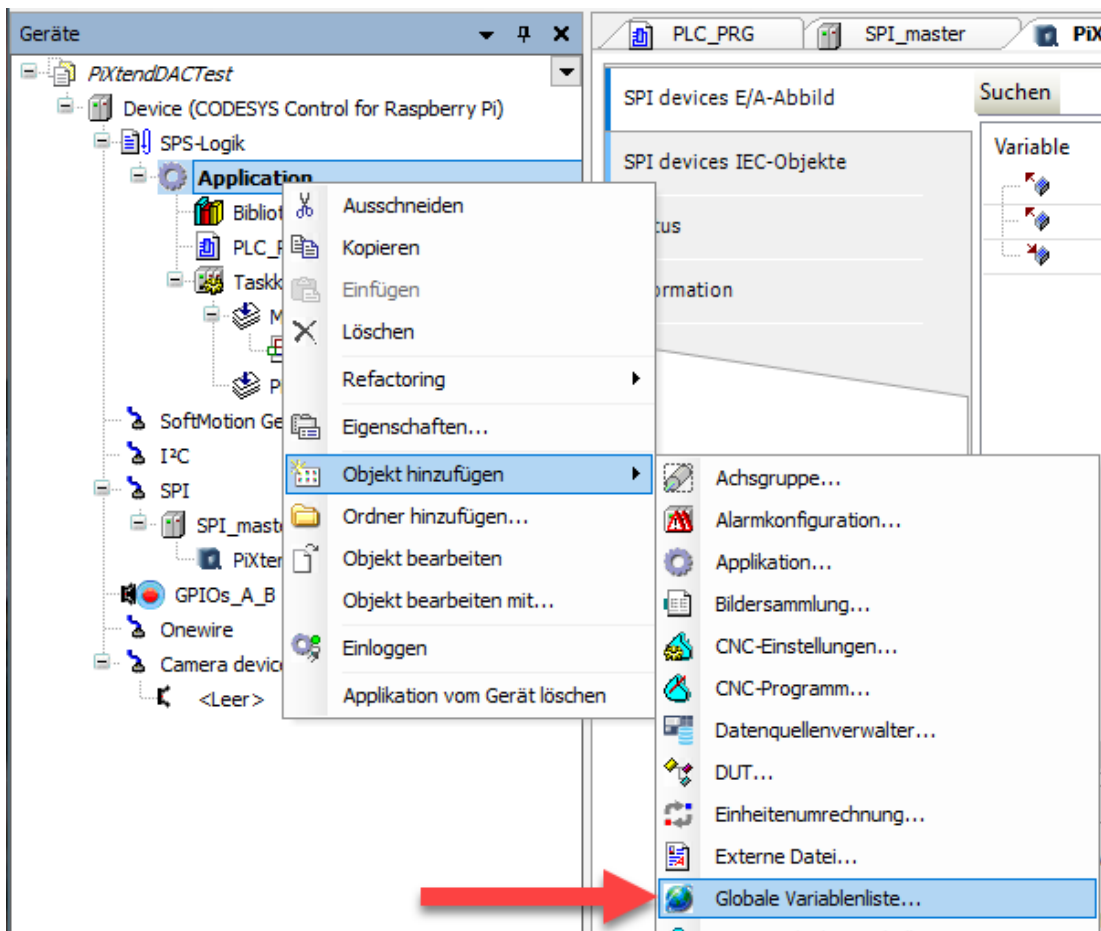


Figure 41: CODESYS - DAC - Adding Global Variable List

Open the GVL and add the following entries:

```
//Analog Outputs
rAOut0: REAL;
rAOut1: REAL;
```

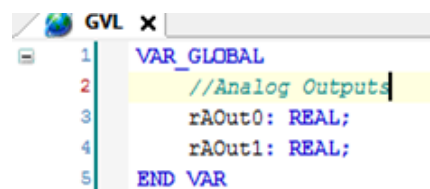


Figure 42: CODESYS - DAC - GVL Variables

7.6.4. Mapping variables

After you have created the required variables in the GVL, they must be “mapped” to the corresponding outputs of the PiXtend V2 - S- DAC (here in our example for a PiXtend V2 -S- board).

Open the previous “SPI devices I/O Mapping” tab by double-clicking on the “PiXtend_V2_S_DAC” device. Assign the two variables to the outputs

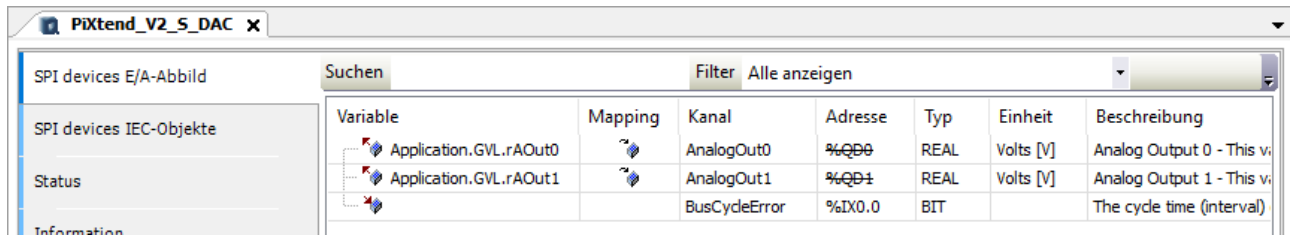


Figure 43: CODESYS - DAC - Variable mapping

by double-clicking on the still empty left column. Three dots appear (...), click on them to open the Input assistance and select the desired variable.

Select the variable Application.GVL.rAOut0 for AnalogOut0

Select the variable Application.GVL.rAOut1 for AnalogOut1

Lastly, the “GPIO bit 24 of the Raspberry Pi must be configured as an output. Open the Raspberry Pi GPIO configuration by double clicking on “GPIOs_A_B” in the project tree. Select “Output” for “GPIO24”.

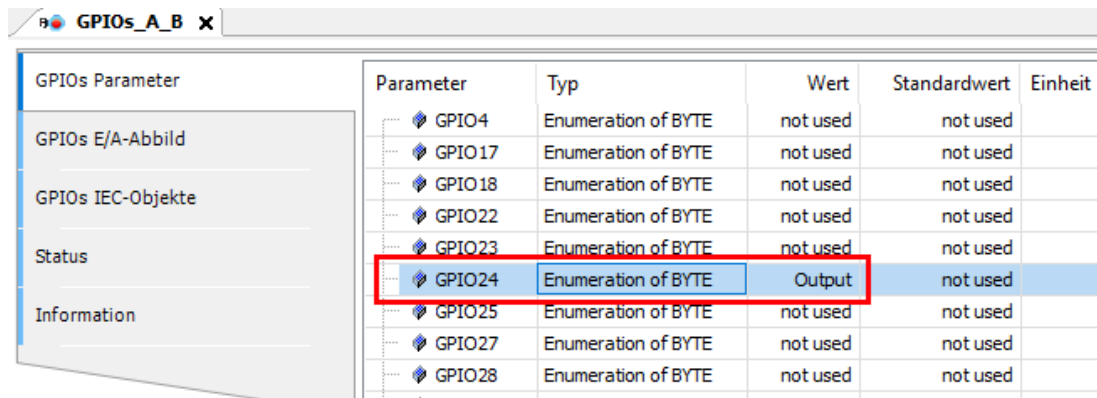


Figure 44: CODESYS - DAC - Using GPIO 24 as an output

Then go to the “GPIO I/O Mapping” tab and click on “Digital Output”. Enter “rpi_gpio24” as a variable for bit 24.

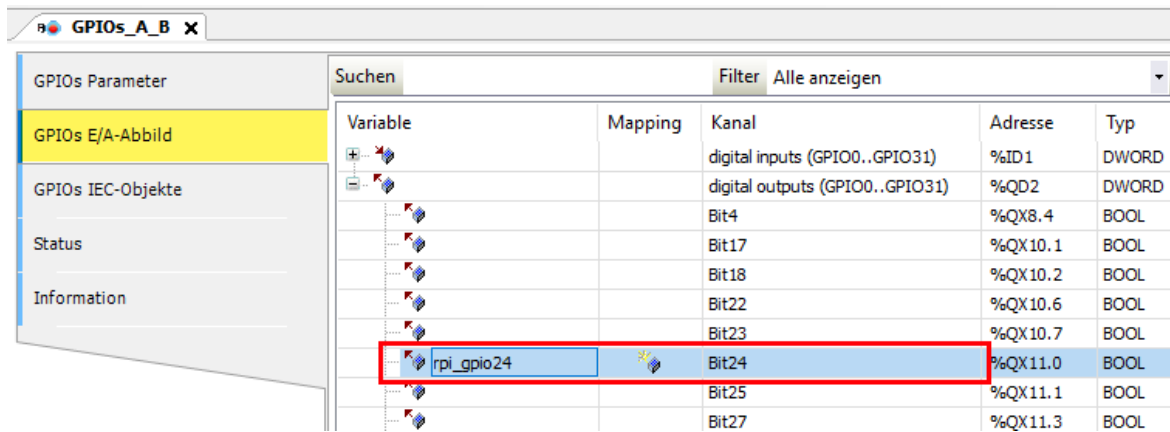


Figure 45: CODESYS - DAC - GPIO 24

NOTICE

The “GPIO 24” must always be set to “TRUE” in the program to enable communication with the PiXtend V2 DAC. First set the variable “rpi_gpio24” in your program to “TRUE” so that this is not forgotten later.

7.6.5. PiXtend V2 Add DAC device

Select the “SPI_master” entry in the project tree and add another device (right click>Add device).

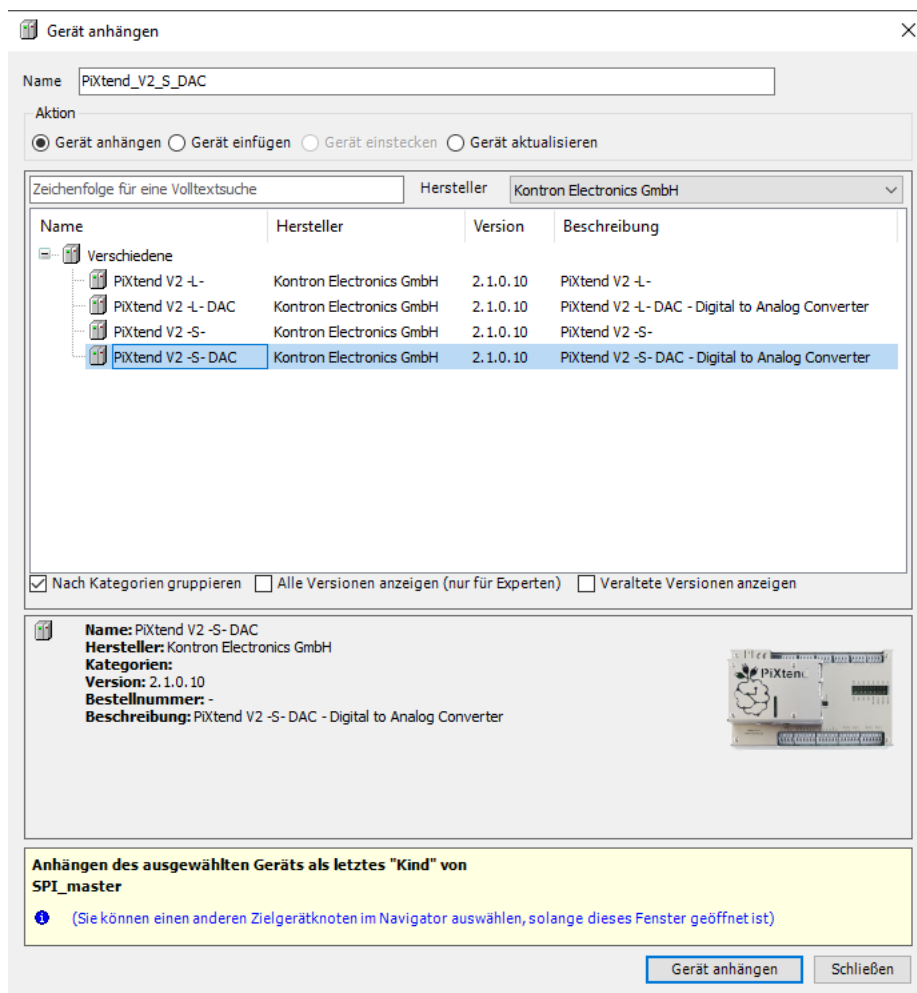


Figure 39: CODESYS - DAC - Adding PiXtend V2 DAC

Select “Kontron Electronics GmbH” in the device manufacturer drop-down menu. Then select “PiXtend V2 -S- DAC” as the device, for a PiXtend V2 -S- board or “PiXtend V2 -L- DAC” for a PiXtend V2 -L- board (not “PiXtend V2 -S-” or “PiXtend V2 -L-”), click on the button "Add device" in the lower right corner and close the window.

Now the PiXtend V2 -S- DAC or PiXtend V2 -L- DAC appears as a device under “SPI_master (SPI Master)”.

Double-click on the "PiXtend_V2_S_DAC" (in our example, the name may differ for the PiXtend V2 -L- DAC) device in the project tree to open the configuration page for the device. The “SPI devices I/O Mapping” tab shows the two analog outputs that are available for the PiXtend V2 for use in CODESYS.

So that this process image is exchanged cyclically, two changes are still necessary. Select the entry “Enabled 1 (use bus cycle task if not used in any task)” for “Update variables” and “PiXtend_TASK” for the bus cycle task if this is not yet the case.

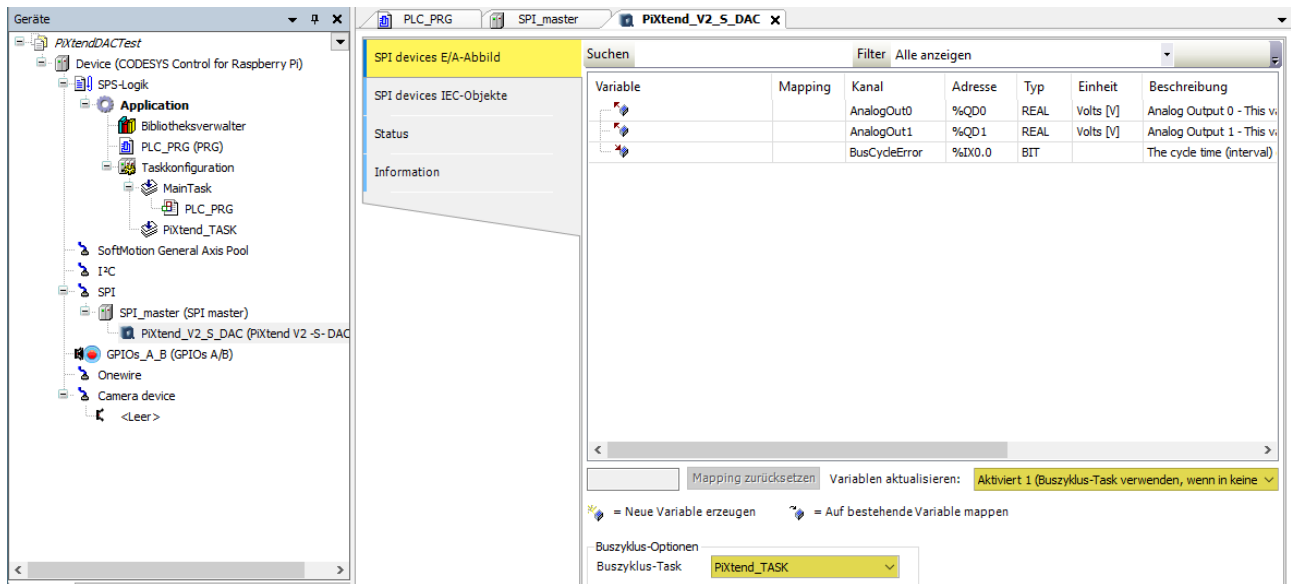


Figure 40: CODESYS - DAC - I/O mapping settings

Later during operation (“Online control”), you can directly assign values to the outputs in this window (using F7 “forcing” values).

However, since we want to access the outputs from the visualization, we will create a Global Variables List (GVL) so we can assign variables to the outputs.

7.6.6. Task configuration

If desired, you can adjust the cycle time of the PiXtend_TASK. Click on the task configuration → PiXtend_TASK. In principle, the cycle time can be reduced to t#1ms, for this test project the default setting of 30 ms is sufficient. If you want to control the PiXtend V2 board yourself later, this value should remain unchanged.

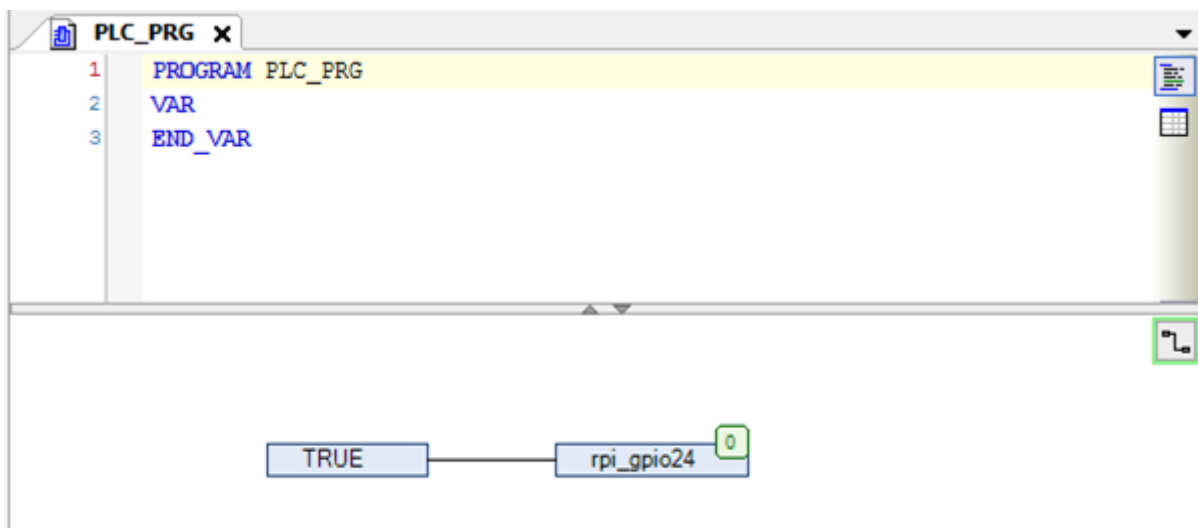
7.6.7. Creating the main program

Double-click on PLC_PRG to open the editor and begin entering the program code.

CFC is graphically programmed. First drag and drop one “input” blocks from the toolbox (right) and place it in the workspace:

Then add an “output” block and connect the two together.

Click in the middle of a block to assign variables or constants as shown in the figure:



Figure

Figure 46: CODESYS - DAC - Setting variable rpi_gpio24

From the menu bar, click on Create → Compile to compile the program.

7.6.8. Connection to the PiXtend V2 and program download

After the program has been compiled without errors, double-click on “Device (CODESYS Control for Raspberry Pi)” in the project tree and then on the “Scan Network” button.

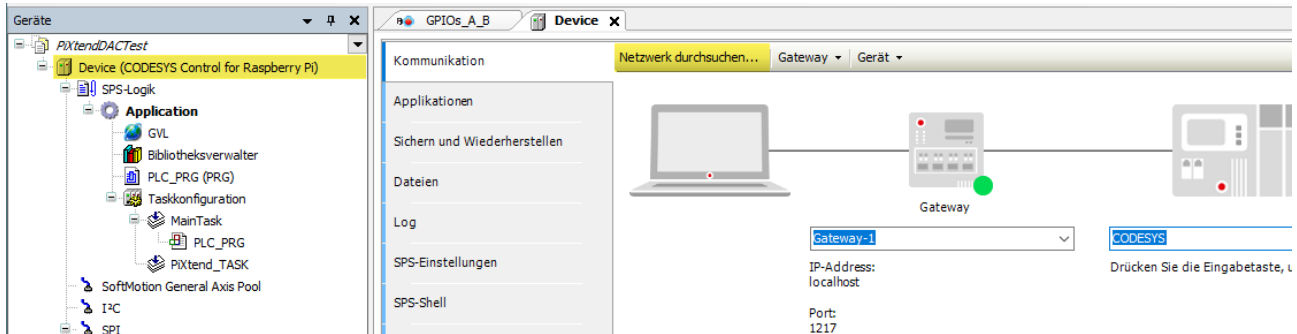


Figure 47: CODESYS - DAC - Scan Network

CODESYS is now searching in your local network for a Raspberry Pi on which the CODESYS Runtime extension is running. If no device is found, check the following:

- Is the correct SD card with the image for the CODESYS runtime extension inserted into the device? **Pre-installed** images are available in the download section (www.pixtend.de). To create your own CODESYS image for the Raspberry Pi please read the corresponding chapter in the CODESYS online help, section “Add-ons”.
- Is the Raspberry Pi switched on and does it have a valid IP address that you can ping from your PC? (via the ping command from the Windows command line)
- The IP address of your Raspberry Pi can be found with the command “ifconfig” in the Raspberry Pi command line when using a keyboard and screen.
- If you see a valid IP here, but the ping is not successful, check the network connection to your Raspberry Pi.
- If the Raspberry Pi has been switched on for more than two hours, the CODESYS Runtime extension will automatically terminate, and the Raspberry Pi must be restarted. (sudo reboot)

If the Raspberry Pi has been found and selected, click on “Online → Login on the main menu and download the program to the Raspberry Pi.

7.6.9. Creating the CODESYS Webvisu

We now want to add a simple visualization to the project.

Right-click in the project tree on “Application → Add object → Visualization”.

CODESYS now automatically creates the “Visualization Manager” and an empty visualization called “Visualization”.

In the manager, parameters for the Webvisu can be set, e.g., the name of the visualization and preferred resolution.

Change the scaling options to the value “Isotropic”:

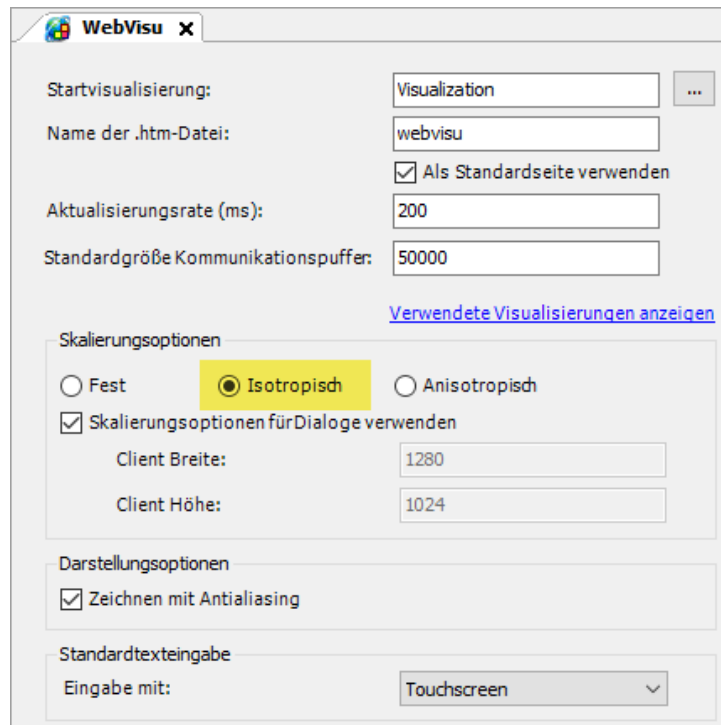


Figure 48: CODESYS - DAC - WebVisu settings

Double-click on “Visualization” to open the editor for the visualization. CODESYS already has a number of pre-made controls elements.

Open the “General controls” group in the toolbar and place two “sliders” in the workspace:

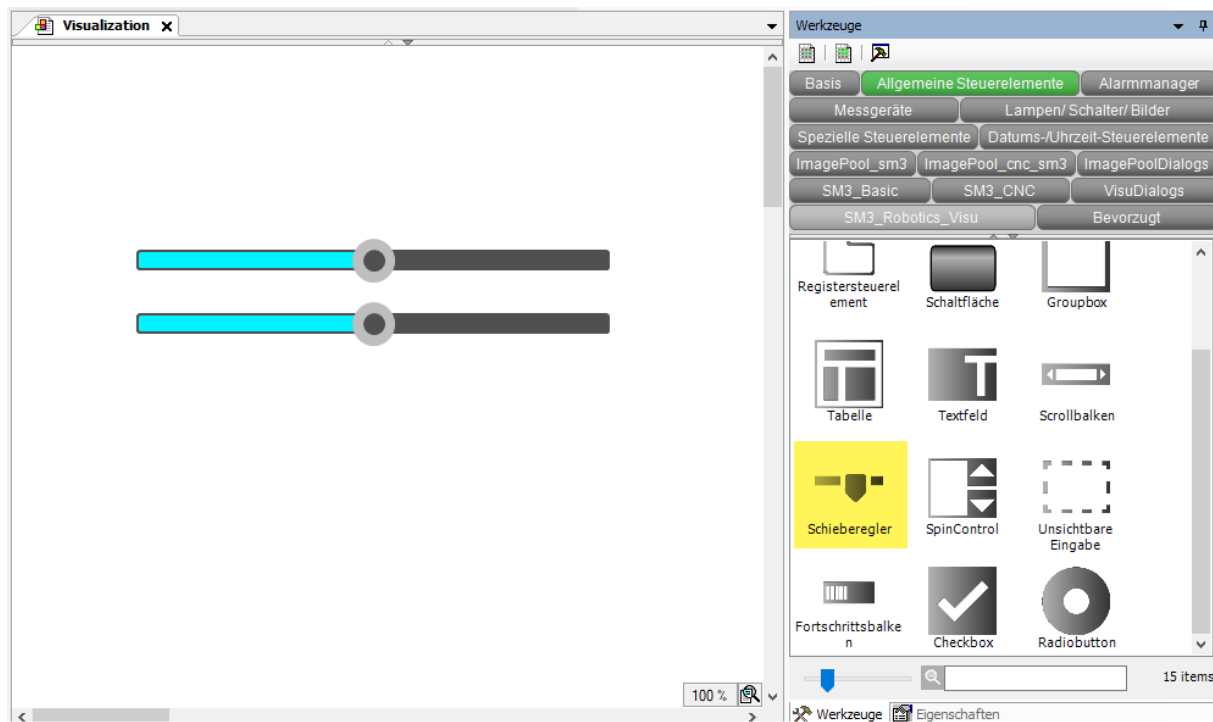


Figure 49: CODESYS - DAC - WebVisu - Sliders

Add two labels and assign the respective variables to the sliders under "Variable" properties (GVL.rAOut0, GVLrAOut1). Change the scale end of both sliders to “10”:

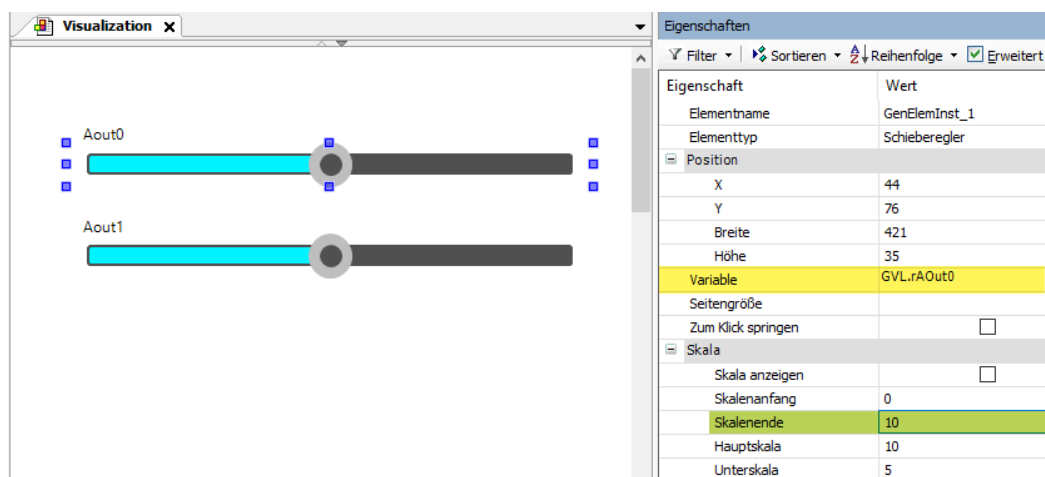


Figure 50: CODESYS - DAC - Slider - Setting scale end

Now add two text fields (rectangles) and enter “%2.1f” for the “Text” property. This creates a placeholder for a REAL variable with two places before and one place after the decimal point.

Assign the value “GVL.rAOut0” or “GVLrAOut1” to the “Text variable” property.

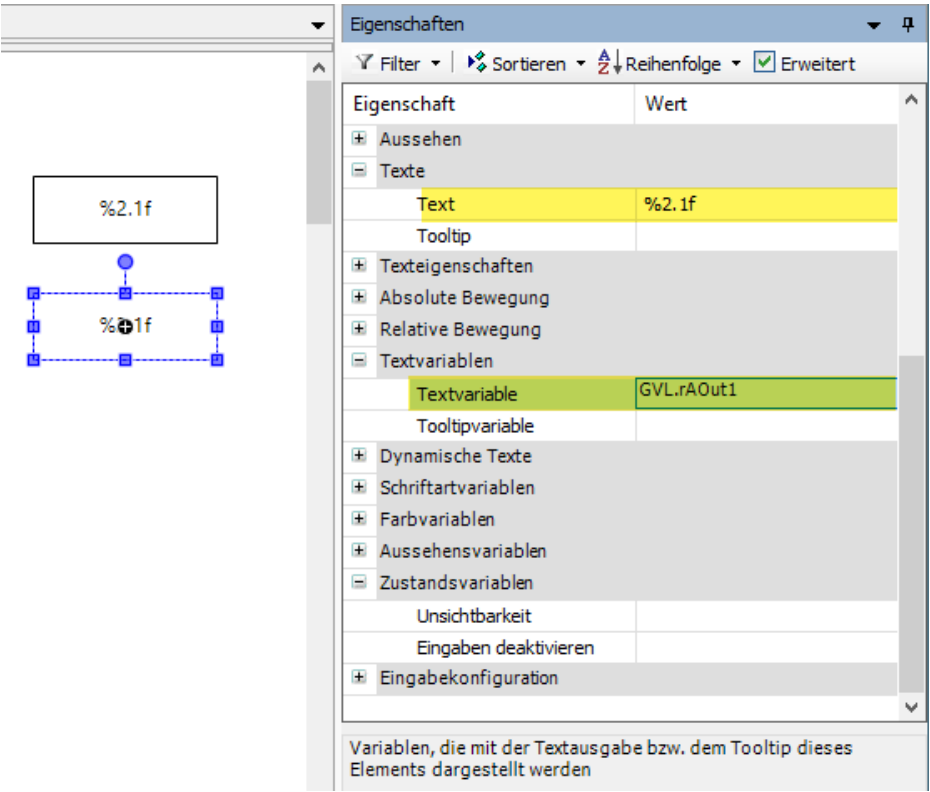


Figure 51: CODESYS - DAC - WebVisu - Text field settings

Finally, we add a headline and an animated CODESYS logo (special controls - waiting symbol cube) to visually enhance it.

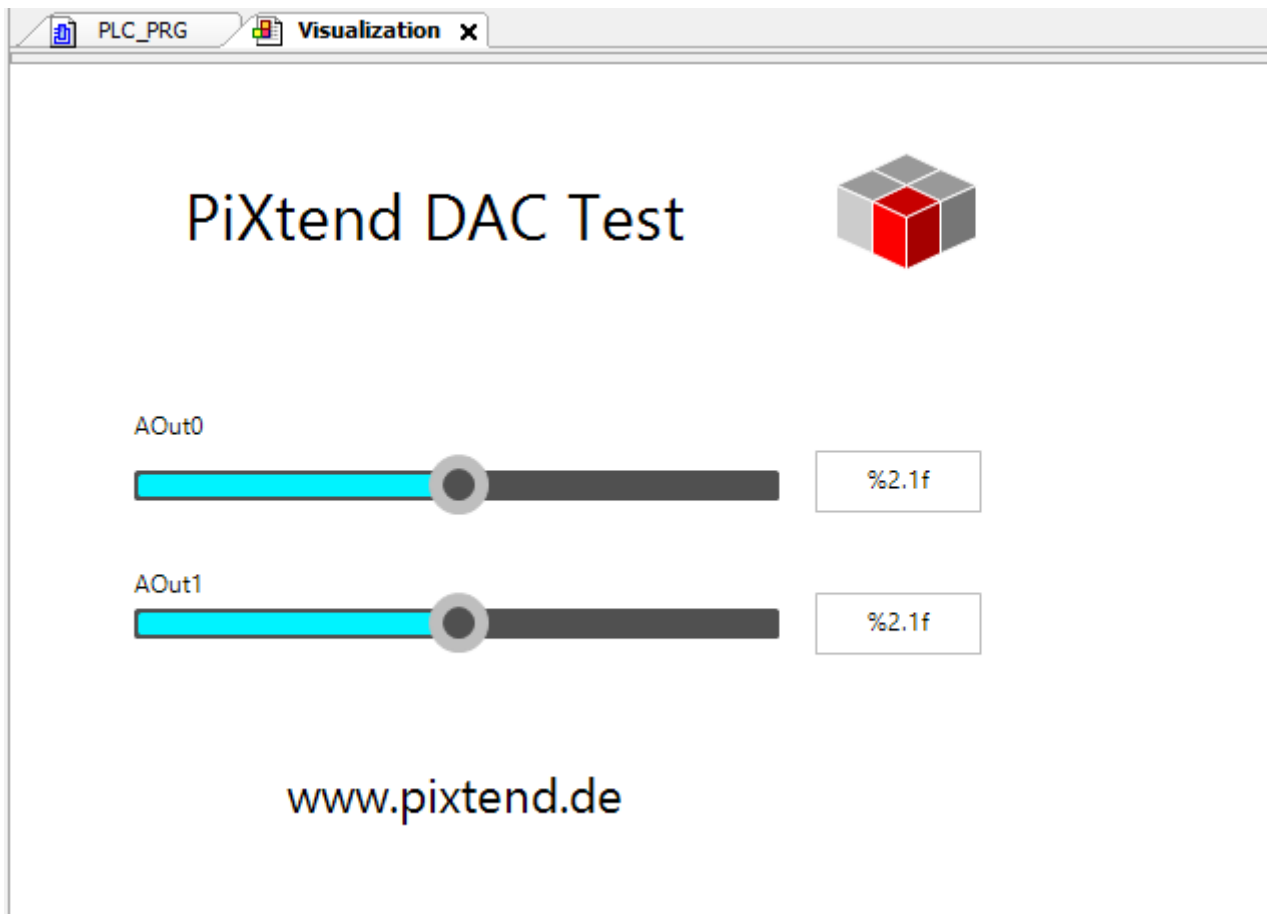


Figure 52: CODESYS - DAC - WebVisu - Completed

Recompile the project Create → Compile and perform a complete download.

Open a browser of your choice on your PC or smartphone/tablet and enter the IP address of your Raspberry Pi, followed by ":8080/webvisu.htm", for example

<http://192.168.137.99:8080/webvisu.htm>.

Now move the sliders to change the voltage of the two analog outputs.

NOTICE

If the PiXtend V2 board is addressed parallel to the PiXtend V2 DAC, make sure that you add the PiXtend V2 device to its own (additional) SPI master and not to the SPI master where the DAC is already located.

Example for the PiXtend V2 -S- (all specifications also apply to the PiXtend V2 -L-):

| Parameter | Typ | Wert |
|----------------|--------|------------------|
| SPI port | STRING | '/dev/spidev0.1' |
| _diMode | DINT | 0 |
| _diBitsPerWord | DINT | 0 |
| _diMaxSpeed | DINT | 0 |

| Parameter | Typ | Wert |
|----------------|--------|------------------|
| SPI port | STRING | '/dev/spidev0.0' |
| _diMode | DINT | 0 |
| _diBitsPerWord | DINT | 0 |
| _diMaxSpeed | DINT | 0 |

Figure 53: CODESYS - Information on how to use both PiXtend V2 SPI devices

The PiXtend V2 -S- DAC is added to the first SPI Master (SPI_Master, highlighted in green), in our example we have added this device first to the SPI bus. The PiXtend V2 -S- DAC device is highlighted in yellow. Note the SPI port setting for the DAC. The communication uses the second chip select of the Raspberry Pi, therefore the device "/dev/spidev0.1" (highlighted yellow on the right side) must be specified.

The PiXtend V2 -S- board itself must be controlled via another, second SPI master. In the picture above, the name "SPI_master_1" is highlighted in red. The PiXtend V2 -S- can be added to this second SPI master (also highlighted in yellow) and operated. The communication uses the first chip select of the Raspberry Pi, therefore the device "/dev/spidev0.0" (highlighted yellow on the lower right side) must be specified.

Of course, you can also add the PiXtend V2 -S- to the first SPI master and the DAC to the second SPI master, but remember to adjust the SPI port setting accordingly for each device.

7.7. CODESYS serial communication

This chapter describes all the steps necessary to establish communication between the Raspberry Pi (CODESYS) and a Windows PC (terminal) via the PiXtend V2 RS232 interface.

7.7.1. Requirements

The steps in this chapter assume that you have read the previous chapters, in particular Chapter 7.2 Prerequisites and 7.3 Installation of the required Software Components, and the CODESYS Development System with the three components CODESYS, CODESYS Control for Raspberry Pi and PiXtend V2 Professional for CODESYS already successfully installed.

7.7.2. RS232 interface – Cable connection

To use the RS232 interface, the data lines must be cross-connected, i.e. RX to TX.

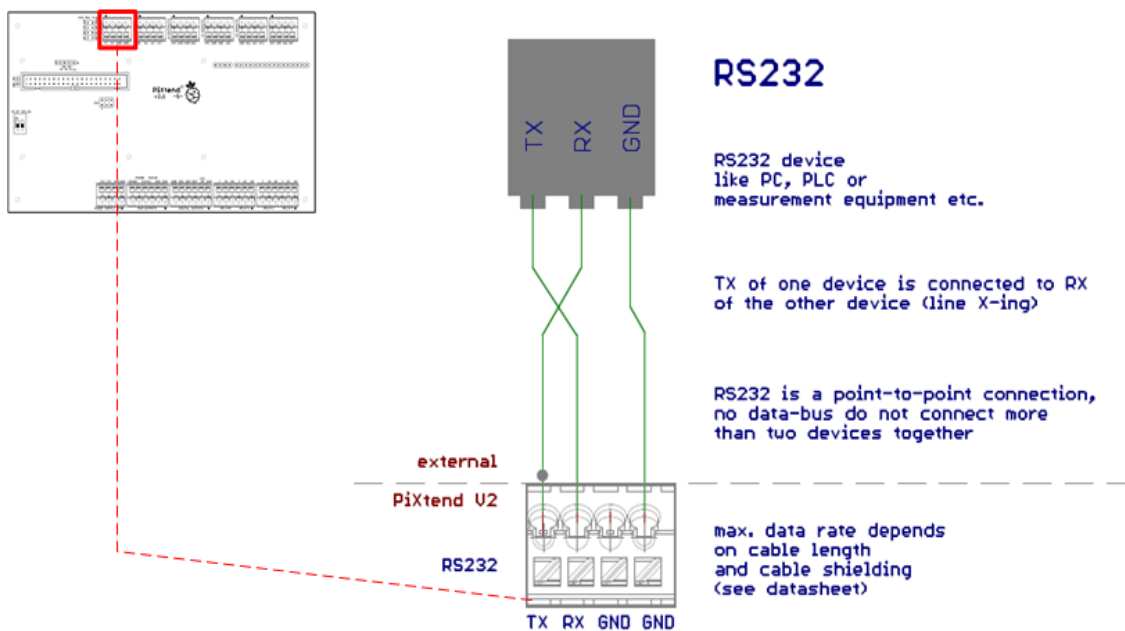


Figure 54: CODESYS - RS232 wiring diagram

A USB-to-serial adapter is required to connect to a PC.

Additional information on RS232 can be found in the data sheet (PiXtend V2 -S- or PiXtend V2 -L- technical data sheet in the hardware manual).

7.7.3. RS485 interface – Cable connection (only PiXtend V2 -L-)

To use the RS485 interface, the data lines must be connected 1: 1, i.e. each A (A) at A (A) and B (B) at B (B).

To use the RS485 interface, the GPIO 18 of the Raspberry Pi should be activated. This connects the serial interface of the Raspberry Pi to the RS485 hardware chip. It is no longer possible to use the RS232 interface after the changeover.

NOTICE

The RS485 interface is only available on the PiXtend V2 -L-!

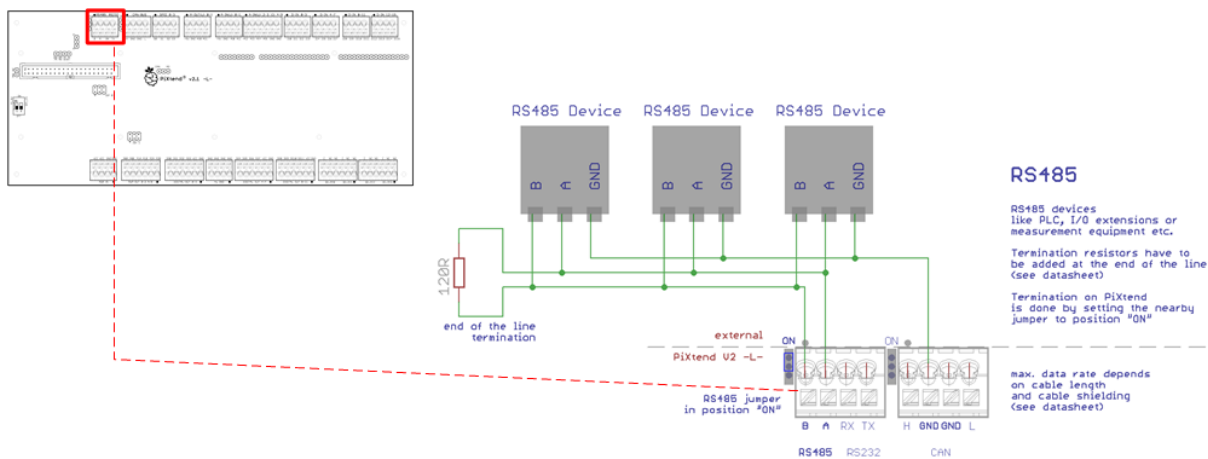


Figure 55: CODESYS - RS485 wiring diagram

7.7.4. Preparing Linux

7.7.4.1 Adjusting the interface settings

First the settings of the Raspberry Pi interfaces need to be adjusted to use the serial interface with CODESYS. This means that the serial port is disabled for Linux with the commands below and no longer sends data (access to the Linux console via the serial port is disabled).

The serial interface of the Raspberry Pi is activated or adjusted using the raspi-config program:

```
sudo raspi-config
```

Switch to the following in the program:

Interfacing Options --> Serial --> <No> --> <Yes> --> <Ok>

The following entry can then be found in the file /boot/config.txt:

```
enable_uart=1
```

Bluetooth® is automatically deactivated and UART is activated. The program raspi-config does all the work.

If you are using our CODESYS SD card image, you can now restart the Raspberry Pi and then continue with chapter 7.7.5 Terminal program.

7.7.4.2 Activating the interface in the CODESYS configuration

In the next step, we enter the interface in the CODESYS configuration, so CODESYS can access it.

Since CODESYS V3.5 SP11 (V3.5.x.x):

As of SP11, the two lines mentioned must be “commented out”. This is done by adding a semicolon ";" before the lines. Note that the changes are not made in CODESYSControl.cfg, but in CODESYSControl_User.cfg.

Use the following command:

```
sudo nano /etc/CODESYSControl_User.cfg
```

Add these lines to the end of the file:

```
[SysCom]
;Linux.Devicefile=/dev/ttyS
;portnum := COM.SysCom.SYS_COMPORT1
```

On the SD image for PiXtend V2 “CODESYS V2.X.X” with SP13 runtime, the lines are already entered and commented out. If you work with this image, there is no need to adjust the CODESYSControl_User.cfg settings.

If you want to use a serial USB interface, for example another RS232/RS485 dongle, the semicolons must be removed, and the name of the Linux device adjusted:

```
[SysCom]  
Linux.Devicefile=/dev/ttyUSB  
portnum := COM.SysCom.SYS_COMPORT1
```

Make sure that there is no number behind ttyUSB otherwise, access from CODESYS will not work.

You can find out the device or device name with the following command:

```
dmesg | grep -i tty
```

7.7.4.3 Reboot the Raspberry Pi

Restart the Raspberry Pi to apply the settings.

```
sudo reboot
```

7.7.5. Terminal Program

You need a terminal program for communication with a PC.

7.7.5.1 Installing a terminal program

If no terminal program is available yet, a terminal program should be installed as the next step. We use HTerm in our example (<http://http://der-hammer.info>).

7.7.5.2 Open the terminal and adjust the settings

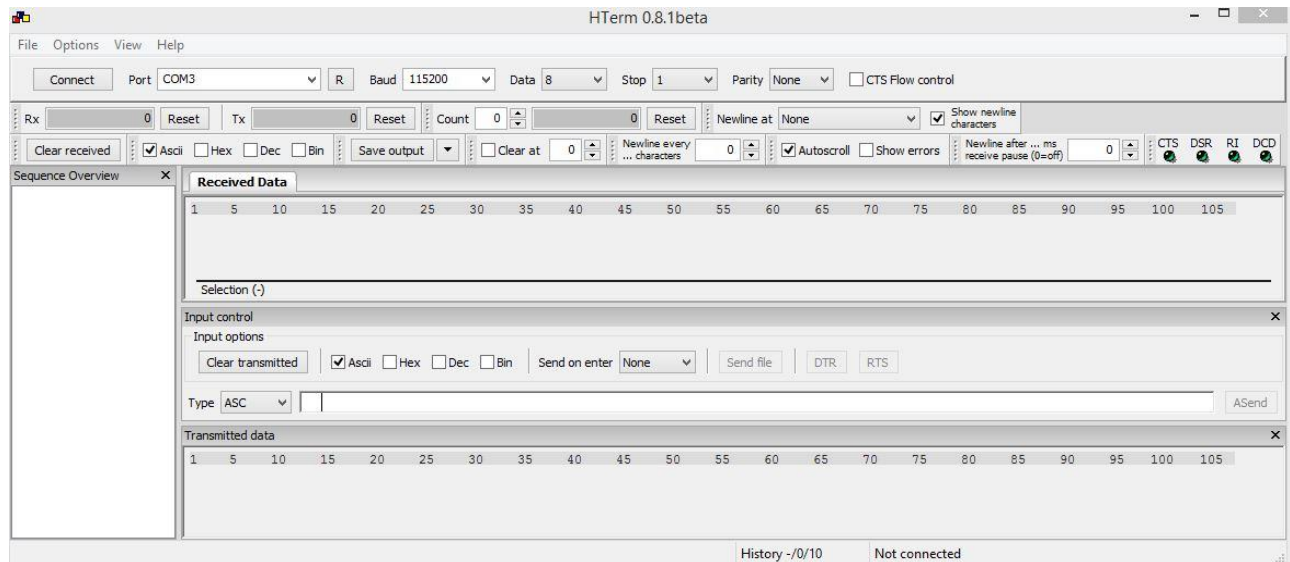


Figure 56: CODESYS - HTerm 0.8.1beta - RS232 test program

Settings: - Port: In the device manager under ports (COM and LPT)

- Baud rate: like with CODESYS, e.g. 9600 baud
- Data: 8
- Stop: 1
- Parity: none

As soon as you have established the connection with the "Connect" button, data can be received and sent.

7.7.6. CODESYS

7.7.6.1 Start the CODESYS project “PiXtendSerialTest”

First the PiXtendSerialTest project must be opened and the program transferred to the Raspberry Pi and started. The test project is included with the PiXtend V2 CODESYS package.

NOTICE

Note that no RS485 interface is available on the PiXtend V2 -S-. Therefore ignore the “Enable RS485” switch.

The RS232 part of the project is fully compatible with PiXtend V2 -S-.

The PiXtend V2 -L- has an RS485 interface and the RS485 test can be performed.

7.7.6.2 Visualization



Figure 57: CODESYS - Visualization - Test program - Serial communication

Quick guide:

1. Set the baud rate (must match the communication partner settings).
2. Press the "CONNECT" button
Data is now automatically received and displayed. To send a message:
3. Enter message in the "Send" field.
4. Press the "SEND" button.

Function of the buttons, switches and selection fields:

"CONNECT / DISCONNECT"

Connect or disconnect to the PC terminal or any other device.

"CLEAR ALL"

The received messages, received (RX) and transmitted (TX) bytes and errors are deleted. In addition, the automatic sending of messages is disabled.

"SEND"

This sends the message entered in the input field.

"Baud rate"

The desired baud rate (bits/second) can be selected here.

"Enable RS485" (only PiXtend V2 -L-)

This switch can be used to switch between the RS232 and RS485 interface. For this purpose, however, a disconnect must first be carried out, since both interfaces can not be operated simultaneously.

"auto scroll"

If this is active, the display of the received messages always scrolls automatically to the most recent message.

"enable" Auto send

This activates the automatic transmission of the entered information at the defined interval.

Input/output fields:

"Receive"

Here the received data is displayed in tabular form with time stamps.

"Send"

The data to be sent is entered here.

"Auto send"

Data will be sent automatically.

"Interval"

This is the interval in which the data is to be sent automatically; the minimum is 1 second.

Status display:

"Error Code"

If an error occurs during opening, closing, reading or writing during operation, this is indicated by the error LEDs. Additionally, the corresponding error code is displayed in the "Error Code" field.

"Status"

This displays the current status of the program.

7.8. CAN communication

7.8.1. Introduction

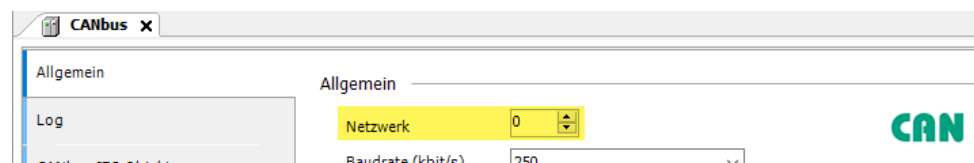
This robust and reliable bus is used in many areas, for example in industrial automation. In automation technology, the CANopen protocol (OSI layer 7 - application layer) is often used, which is supported by CODESYS and available for the Raspberry Pi.

In this chapter we describe the necessary steps to set up CANOpen communication with the PiXtend V2 -L- under CODESYS. For this you need an SD card with the PiXtend V2 SD card image "CODESYS Control", afterwards the CANopen configuration is explained in CODESYS.

NOTICE

The CAN bus interface is only available on the PiXtend V2 -L-!

Under CODESYS, the network number "0" must always be set for the CANBus device to be able to use the CAN bus of the PiXtend V2 -L-.



If you have any further questions, please send them to us by e-mail (support@pixtend.de). You will receive an answer as soon as possible.

The latest versions of all documents and software components can be found in the download section of our website <https://www.pixtend.de>.

7.8.1.1 Requirements

To activate the CAN bus functionality, basic knowledge of the Raspbian operating system (Linux) and the operation of CODESYS is required.

To test CAN communication under Linux, the PiXtend V2 -L- must be connected to a CAN bus with at least one active CAN device. It can be connected to one or more other PiXtends (PiXtend V2 -L-).

For the CODESYS CANopen master/slave communication, two PiXtend V2 -L- boards are used in this chapter.

7.8.1.2 Restrictions

The PiXtend V2 -L- CAN controller cannot be used simultaneously with the PiXtend V2 -L- DAC, both chips “share” one SPI chip-select channel.

7.8.2. Hardware connection to CAN bus

Refer to the PiXtend V2 -L- hardware manual for the correct pin assignment of the PiXtend V2 -L- CAN port.

CAUTION

To use the CAN bus hardware on the PiXtend V2 -L-, the jumper “CAN” / “AO” must be set to “CAN”; otherwise, the CAN bus cannot be used. The DAC is disabled by this action and can no longer be used.

Note the jumper for the “termination” of the CAN bus.

7.8.3. Preparation and tests under Linux

To use the CAN controller in CODESYS, some adjustments to the Raspberry Pi OS (Linux) operating system are necessary.

If you have already made your own adjustments to your image, we recommend creating a new SD card for initial CAN tests. For example, our "CODESYS Control" SD card image, which we provide in the latest version in our download section.

NOTICE

All steps described here refer to the Raspberry Pi OS image starting with "Stretch" and are performed as the user "pi". The support of the CAN controller MCP2515 on the PiXtend V2 -L- board is only guaranteed from kernel version 4.0.8. If you are using a "Wheezy" image, you must first update to kernel version 4.0.8 (or later).

CAUTION

If you are already using an SD card with a "CODESYS Control" image, stop the CODESYS runtime component to start the CAN bus. Use the following command: `sudo service codesyscontrol stop`

After rebooting the Raspberry Pi, CODESYS is available as usual.

Preparing the SD card image

Download the latest Raspberry Pi OS image from the official download site:

<https://www.raspberrypi.com/software/>

Transfer the downloaded image to an SD card (8 GB recommended) using a tool of your choice. The free program Win32DiskImager can be used under Windows for this task or use the Raspberry Pi Imager, the all-in-one tool from the Raspberry Pi Foundation.

Boot the new image and make the usual adjustments using `raspi-config`:

- Extend the file system
- Time zone settings
- Country settings
- Language settings
- Keyboard layout
- Enable SSH access

Reboot the Raspberry Pi with this command

```
sudo reboot
```

and then login via SSH in the command line as the user pi (for example with Putty) or directly with a screen and keyboard.

Tip:

If you use our PiXtend image "CODESYS", you can skip sections 7.8.3.3 and 7.8.3.4, because we have already prepared everything for you. Only a small adjustment is necessary, as described in 7.8.3.2.

7.8.3.1 Configuration of the Device Tree Overlay

To operate the MCP2515 CAN chip of the PiXtend board from the kernel, the following adjustments to the file `/boot/config.txt` are necessary. Open the editor with the following command

```
sudo nano /boot/config.txt
```

and add the following entries at the end:

```
dtparam=spi=on
dtoverlay=mcp2515-can1
dtparam=oscillator=20000000
dtparam=interrupt=4
dtparam=spimaxfrequency=1000000
dtoverlay=spi-bcm2835-overlay
dtoverlay=spi-dma
dtdebug=on
```

Save the changes with `Ctrl+O` and exit the editor with `Ctrl+X`

Reboot the Raspberry Pi with this command:

```
sudo reboot
```

7.8.3.2 Edit blacklist

To prevent the kernel module “mcp251x” from being loaded automatically at boot time, add it to the blacklist.

Open the file `/etc/modprobe.d/raspi-blacklist.conf`

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

add the following lines:

```
blacklist mcp251x
```

The file is initially completely empty, but can still be used.

Save the changes with `Ctrl+O` and exit the editor with `Ctrl+X`

Reboot the Raspberry Pi with this command:

```
sudo reboot
```

7.8.3.3 Creating a CAN script

To activate the PiXtend V2 -L- CAN controller, the following actions must be taken:

- Configure GPIO 24 as an output and set to TRUE (SPI Enable)
- Configure GPIO 27 as an output and set to TRUE (SPI CS1 is passed on to the CAN controller, the DAC is deactivated)
- Load the kernel module "mcp251x"
- Configure and enable the interface can0

Since the same steps are always necessary, we will create a new script in the home directory of the user pi:

```
nano activateCAN.sh
```

Add the following lines:

```
#!/bin/sh
if [ ! -d "/sys/class/gpio/gpio24" ]; then
    echo "24" > /sys/class/gpio/export
fi
echo "out" > /sys/class/gpio/gpio24/direction
echo "1" > /sys/class/gpio/gpio24/value
if [ ! -d "/sys/class/gpio/gpio27" ]; then
    echo "27" > /sys/class/gpio/export
fi
echo "out" > /sys/class/gpio/gpio27/direction
echo "1" > /sys/class/gpio/gpio27/value
sleep 1
sudo modprobe mcp251x
sudo /sbin/ip link set can0 up type can bitrate 125000
sudo ip -s -d link show can0
```

Save the changes with Ctrl+O and exit the editor with Ctrl+X.

Run the script by entering the following command:

```
chmod +x activateCAN.sh
```

7.8.3.4 Activating CAN

Using the script

```
sudo ./activateCAN.sh
```

activates the CAN controller.

Use

```
ifconfig
```

to check if there is an entry for a can0 interface.

7.8.3.5 Installing and using CAN utilities

Install “can-utils”, a collection of useful user space programs to access the CAN interface directly with SocketCAN under Linux:

```
sudo apt-get install can-utils
```

With the command

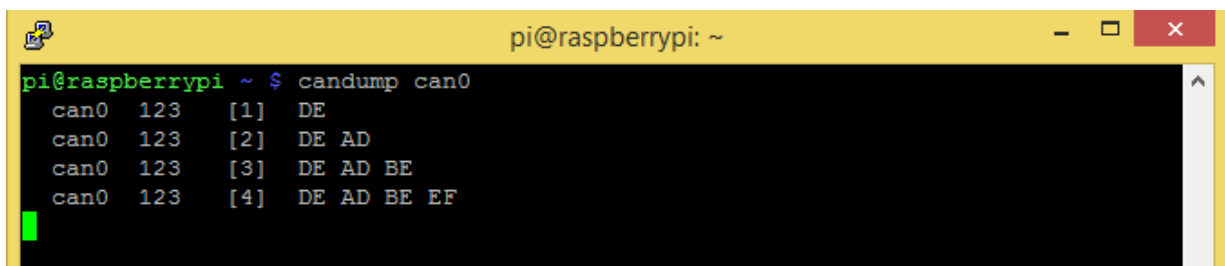
```
cansend can0 123#DEADBEEF
```

a CAN message with ID 123 and the content 0xDEADBEEF is sent.

With the command

```
candump can0
```

all the data traffic on can0 can be monitored:



```
pi@raspberrypi ~ $ candump can0
can0 123 [1] DE
can0 123 [2] DE AD
can0 123 [3] DE AD BE
can0 123 [4] DE AD BE EF
```

Figure 58: CODESYS - CAN bus test under Linux

CAN messages can be generated continuously for test purposes with the program “cangen”.

Further useful functions and parameters of can-utils can be found in the respective help pages.

With these tools, you can already participate in a CAN bus communication under Linux.

Continue with the further steps (preparation for CODESYS) only after you have successfully received and sent data with the “can-utils”.

7.8.4. Preparations for CODESYS

For the steps in this chapter, you need an SD card with CODESYS Runtime installed. Regardless of whether you are just starting with the CAN bus or have already built up your own image and system, we recommend that you first use our SD card image “CODESYS Control”.

7.8.4.1 Creating the start script

To load the kernel module for the PiXtend V2 -L- CAN controller correctly, we need a start script which executes the following actions in exactly this order:

- Stop the CODESYS Control
- Activate CAN with the help of the previously created script activateCAN.sh (see section 7.8.3.4)
- Start the CODESYS Control

Open a console on the Raspberry Pi (via SSH with the tool Putty) and create the file “startPLCCAN.sh” in the home directory:

```
nano startPLCCAN.sh
```

Add the following lines:

```
#!/bin/sh
sudo service codesyscontrol stop
sudo /home/pi/activateCAN.sh
sudo service codesyscontrol start
```

Save the changes with Ctrl+O and exit the editor with Ctrl+X.

Run the script by entering the following command:

```
chmod +x startPLCCAN.sh
```

7.8.4.2 Creating the baud rate script

Create the following baud rate script:

```
sudo nano /var/opt/codesys/rts_set_baud.sh
```

Add the following lines:

```
#!/bin/sh
BITRATE=`expr $2 \/* 1000`
ifconfig $1 down
echo ip link set $1 type can bitrate $BITRATE
ip link set $1 type can bitrate $BITRATE
ifconfig $1 up
```

Save the changes with Ctrl+O and exit the editor with Ctrl+X.

Run the script by entering the following command:

```
sudo chmod +x /var/opt/codesys/rts_set_baud.sh
```

7.8.4.3 Starting the CODESYS Control with PiXtend V2 -L- CAN support

Reboot the Raspberry Pi with this command:

```
sudo reboot
```

After the boot process, the CODESYS Control Runtime is started automatically.

To use CODESYS with CAN support, the script “startPLCCAN.sh” must also be executed after each start:

```
cd ~  
sudo ./startPLCCAN.sh
```

By executing the script, the CODESYS runtime is terminated, the GPIOs are configured accordingly, the mcp251x kernel module is loaded, the can0 interface is configured, and the CODESYS Control is then restarted.

If you want to enable CAN support permanently, let the start script run automatically on every reboot by editing the file.

```
sudo nano /etc/rc.local
```

and add the following lines before “exit 0”:

```
# Start CODESYS Control with PiXtend CAN Support  
sudo /home/pi/startPLCCAN.sh
```

Test the configuration by restarting the Raspberry Pi:

```
sudo reboot
```

Use “lsmod” to check whether the “mcp251x” module was loaded after the restart. Use “top” to check whether the CODESYS Control is working.

7.8.5. CODESYS Project with CANopen

If the preparations and tests under Linux were successful, a CANopen test project can now be created in CODESYS. We use two PiXtend V2 -L- devices and let them communicate with each other via CANopen.

We create an empty project to which two PiXtend V2 -L- devices are added. One device is the CANopen master and the other is the CANopen slave. Instead of using a second PiXtend V2 -L-, you can also use any other CAN bus device as a slave. For configuration, simply refer to the manual of your CAN slave.

In our example, we initially use only one byte that is transferred from the master to the slave. The example can then be expanded as required.

First create an empty project (File -> New Project -> Empty Project) and name it "PiXtendCAN_MasterSlave". Specify the location of your project if you do not want to use the default.

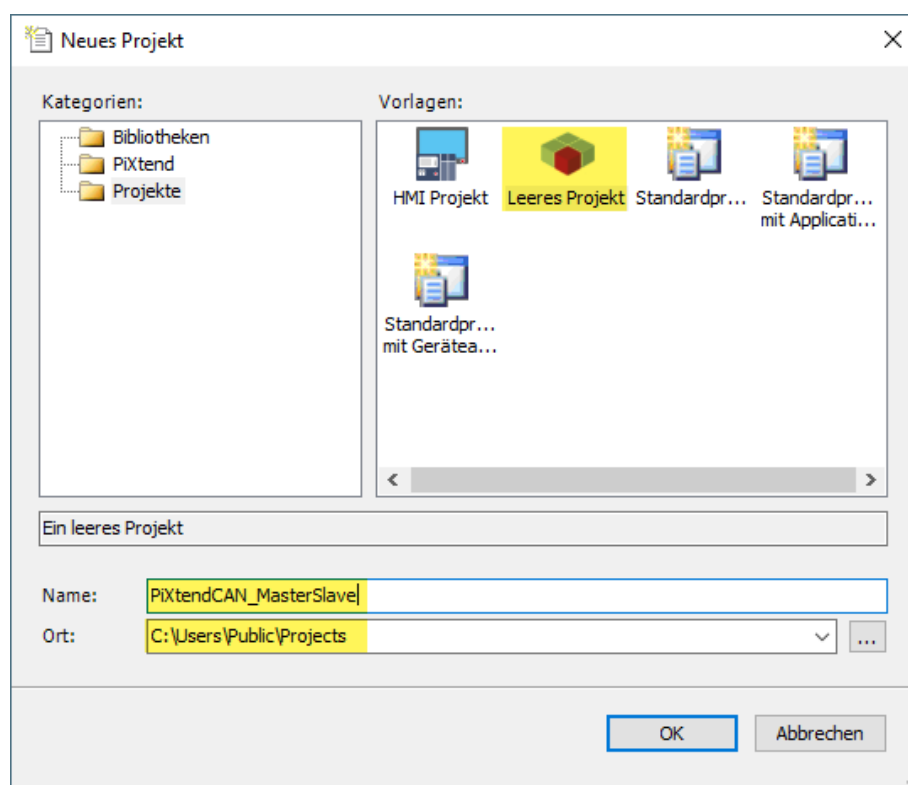


Figure 59: CODESYS - CAN bus - Creating a new project

7.8.5.1 PiXtend V2 -L- as CAN slave

Add a new device to the project by right-clicking on the project in the project tree and selecting “Add device”.

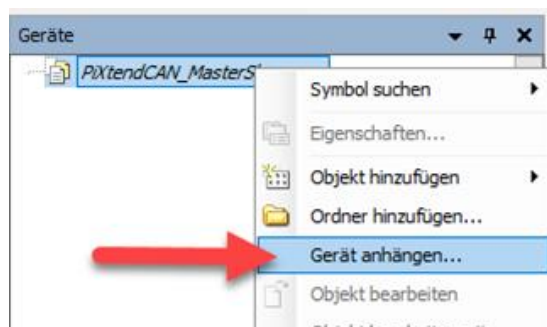


Figure 60: CODESYS - CAN bus slave - Adding SPS device

First change the manufacturer to “3S - Smart Software Solutions GmbH”, select the “CODESYS Control for Raspberry Pi” as the device, give it the name “PiXtendCAN_Slave”. Lastly, click on “Add device” and “Close”.

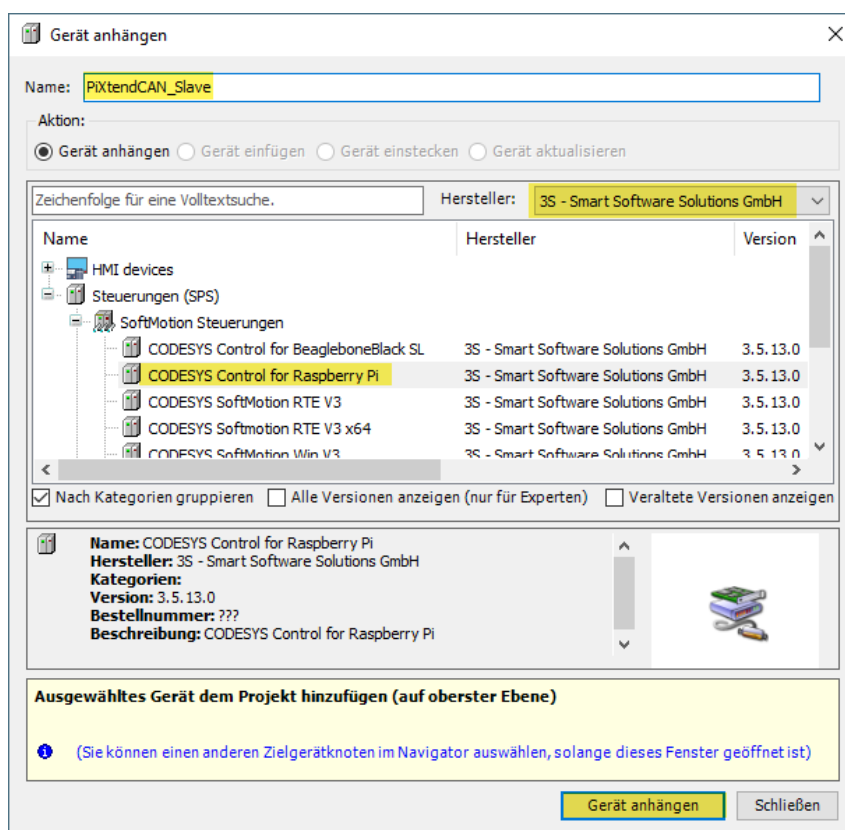


Figure 61: CODESYS - CAN bus slave - Adding Raspberry Pi

In the project tree, right-click on the “PiXtendCAN_Slave” device you have just added:

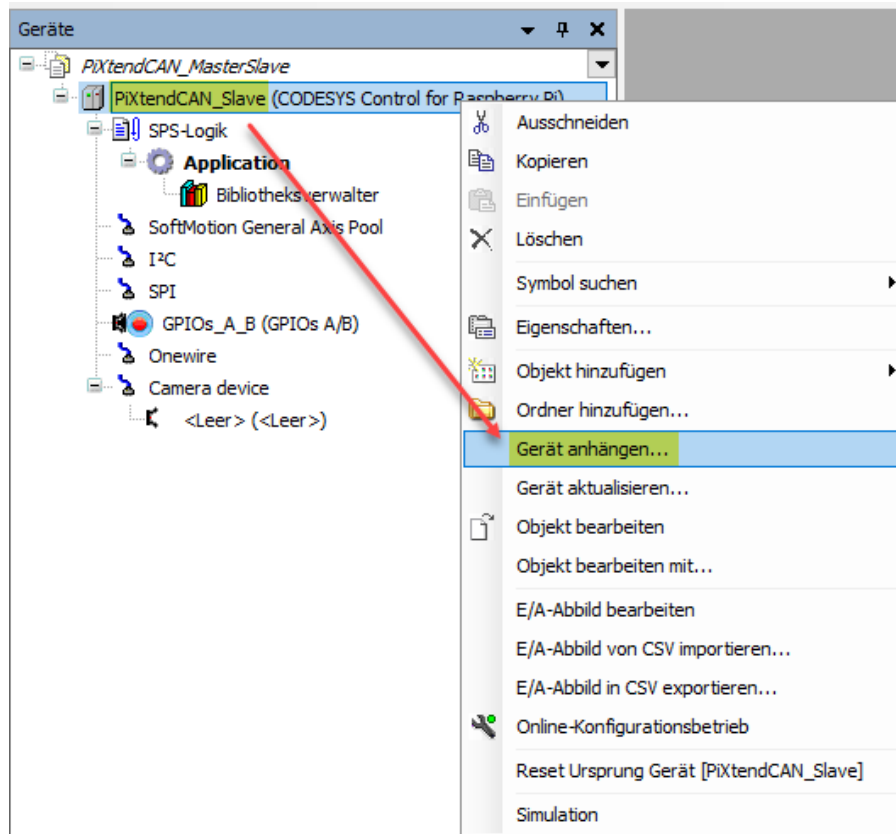


Figure 62: CODESYS - CAN bus slave - Adding a device to the SPS

Select "CANbus" in the dialog and add the device. The selection is faster when you select the company "3S - Smart Software Solutions GmbH" under "Manufacturer".

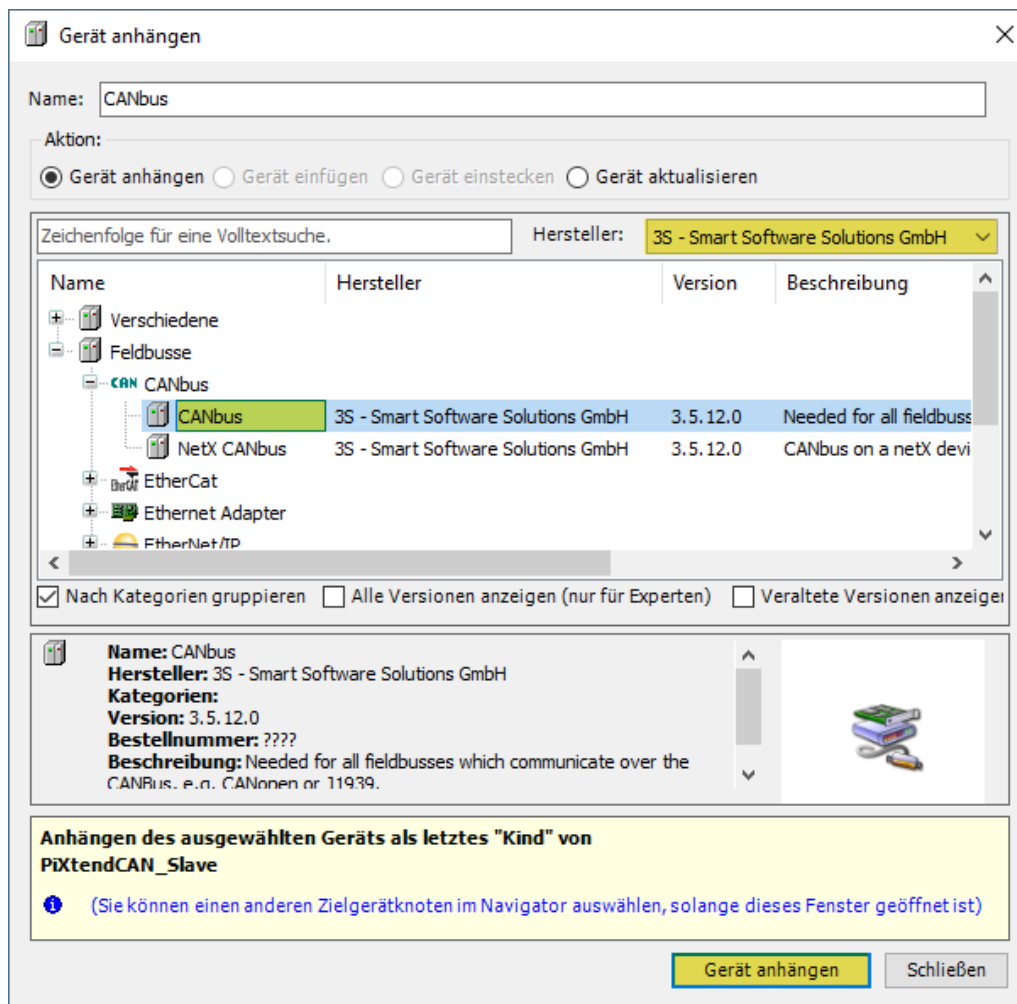


Figure 63: CODESYS - CAN bus slave - Adding CAN bus device

Leave the window open, we will need it again on the next page.

Now left-click on the “CANbus” device that was just added in the device tree, while the window “Add devices” is still open. After the click, the content of the “Add devices” window changes and you can add a “CANopen Device” below the CANbus device. The selection is easier when you select the company “3S - Smart Software Solutions GmbH” under “Manufacturer”

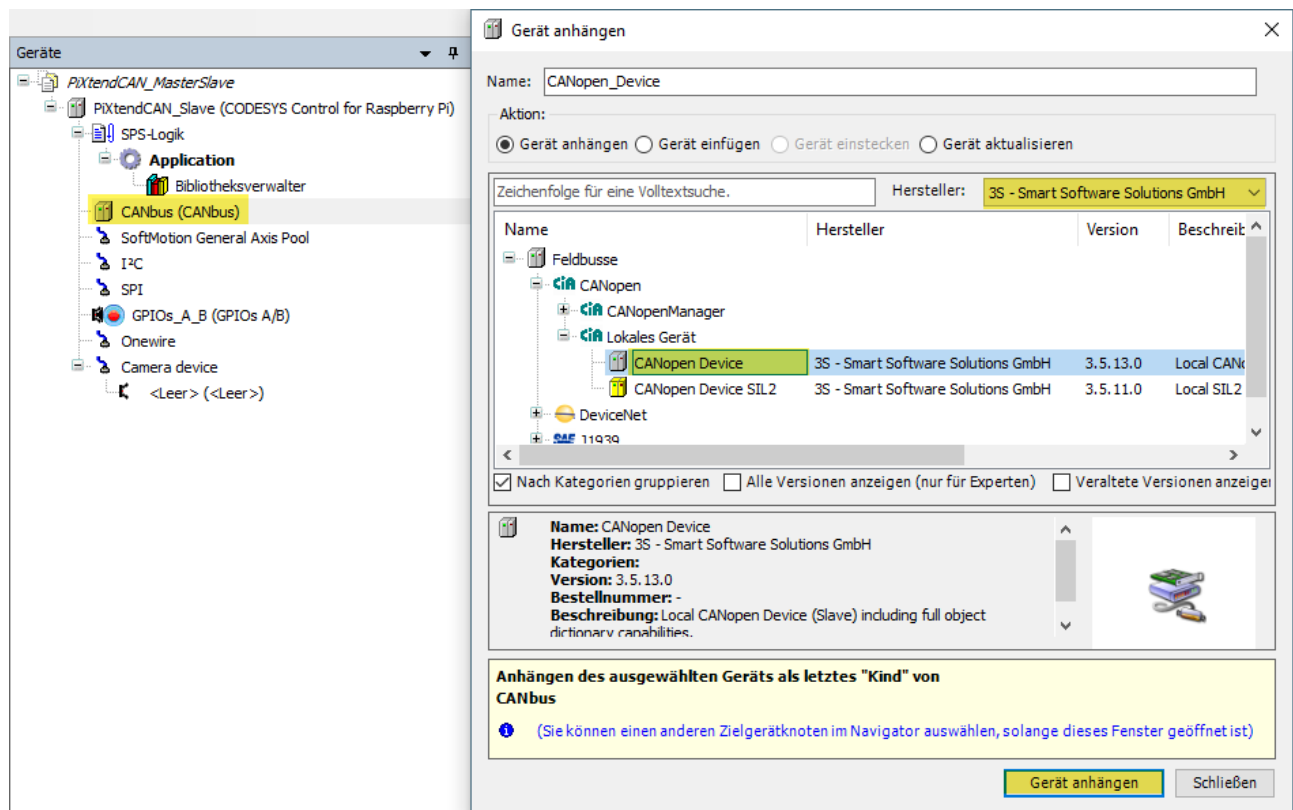


Figure 64: CODESYS - CAN bus slave - Adding CANopen device

After the addition, close the “Add devices” window.

Now double-click on the CANopen device that was added in the device tree and then click on the “Edit I/O area” button.

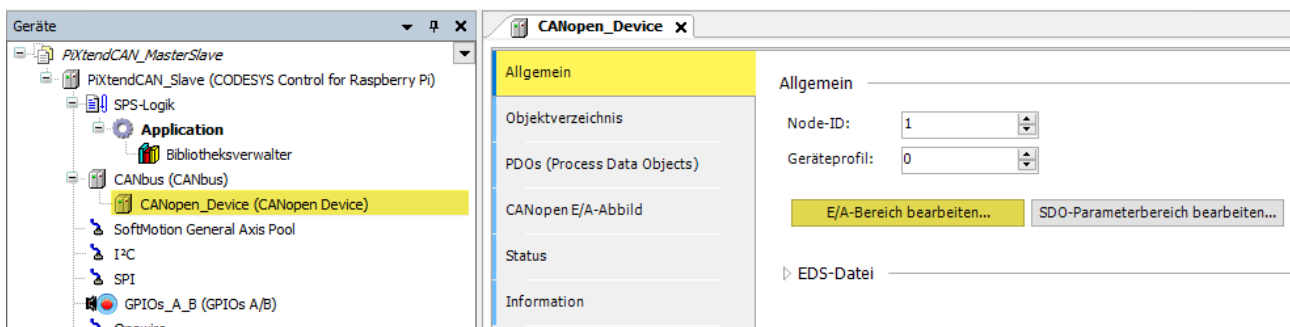


Figure 65: CODESYS - CAN bus slave - Configuring I/O area

Add a new area and select “Receive”. The remaining fields can remain unchanged since the master should initially send only a single USINT (BYTE) to the slave.

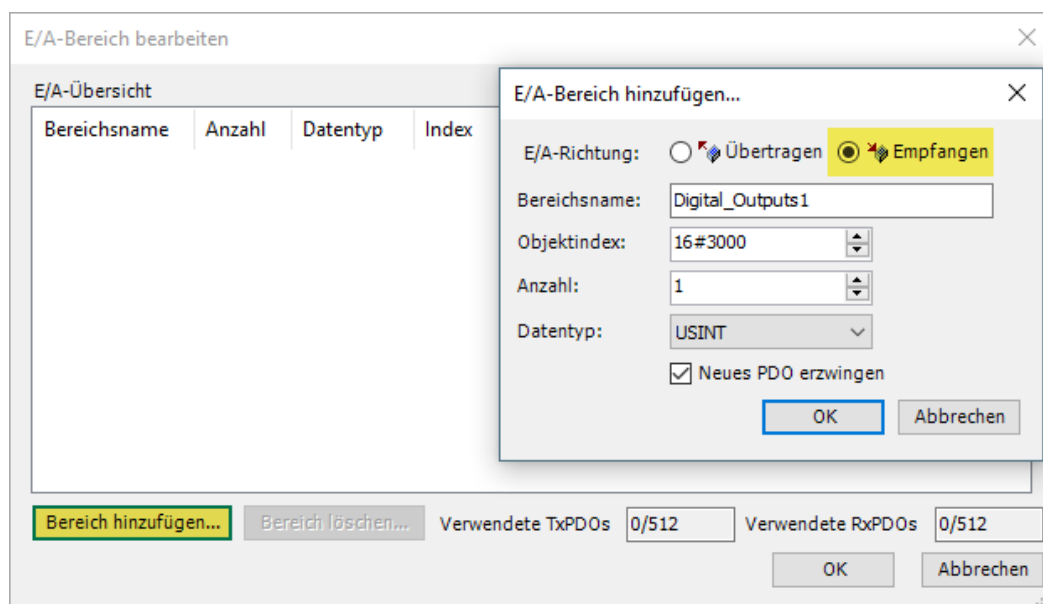


Figure 66: CODESYS - CAN bus slave - Setting PDO

Confirm your entries and close the dialog.

Now open the “EDS file” area and install the device you have just created in the device repository using the corresponding button. To find your device quicker, change the manufacturer name and/or the product name.

The screenshot shows the 'CANopen_Device' configuration window in CODESYS. The left sidebar contains a tree view with the following items: 'Allgemein' (highlighted), 'Objektverzeichnis', 'PDOs (Process Data Objects)', 'Log', 'CANopen E/A-Abbild', 'CANopen IEC-Objekte', 'Status', and 'Information'. The main area is divided into two sections: 'Allgemein' and 'EDS-Datei'. In the 'Allgemein' section, 'Node-ID' is set to 1 and 'Geräteprofil' is set to 0. Below these are two buttons: 'E/A-Bereich bearbeiten...' and 'SDO-Parameterbereich bearbeiten...'. In the 'EDS-Datei' section, 'Herstellername' is 'Kontron Electronics GmbH', 'Herstellernummer' is 801, 'Produktname' is 'MyCANopenDevice', 'Produktnummer' is 1, and 'Revisionsnummer' is 1. At the bottom of the 'EDS-Datei' section are two buttons: 'Ins Geräterepositary installieren' and 'EDS-Datei exportieren...'.

Figure 67: CODESYS - CAN bus slave - Installing your own CAN device

If you receive a message that a device with this name or manufacturer number already exists, you can change the manufacturer number. If you are sure that the other device is only an experimental device, overwrite it.

All information about our self-made CANopen device is now stored in the CODESYS device database and is ready for use, for example, by the CANopen master.

Now add a Task configuration to the application and a POU named "POU_Main".

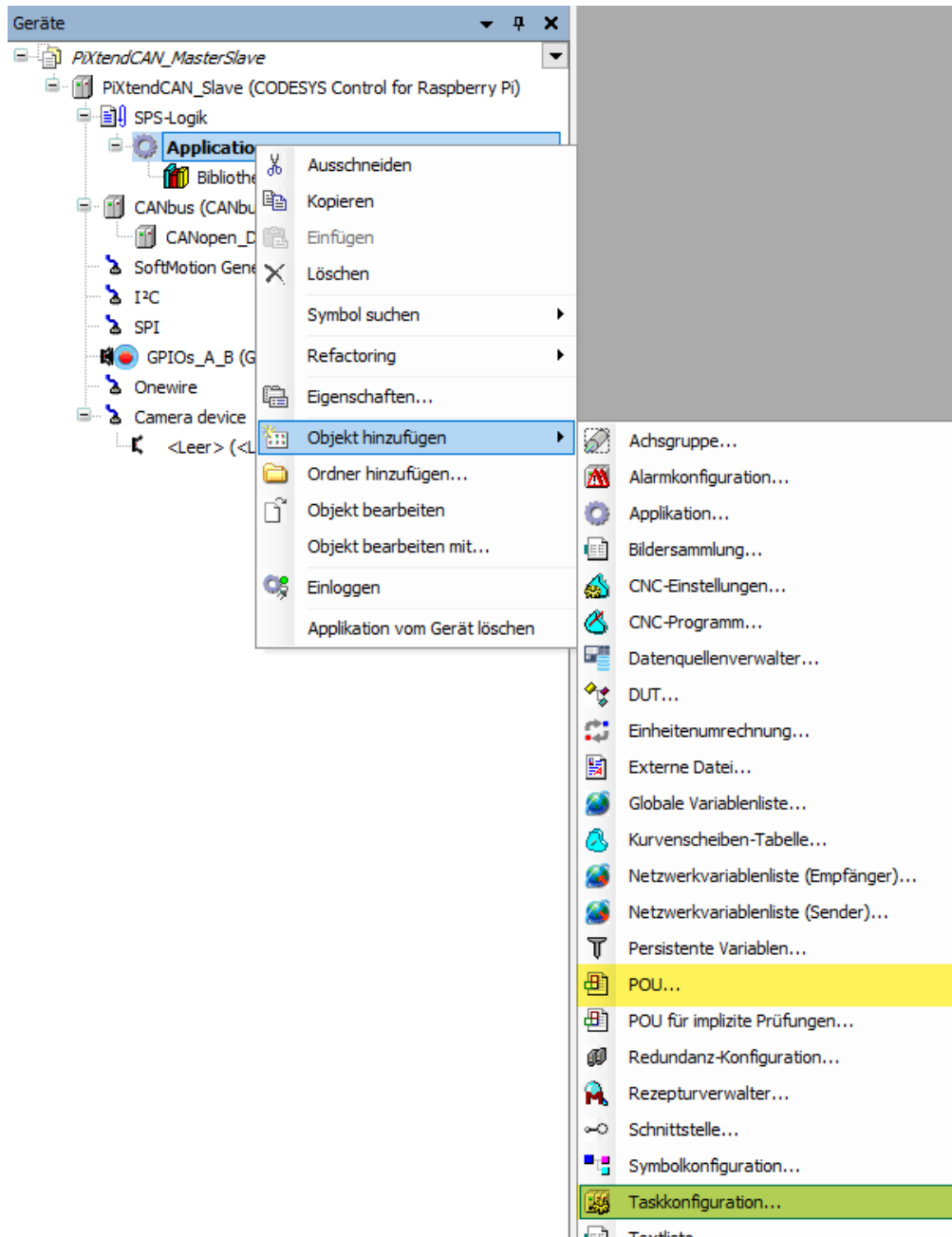


Figure 68: CODESYS - CAN bus slave - POU and task configuration

Open the task configuration and add the execution of the main program “POU_Main” to the task:

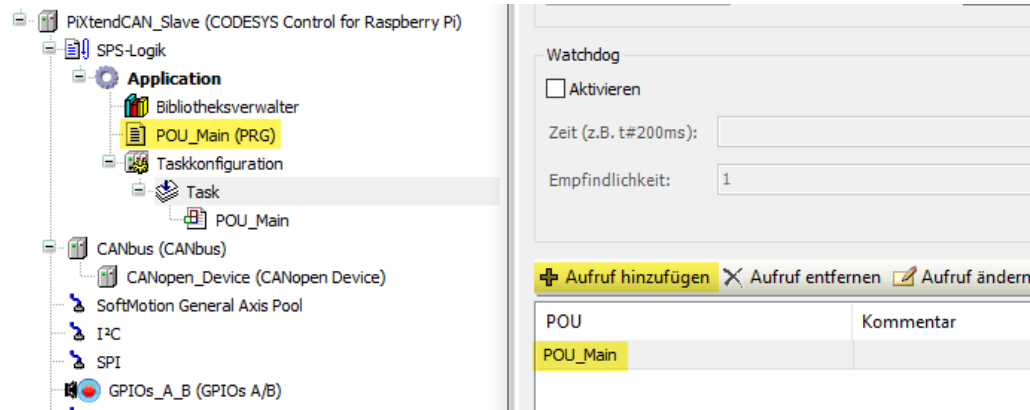


Figure 69: CODESYS - CAN bus slave - Task configuration

Switch to the “CANopen_Device” and change the settings for updating the variables under the tab “CANopen I/O mapping”:

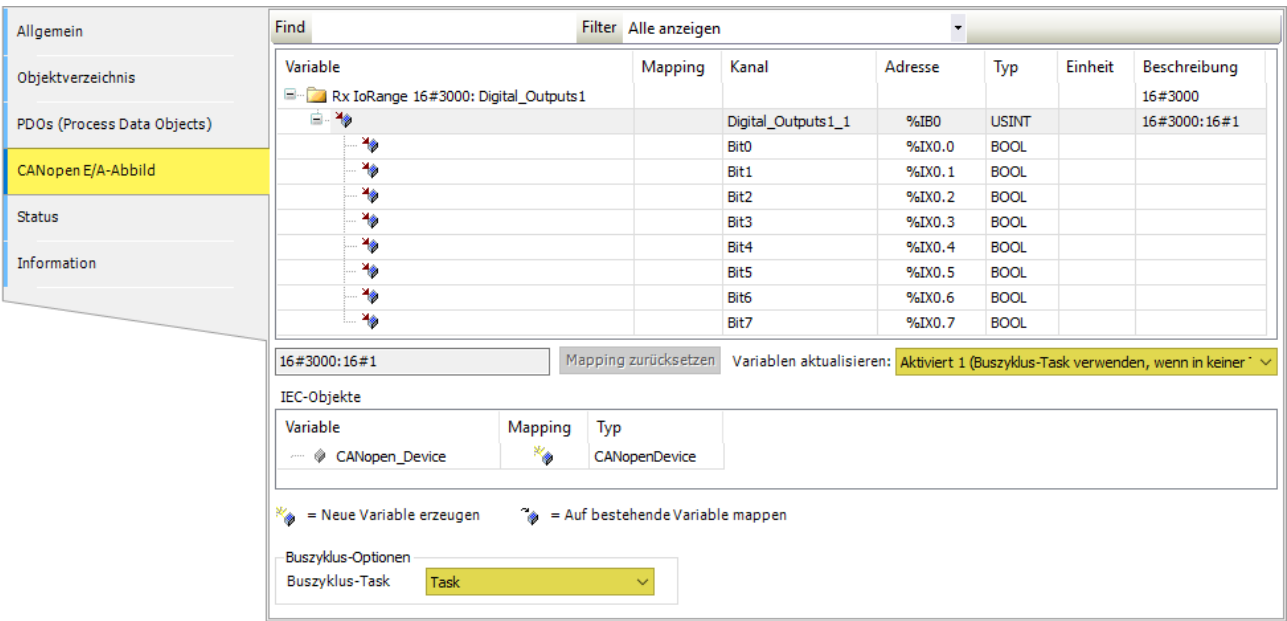


Figure 70: CODESYS - CAN bus slave - CANopen I/O mapping

Select the option “Enabled 1 - Use bus cycle task if not used in any task” and “Task” as the bus cycle task.

Lastly, the baud rate for the CAN bus should be reduced to “125 kBit/s”:

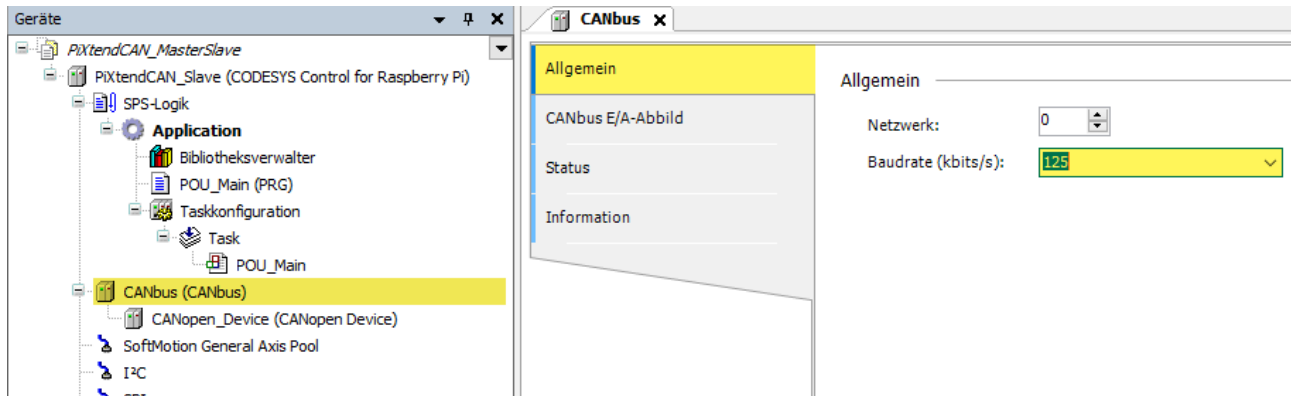


Figure 71: CODESYS - CAN bus slave - Baud rate

Now the configuration of the slave is completed.

7.8.5.2 PiXtend V2 -L- as CANopen master

We add another PiXtend V2 -L- device to the project, which takes over the role of the CANopen master.

Right-click on the “PiXtendCAN_MasterSlave” project in the project tree and add another “CODESYS Control for Raspberry Pi” device with the name “PiXtendCAN_Master”:

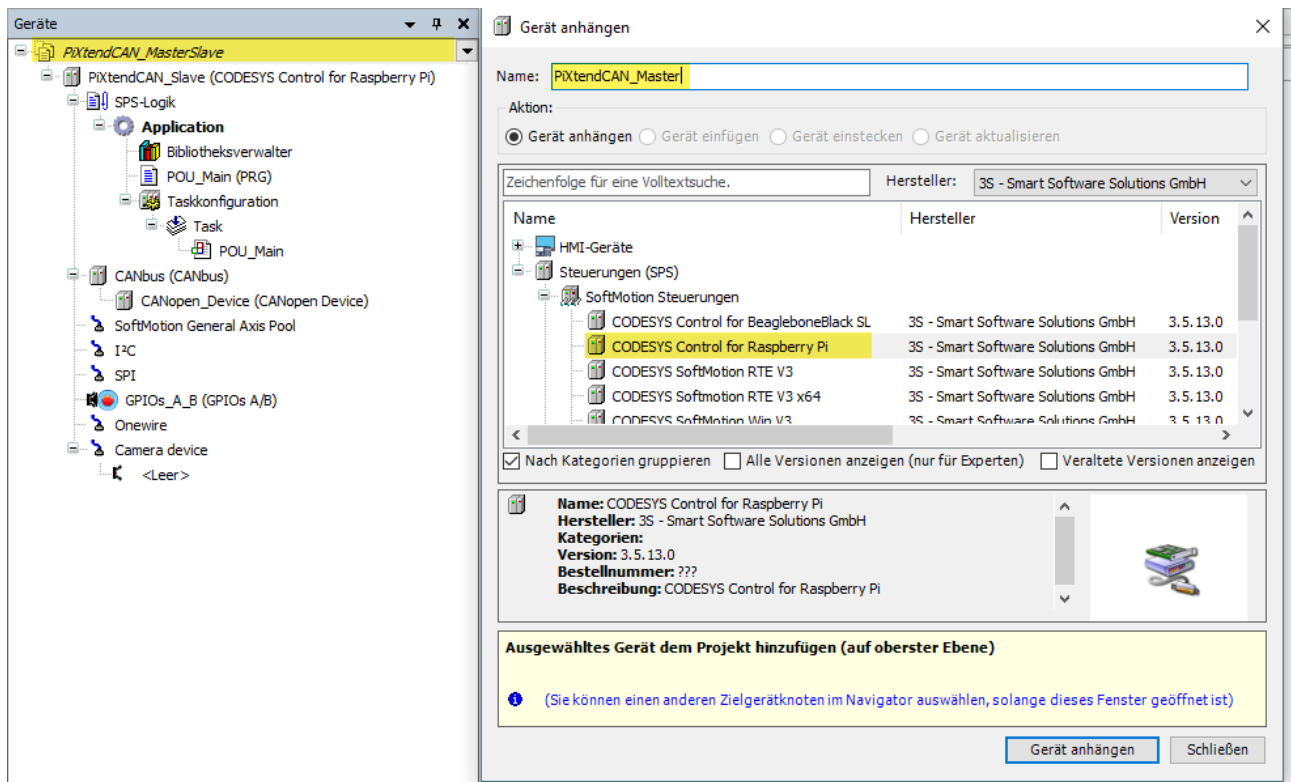


Figure 72: CODESYS - CAN bus master - Adding Raspberry Pi

Add a “CANbus” device to the “PiXtendCAN_Master” as well:

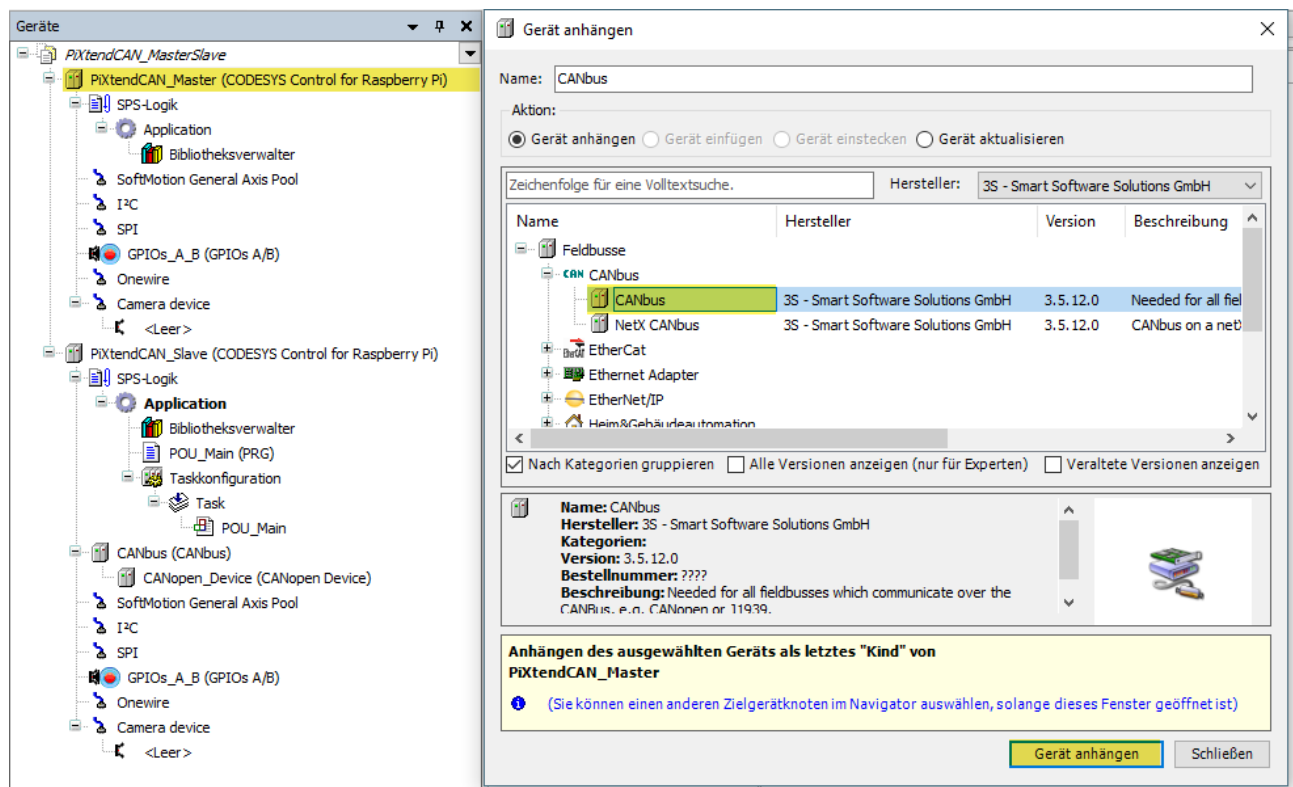


Figure 73: CODESYS - CAN bus master - Adding a CANbus

Set the baud rate of the CANbus to “125 kBit/s”, or to the same setting that was selected for the slave.

Now right-click on the “CANbus” entry you just added and add a “CANOpen_Manager” to it:

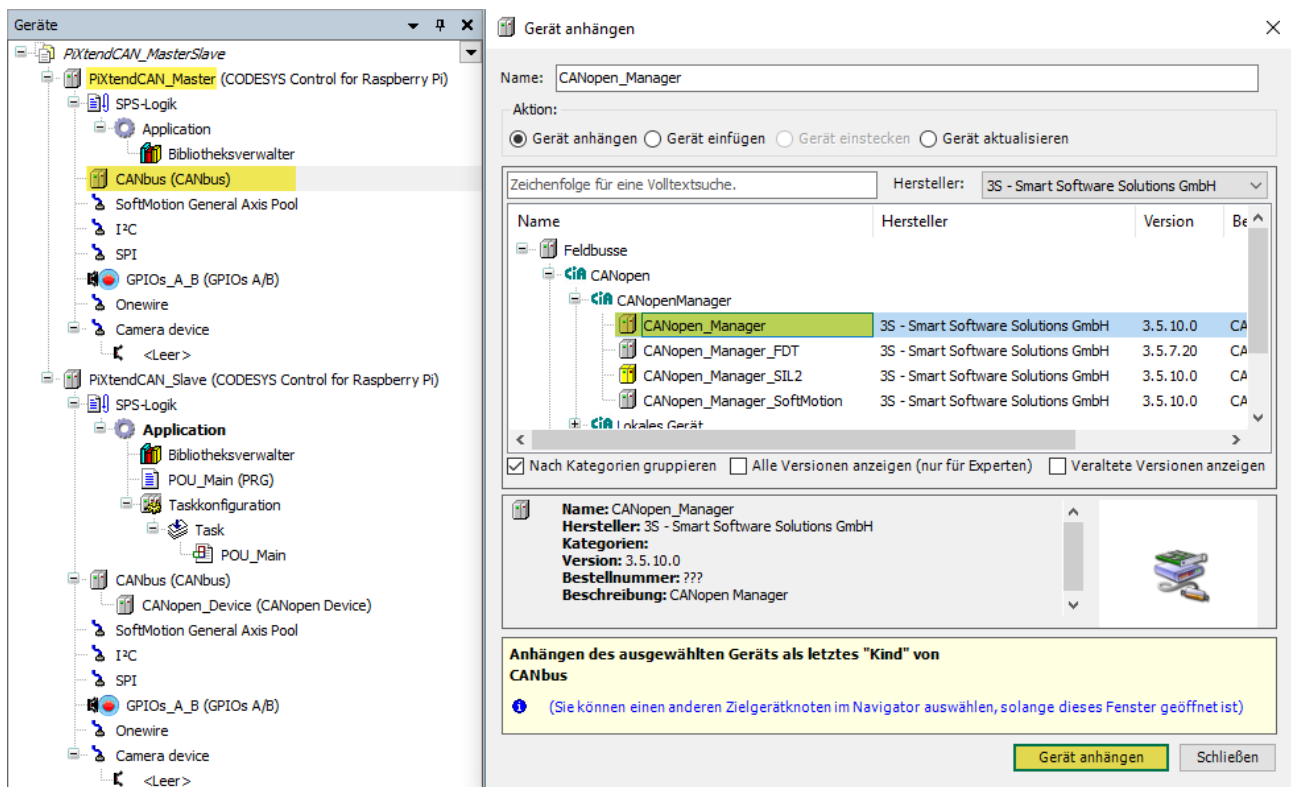


Figure 74: CODESYS - CAN bus master - Adding a CANOpen manager

Leave the window open, we will need it again on the next page.

Now select the “CANopen_Manager” you have just added and add the “MyCANopenDevice” you created. If you entered a manufacturer name when creating the CANopen slaves device, you can select this name under “Manufacturer”.

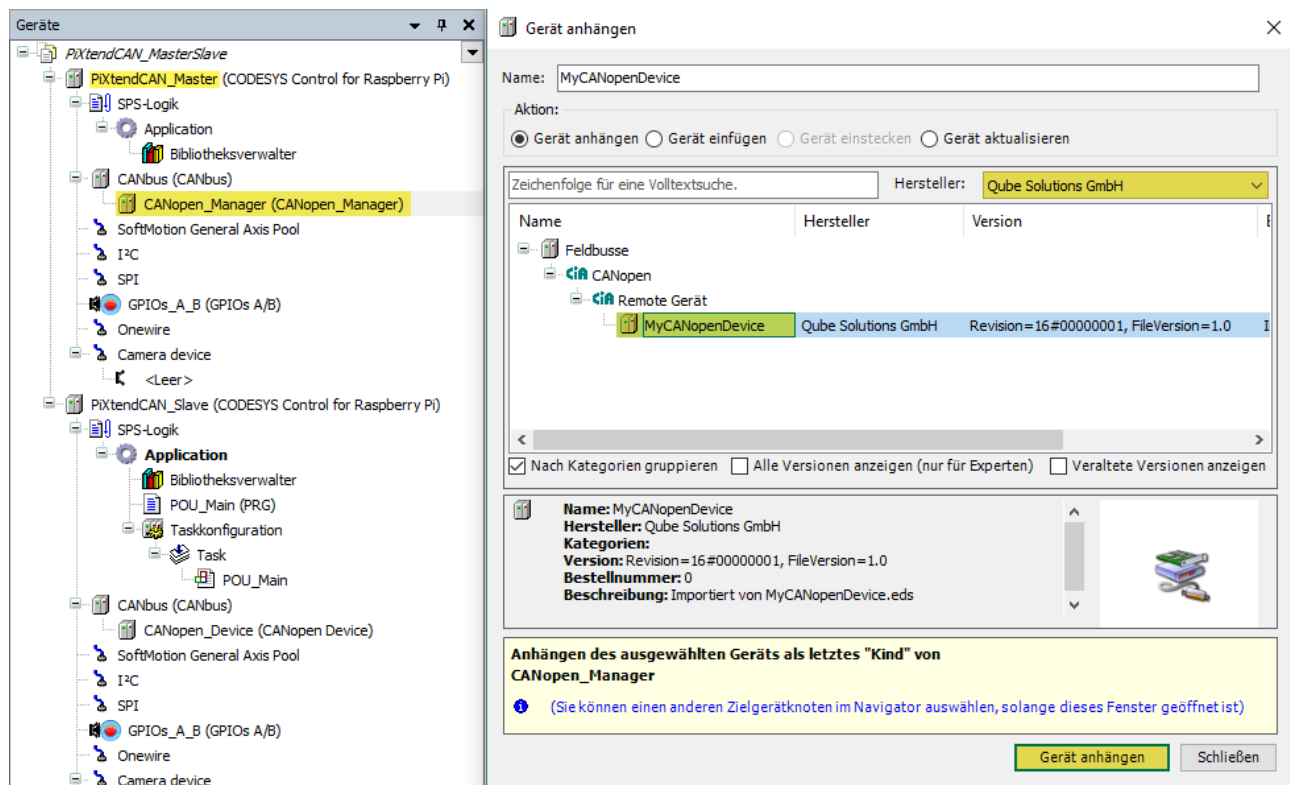


Figure 75: CODESYS - CAN bus master - Adding CAN bus slave device

Add to the “PiXtendCAN_Master” device a “Task Configuration” and a “POU_Main” as the main program for the application. In the “Task” entry in the “Task Configuration”, create an execution of the “POU_Main” block, similar to what you have done when creating the CANopen slave device.

Now open the I/O configuration (CANopen I/O mapping) of the “MyCANopenDevice” device and select the entry “Enabled 1 - Use bus cycle task if not used in any task”.

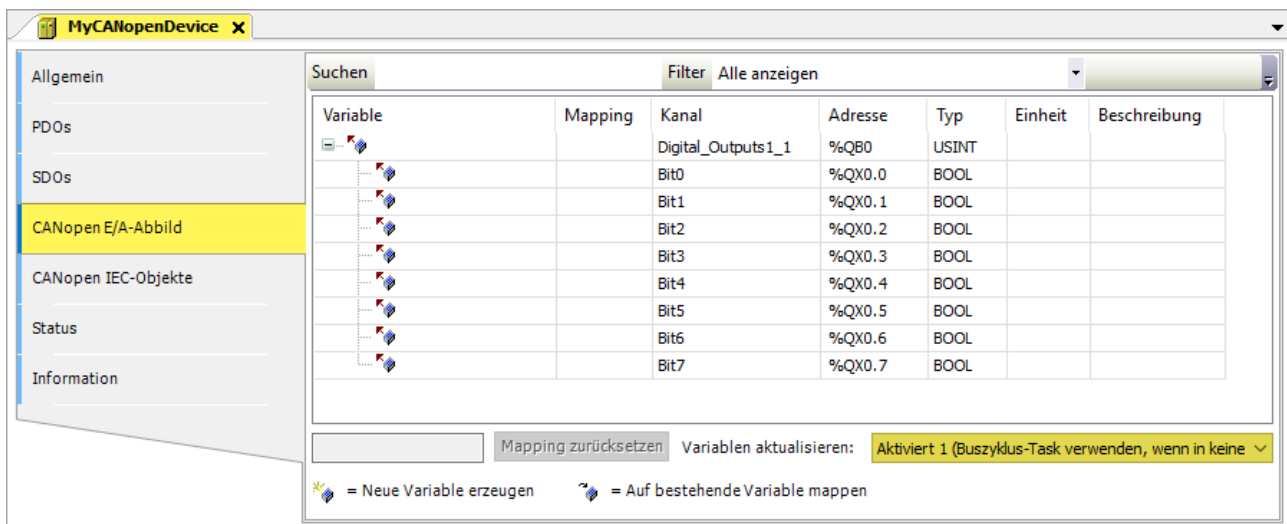


Figure 76: CODESYS - CAN bus master - CAN slave configuration

7.8.5.3 Program download and test

Open the communication settings for the two devices “PiXtendCAN_Master” and “PiXtendCAN_Slave” one after the other and select the corresponding Raspberry Pi devices to be downloaded.

You can then use the function "Multiple Download" in the main menu "Online" to simultaneously load the applications onto the corresponding controllers.

Note: As soon as several applications are in the project tree, you can right-click on the application -> "Set Active Application" to select or activate the desired application.

Go “Online” to the applications one after the other and start them.

If everything has been configured correctly and both devices are connected correctly, the CAN entries in the project tree are displayed in green.

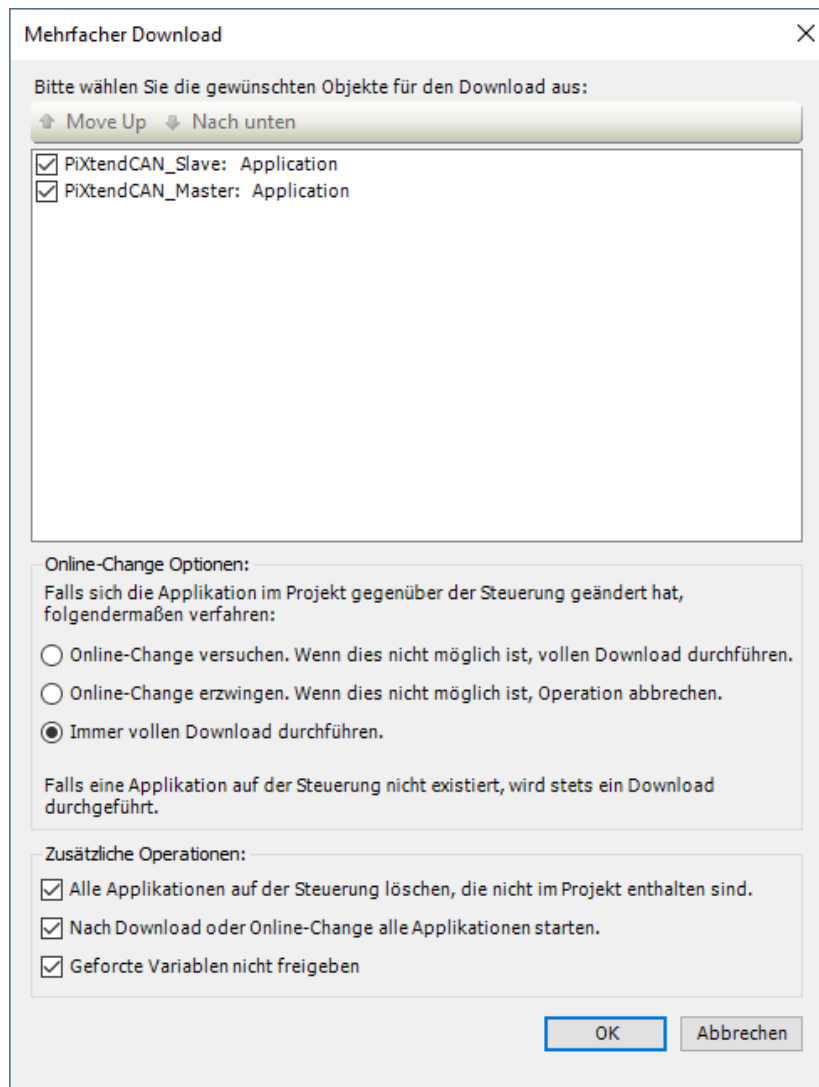


Figure 77: CODESYS - CAN bus test - Multiple download

Now open the I/O area of the “CANopen Master”, you change and write (Ctrl+F7) the values, then these values are automatically transferred from the master to the slave:

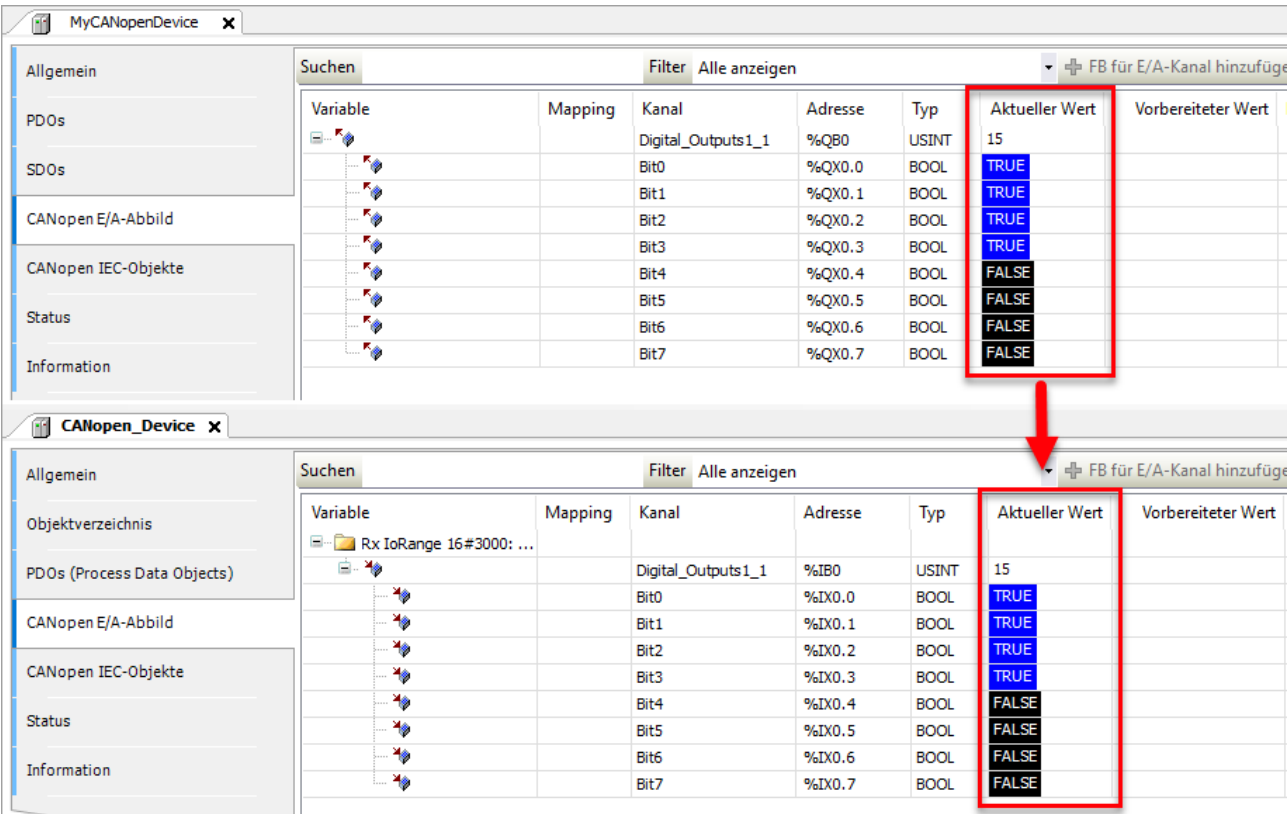


Figure 78: CODESYS - CAN bus test - Master and slave exchange data

Of course, the variable could be mapped and used for example directly in the main program.

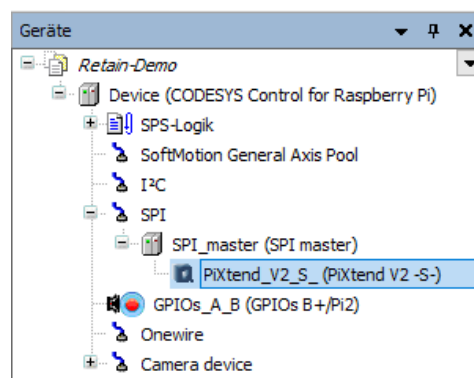
7.9. CODESYS – PiXtend Retain memory

The PiXtend V2 -S- has a 32-byte flash memory (64-bytes for PiXtend V2 -L-), which a CODESYS user can write to with any value. This chapter illustrates the use of the retain memory and shows which points must be observed. We use a PiXtend V2 -S-, you are welcome to use a PiXtend V2 -L-.

7.9.1. Preparation

As preparation for this example, you need a new empty CODESYS standard project and to have the SPI bus that is configured with one SPI master and one PiXtend V2 -S-.

In the Raspberry Pi GPIOs, pin 24 should be defined as an output and should be given the variable name “SPI_ENABLE” in the I/O mapping.



In the I/O mapping of the PiXtend V2 -S-, several variables are needed, these are listed in the following table.

Figure 79: CODESYS - Retain Demo – Device tree

| Folder | Channel | Variable | Description |
|-------------|--------------------|---------------------|---|
| Control | RetainDataEnable | xRetainDataEnable | Turn the Retain function on and off |
| State | Firmware | byFirmware | Firmware version of the micro-controller |
| State | Hardware | byHardware | Hardware revision of the PiXtend V2 -S- |
| State | ModelIn | byModel | Model number of the PiXtend V2 -S- |
| State | RetainCRCError | xRetainCRCError | Retain CRC error bit |
| State | RetainVoltageError | xRetainVoltageError | Retain power supply below 19 volts |
| State | Run | xRun | Micro-controller active signal |
| State | CRCHeaderInError | xCRCHeaderInError | CRC error detected in the SPI header data |
| State | CRCDataInError | xCRCDataInError | CRC error detected in SPI payload data |
| Retain Data | RetainDataOut | arRetainDataOut | Retain data to the micro-controller |
| Retain Data | RetainDataIn | arRetainDataIn | Retain data from the micro-controller |

The CODESYS project is now prepared, all necessary settings have been made.

7.9.2. Creating a program

In program block “PLC_PRG”, we need four new variables to control our test program. Create the following variables in the declaration section:

- xInit: BOOL
- xRetainInit: BOOL
- iRetainStep: INT
- xSetNewValue: BOOL

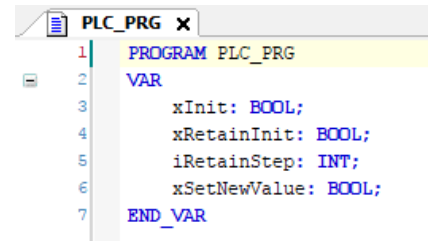


Figure 80: CODESYS - Retain - Variable declaration

Insert the following program in the program section:

```
IF xInit = FALSE THEN
    xInit := TRUE;
    SPI_ENABLE := TRUE;
END_IF

IF xRetainInit = FALSE THEN
    CASE iRetainStep OF
        0:
            IF xRun = TRUE AND byFirmware = 4 AND byHardware = 21 AND byModel = 83 AND
                xRetainCRCError = FALSE AND xRetainVoltageError = FALSE AND
                xCRCHeaderInError = FALSE AND xCRCDataInError = FALSE
            THEN
                iRetainStep := 1;
            END_IF
        1:
            arRetainDataOut[0] := arRetainDataIn[0];
            iRetainStep := 2;
        2:
            xRetainDataEnable := TRUE;
            iRetainStep := 3;
        3:
            xRetainInit := TRUE;
    END_CASE
END_IF

IF xSetNewValue THEN
    xSetNewValue := FALSE;
    arRetainDataOut[0] := arRetainDataOut[0] + 1;
END_IF
```

The numbers (4, 21 and 83) for the comparisons of byFirmware, byHardware and byModel must be replaced by the actual values of the presentPiXtend V2 board. These are only example values!

```

1  IF xInit = FALSE THEN
2      xInit := TRUE;
3      SPI_ENABLE := TRUE;
4  END_IF
5
6  //Retain init sequence
7  IF xRetainInit = FALSE THEN
8      CASE iRetainStep OF
9          0: //Startup check - We have the Run bit, now lets check
              //the PiXtend state
10             IF xRun = TRUE AND byFirmware = 4 AND byHardware = 21 AND byModel = 83 AND
11                xRetainCRCError = FALSE AND xRetainVoltageError = FALSE AND
12                xCRCHeaderInError = FALSE AND xCRCDataInError = FALSE
13            THEN
14                //All OK - go to next step
15                iRetainStep := 1;
16            END_IF
17
18
19            1: //Start - Get retain data from the micro-controller
20                arRetainDataOut[0] := arRetainDataIn[0];
21                //Go to next step - Enable retain data storage
22                iRetainStep := 2;
23
24            2: //Activate retain data function
25                xRetainDataEnable := TRUE;
26                iRetainStep := 3;
27
28            3: //Done - Retain data setup & restore complete
29                xRetainInit := TRUE;
30            END_CASE
31        END_IF
32
33        IF xSetNewValue THEN
34            xSetNewValue := FALSE;
35            //Increment the retain output byte 0 by 1
36            arRetainDataOut[0] := arRetainDataOut[0] + 1;
37        END_IF

```

Figure 81: CODESYS - Retain - Example program

Explanation of the program flow:

In the first section with the line “If xInit = False then” a very short initialization is carried out, only the SPI communication with “SPI_ENABLE := True” is switched on.

In the next section, existing retain data is read into CODESYS by means of a sequencer, provided that the check of various variables, see Preparation, is successful. In the first step, the first byte of the RetainDataIn array variable is written to the first byte of the RetainDataOut array variable and in third step the Retain function of the PiXtend V2 -S- is activated.

It is recommended to always enable the retain function as the last step after the “old” (previous) data has been restored.

We can use the last section to increment the first byte of the RetainDataOut array variable by one counter each by setting the xSetNewValue variable to True. We disconnect the power supply of the PiXtend V2-S-. After a restart we should find the value that was there before the restart in the RetainDataOut array variable.

Example script:

1. After the program start, arRetainDataOut[0] and arRetainDataIn[0] have the same value. In this example it is the number 13.

```
1: //Start - Get retain data from the micro-controller
   arRetainDataOut[0][13] := arRetainDataIn[0][13];
   //Go to next step - Enable retain data storage
   iRetainStep[3] := 2;
2: //Activate retain data function
   xRetainDataEnable TRUE := TRUE;
   iRetainStep[3] := 3;
```

Figure 82: CODESYS - Retain – Start value 13

2. After setting the variable xSetNewValue to true, the value in arRetainDataOut[0] increases by one to 14 .

```
1: //Start - Get retain data from the micro-controller
   arRetainDataOut[0][14] := arRetainDataIn[0][13];
   //Go to next step - Enable retain data storage
   iRetainStep[3] := 2;
2: //Activate retain data function
   xRetainDataEnable TRUE := TRUE;
   iRetainStep[3] := 3;
```

Figure 83: CODESYS - Retain – arRetainDataOut[0] increased by one to 14

The variable arRetainDataIn[0] still shows us the "old" value.

3. Pull the plug and disconnect the power supply of the PiXtend V2 -S-, wait briefly and restore the supply (power cycle).

4. After the restart, the old value was restored from the

flash memory of the micro-controller and immediately written to the first byte of the RetainDataOut array variable. From now on we can continue to work with this value.

```
1: //Start - Get retain data from the micro-controller
   arRetainDataOut[0][14] := arRetainDataIn[0][14];
   //Go to next step - Enable retain data storage
   iRetainStep[3] := 2;
2: //Activate retain data function
   xRetainDataEnable TRUE := TRUE;
   iRetainStep[3] := 3;
```

Figure 84: CODESYS - Retain – After a power cycle

7.9.3. Further Information

7.9.3.1 Improving data security

The retain data is stored in the micro-controller flash memory with a CRC and is read and checked at startup. It is useful to perform this check also in CODESYS to ensure the correctness of the persistent data.

For this purpose, a checksum (CRC) with 16 bits can be created. In CODESYS, these functions are available in the libraries CAA Memory and CmpChecksum or in the CODESYS Open Source library, OSCAT for short.

A 16-bit checksum can be split into two bytes and written to the last two bytes of the Retain memory. After rebooting, CODESYS allows you to check the restored values based on this checksum.

7.9.3.2 Working with structures

The outbound and inbound Retain data is provided as an array of 32 bytes. Each byte in this array can be addressed individually, allowing for easy looping and direct access.

If 16-bit, 32-bit or 64-bit values are stored in the Retain memory, please note that you have to assign each byte individually.

CODESYS offers two practical approaches to simplify these situations. Structures (Struct) can be used to organize and summarize data in CODESYS. If you combine a structure with a union data type, this is called an overlay and gives you the possibility to address the Retain data via a structure. This procedure has the advantage that you do not have to worry about splitting the individual bytes in the array.

Example:

A DWORD variable is four bytes in size. If we combine this variable in a Union data type with an array of four bytes in length, each value written to the DWORD variable is automatically split up among the four bytes of the array. This is possible because the DWORD variable and the 4-byte array occupy the same memory space in the CODESYS memory (overlapping).



```

TYPE uDword :
    UNION
        dwValue : DWORD;
        arValue : ARRAY[0..3] OF BYTE;
    END_UNION
END_TYPE
    
```

```

uValue.dwValue 2000000;
uValue.arValue[0] 128;
uValue.arValue[1] 132;
uValue.arValue[2] 30;
uValue.arValue[3] 0;
    
```

7.10. CODESYS – Shutting down the Raspberry Pi

CODESYS does not provide its own function to shut down, switch off and disconnect the Raspberry Pi "properly".

Shutting down the Raspberry Pi offers advantages in terms of data security, additionally the CODESYS own retain function can be used. The values of the CODESYS retain function are only written to the SD card during shutdown and are read in again at the next start. This is useful if you need more than 32 or 64 bytes of retain memory.

The following short example shows how to shut down the Raspberry Pi from CODESYS.

Add the two new libraries "SysProcess" and "SysTypes2 Interfaces" to your project 2. When adding them, use the search bar in the library manager and enter the names of the libraries directly. To make the system libraries visible, click on the plus symbol to add library.

In the "SysProcess" library, there is the function "SysProcessExecuteCommand" which we need to perform a shutdown.

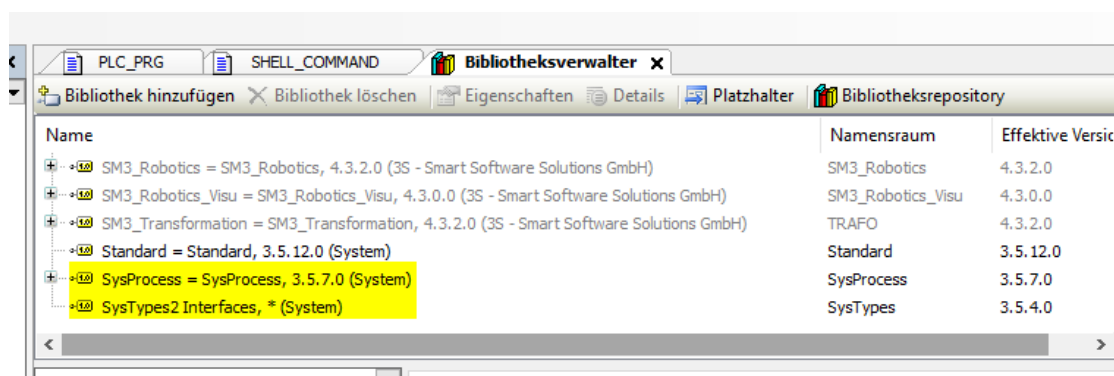


Figure 85: CODESYS - Shutdown - Library Manager

When adding libraries, the search bar (2) at the top of the dialog is very helpful, activate the extended view (1). Click on the bold name of the library (3) to select it.

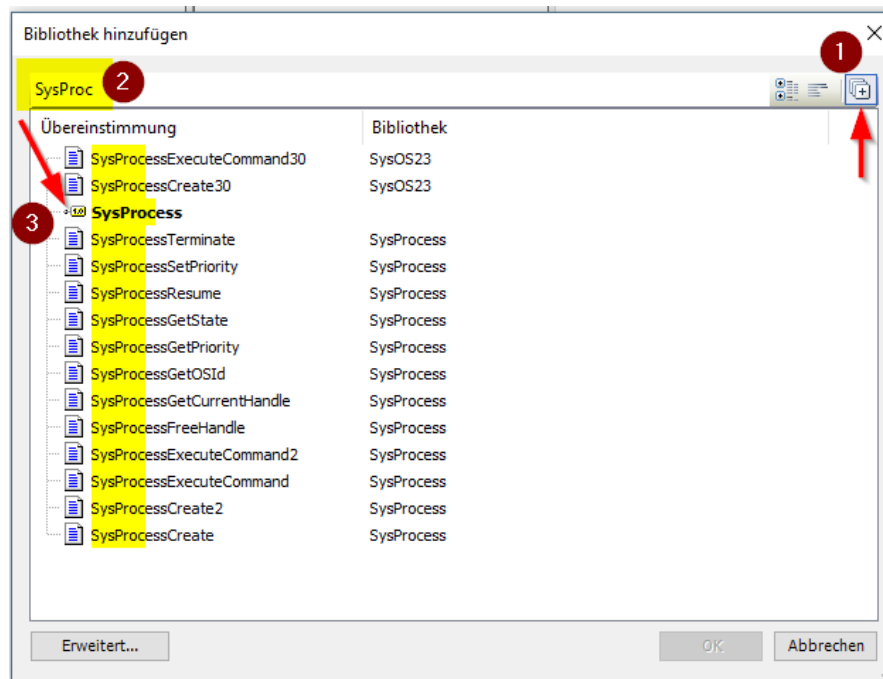


Figure 86: CODESYS - Shutdown - Adding libraries

If both libraries were added to the project, create a new program block and additionally four new variables.

Four variables with the following types are required:

- 1 x BOOL
- 1 x DINT
- 1 x SysTypes.RTS_IEC_RESULT
- 1 x STRING

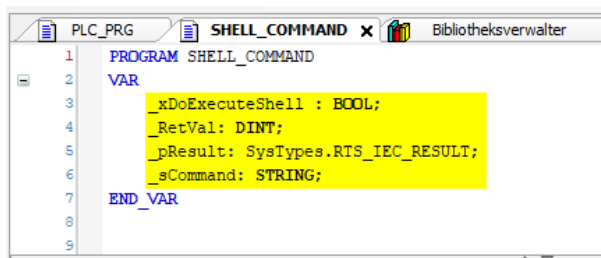


Figure 87: CODESYS - Shutdown – Variables

The execution of the function “SysProcessExecuteCommand” can be encapsulated with an IF query in the new program block. It is important that we execute the shutdown command only once. If the execution is repeated, this can lead to problems with CODESYS.

The command for the operating system is:

```
sudo shutdown -h now & disown
```

The “&” character and the word “disown” are not typographical errors, they are part of the command and must be present.

If the boolean variable is TRUE, we go into the IF query, reset the boolean, put our desired command into the string variable and pass everything to the function “SysProcessExecuteCommand”. We pass the result of the function as a pointer with the operator ADR.

This way the Raspberry Pi can be easily shutdown. After executing the command, wait shortly, similar to Windows, until all services and CODESYS have been stopped. Then the operating system is in a safe state and the power supply can be safely disconnected.

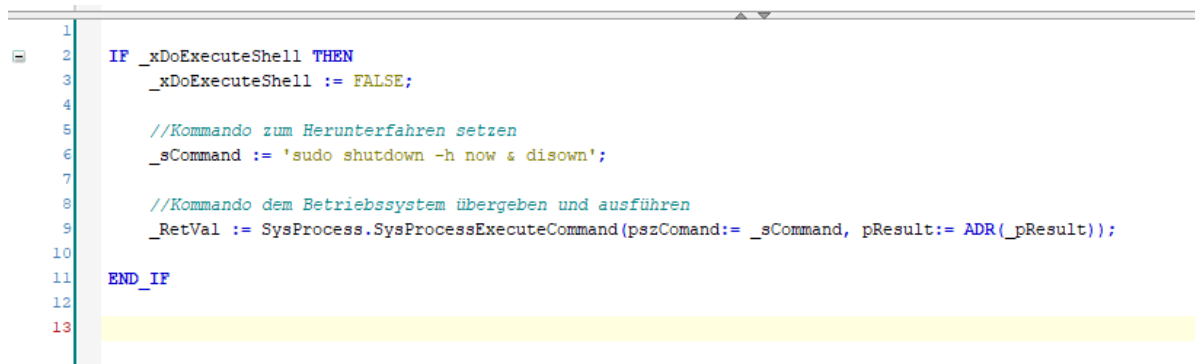


Figure 88: CODESYS - Shutdown - Example program

7.11. PiXtend V2 -S- Micro-controller

7.11.1. SPI device parameter

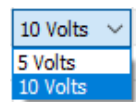
The PiXtend V2 -S- SPI device in CODESYS has some settings that can be set in the tab SPI devices parameters before compiling. These parameters can no longer be changed during runtime.

| Parameter | Typ | Wert | Standardwert | Einheit |
|---------------------------------|---------------------|----------|--------------|-----------|
| 5 Volts/10 Volts Jumper Setting | | | | |
| JumperSettingAI0 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| JumperSettingAI1 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| GPIO Configuration | | | | |
| GPIO0Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO1Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO2Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO3Ctrl | Enumeration of BYTE | Input | Input | |
| GIOPullupsEnable | Enumeration of BOOL | Off | Off | |
| Microcontroller Settings | | | | |
| WatchdogEnable | Enumeration of BYTE | Disabled | Disabled | |
| StateLEDDisable | Enumeration of BOOL | False | False | |

Figure 89: CODESYS PiXtend V2 -S- - SPI devices parameter tab

7.11.1.1 5 volt/10 volt jumper setting

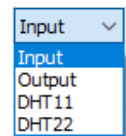
This parameter is used to tell the CODESYS driver for the PiXtend whether the analog signals at AnalogIn0 and AnalogIn1 are in the range of 5 volts (jumper set) or 10 volts (jumper not set, default setting). Depending on the settings, the driver uses a different conversion factor to calculate the voltage values. The PiXtend does not detect if the jumper is set. The setting in CODESYS must be compared with the actual (physically) existing jumpers on PiXtend to make a corresponding setting for this parameter.



7.11.1.2 GPIO Configuration

The GPIO configuration defines which of the four following settings each individual GPIO should have.

- Input – default setting
- Output
- DHT11 - (input for temperature and humidity measurement)
- DHT22 - (input for temperature and humidity measurement)

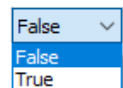
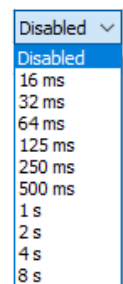


Furthermore, the GPIO configuration offers the possibility to activate the GPIO PullUps. If this setting is On, the GPIO PullUps can be activated by setting the respective GPIO output, while a GPIO is configured as an input.

7.11.1.3 Micro-controller settings

The PiXtend V2 -S- micro-controller offers two settings that the user can change.

1. WatchdogEnable
 With this setting, the watchdog of the micro-controller can be switched on and off. When set to Disabled, the watchdog is off (inactive), selecting a corresponding waiting time activates the watchdog. Times from 16 milliseconds up to 8 seconds are available for selection.
2. StateLEDDisable
 The PiXtend V2 -S- has a status LED, which signals an error when a problem occurs in the micro-controller. If necessary, this LED can be deactivated. The setting False means the LED is active (on), setting to True deactivates the LED.



7.11.2. I/O error

In this chapter you will find a list of all I/Os that are available in CODESYS for the PiXtend V2 -S-, together with the data type and a short description.

| Name | Type | Data type | Description |
|---------------------|--------|-----------|---|
| Control | | | |
| PWM0Ctrl1 | Output | WORD | The effect of the PWM0Ctrl1 value depends on the selected mode in PWM0Ctrl0. This value applies to channels A and B of PWM 0. |
| PWM1Ctrl1 | Output | BYTE | The effect of the PWM1Ctrl1 value depends on the selected mode in PWM1Ctrl0. This value applies to channels A and B of PWM 1. |
| PWM0Ctrl0 | Output | BYTE | PWM 0 Control 0 - enables the setting of PWM mode, channel enable and prescaler. PWM 0 uses 16-bit values. |
| PWM1Ctrl0 | Output | BYTE | PWM 1 Control 0 - enables the setting of PWM mode, channel enable and prescaler. PWM 1 uses 8-bit values. |
| GPIODebounce01 | Output | BYTE | Debounce of GPIO inputs 0 and 1, value * bus cycle interval = debounce time. |
| GPIODebounce23 | Output | BYTE | Debounce of GPIO inputs 2 and 3, value * bus cycle interval = debounce time. |
| DigitalInDebounce01 | Output | BYTE | Debounce of the digital inputs 0 and 1, value * bus cycle interval = debounce time. |
| DigitalInDebounce23 | Output | BYTE | Debounce of the digital inputs 2 and 3, value * bus cycle interval = debounce time. |
| DigitalInDebounce45 | Output | BYTE | Debounce of the digital inputs 4 and 5, value * bus cycle interval = debounce time. |
| DigitalInDebounce67 | Output | BYTE | Debounce of the digital inputs 6 and 7, value * bus cycle interval = debounce time. |
| SafeState | Output | bit | Puts the micro-controller in a safe state should the controller require a restart or shutdown. True = Safe state, False = Remain On. |
| RetainDataEnable | Output | bit | Switch on retain data storage in the micro-controller. TRUE = On, FALSE = Off. |
| RetainCopy | Output | bit | If TRUE, then the current RetainDataOut bytes in the micro-controller are copied to RetainDataIn. If FALSE, the data currently stored in the flash is sent to RetainDataIn. |
| | | | |
| State | | | |
| Firmware | Input | BYTE | Firmware version of the micro-controller on the PiXtend V2 -S- board. |
| Hardware | Input | BYTE | PiXtend V2 -S- Board revision, 20 = 2.0, 21 = 2.1, etc. |
| ModelIn | Input | BYTE | Read model number from PiXtend V2 -S-. |
| Error | Input | BYTE | Error byte of the micro-controller See status bytes. |
| RetainCRCErr | Input | bit | The CRC of the retain memory in the micro-controller is wrong, the stored data may be faulty. |
| RetainVoltageErr | Input | bit | If TRUE, then the Retain function cannot be used, the supply voltage is below 19 volts. |

| | | | |
|------------------|--------|------|--|
| Run | Input | bit | The communication with the micro-controller is ok. |
| BusCycleError | Input | bit | The cycle time (interval) of the bus cycle task is too fast. |
| CRCHeaderInError | Input | bit | A CRC error was detected in the SPI header, there is a communication error. |
| CRCDataInError | Input | bit | A CRC error was found in the PiXtend V2 -S- SPI data, there is no usable data. |
| ModelInError | Input | bit | Model error detected, the set PiXtend V2 -S- model in CODESYS V3 and the actual hardware do not match. |
| Sensor0Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 0. The temperature and humidity values are invalid. |
| Sensor1Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 1. The temperature and humidity values are invalid. |
| Sensor2Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 2. The temperature and humidity values are invalid. |
| Sensor3Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 3. The temperature and humidity values are invalid. |
| | | | |
| Analog inputs | | | |
| AnalogIn0 | Input | REAL | Analog input 0 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn0Raw | Input | WORD | Analog input 0 raw value, actual 16 bit value from ADC. |
| AnalogIn1 | Input | REAL | Analog input 1 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn1Raw | Input | WORD | Analog input 1 raw value, actual 16 bit value from ADC. |
| | | | |
| Digital inputs | | | |
| DigitalInputs | Input | BYTE | Digital inputs 0-7, bit access is possible. |
| GPIOInputs | Input | BYTE | GPIO inputs 0-3 as byte, bit access is possible. This function depends on the settings on the Parameter tab. |
| | | | |
| Digital outputs | | | |
| DigitalOutputs | Output | BYTE | Digital outputs 0-3 as byte, bit access is possible. |
| GPIOOutputs | Output | BYTE | GPIO outputs 0-3 as byte, bit access is possible. This function depends on the settings on the Parameter tab. |
| RelayOutputs | Output | BYTE | Relay outputs 0-3 as byte, bit access is possible. |
| | | | |
| PWM Outputs | | | |

| | | | |
|--------------------|---|-----------------------|---|
| PWM0A | Output | WORD | PWM 0, channel A output value. The actual PWM behavior depends on the mode setting in byte PWM0Ctrl0. |
| PWM0B | Output | WORD | PWM 0, channel B output value. The actual PWM behavior depends on the mode setting in byte PWM0Ctrl0. |
| PWM1A | Output | BYTE | PWM 1, channel A output value. The actual PWM behavior depends on the mode setting in byte PWM1Ctrl0. |
| PWM1B | Output | BYTE | PWM 1, channel B output value. The actual PWM behavior depends on the mode setting in byte PWM1Ctrl0. |
| | | | |
| Humidity inputs | Air humidity value of a sensor connected to GPIO 0-3. The sensor option must be activated in the Parameter tab. The unit is %RH. | | |
| Humid0 | Input | REAL | |
| Humid1 | Input | REAL | |
| Humid2 | Input | REAL | |
| Humid3 | Input | REAL | |
| | | | |
| Temperature inputs | Temperature value of a sensor connected to GPIO 0-3. The sensor option must be activated in the Parameter tab. The unit is degrees celsius (°C). | | |
| Temp0 | Input | REAL | |
| Temp1 | Input | REAL | |
| Temp2 | Input | REAL | |
| Temp3 | Input | REAL | |
| | | | |
| Retain Data | The retain data can be used to save up to 32 bytes in the memory of the micro-controller in case of a power failure. After a restart, this data can be restored or read back. | | |
| RetainDataOut | Output | ARRAY[0 - 31] OF BYTE | |
| RetainDataIn | Input | ARRAY[0 - 31] OF BYTE | |

Table 3: CODESYS I/O Overview for PiXtend V2 -S-

7.12. PiXtend V2 -L- as master

7.12.1. SPI device parameter

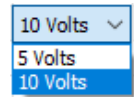
The PiXtend V2 -L- SPI device in CODESYS has some settings that can be set in the tab SPI devices parameters before compiling. These parameters cannot be changed later.

| Parameter | Typ | Wert | Standar... | Einheit |
|---------------------------------|---------------------|----------|------------|-----------|
| 5 Volts/10 Volts Jumper Setting | | | | |
| JumperSettingAI0 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| JumperSettingAI1 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| JumperSettingAI2 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| JumperSettingAI3 | Enumeration of BOOL | 10 Volts | 10 Volts | Volts [V] |
| GPIO Configuration | | | | |
| GPIO0Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO1Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO2Ctrl | Enumeration of BYTE | Input | Input | |
| GPIO3Ctrl | Enumeration of BYTE | Input | Input | |
| GPIOPullupsEnable | Enumeration of BOOL | Off | Off | |
| Microcontroller Settings | | | | |
| WatchdogEnable | Enumeration of BYTE | Disabled | Disabled | |
| StateLEDDisable | Enumeration of BOOL | False | False | |

Figure 90: CODESYS PiXtend V2 -L- - SPI devices parameter tab

7.12.1.1 5-volt/10-volt jumper setting

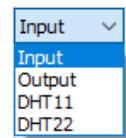
This parameter is used to tell the CODESYS driver for the PiXtend whether the analog signals at AnalogIn0 to AnalogIn3 are in the range of 5 volts (jumper set) or 10 volts (jumper not set, default setting). Depending on the settings, the driver uses a different conversion factor to calculate the voltage values. The PiXtend does not detect if the jumper is set. The user must compare the setting in CODESYS with the actual (physically) existing jumpers on the PiXtend and make a corresponding setting for this parameter.



7.12.1.2 GPIO Configuration

The GPIO configuration allows you to specify which of the four settings each individual GPIO should have:

- Input – default setting
- Output
- DHT11 - (input for temperature and humidity measurement)
- DHT22 - (input for temperature and humidity measurement)

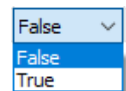
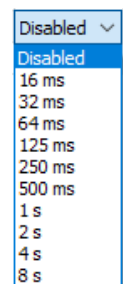


Furthermore, the GPIO configuration offers the possibility to activate the GPIO PullUps. If this setting is On, the GPIO PullUps can be activated by setting the respective GPIO output, while a GPIO is configured as an input.

7.12.1.3 Micro-controller settings

The PiXtend V2 -L- micro-controller offers two settings that the user can change.

1. WatchdogEnable
With this setting, the watchdog of the micro-controller can be switched on and off. When set to Disabled, the watchdog is off (inactive), selecting a corresponding waiting time activates the watchdog. Times from 16 milliseconds up to 8 seconds are available for selection.
2. StateLEDDisable
The PiXtend V2 -L- has a status LED, which signals an error when a problem occurs in the micro-controller. If necessary, this LED can be deactivated. The setting False means the LED is active (on), setting to True deactivates the LED.



7.12.2. I/O overview

In this chapter you will find a list of all I/Os that are available in CODESYS for the PiXtend V2 -L-, together with the data type and a short description.

| Name | Type | Data type | Description |
|-----------------------|--------|-----------|---|
| Control | | | |
| PWM0Ctrl1 | Output | WORD | The effect of the PWM0Ctrl1 value depends on the selected mode in PWM0Ctrl0. This value applies to channels A and B of PWM 0. |
| PWM1Ctrl1 | Output | WORD | The effect of the PWM1Ctrl1 value depends on the selected mode in PWM1Ctrl0. This value applies to channels A and B of PWM 1. |
| PWM2Ctrl1 | Output | WORD | The effect of the PWM2Ctrl1 value depends on the selected mode in PWM2Ctrl0. This value applies to channels A and B of PWM 2. |
| PWM0Ctrl0 | Output | BYTE | PWM 0 Control 0 - enables the setting of PWM mode, channel enable and prescaler. PWM 0 uses 16-bit values. |
| PWM1Ctrl0 | Output | BYTE | PWM 1 Control 0 - enables the setting of PWM mode, channel enable and prescaler. PWM 1 uses 16-bit values. |
| PWM2Ctrl0 | Output | BYTE | PWM 2 Control 0 - enables the setting of PWM mode, channel enable and prescaler. PWM 2 uses 16-bit values. |
| GPIODebounce01 | Output | BYTE | Debounce of GPIO inputs 0 and 1, value * bus cycle interval = debounce time. |
| GPIODebounce23 | Output | BYTE | Debounce of GPIO inputs 2 and 3, value * bus cycle interval = debounce time. |
| DigitalInDebounce01 | Output | BYTE | Debounce of the digital inputs 0 and 1, value * bus cycle interval = debounce time. |
| DigitalInDebounce23 | Output | BYTE | Debounce of the digital inputs 2 and 3, value * bus cycle interval = debounce time. |
| DigitalInDebounce45 | Output | BYTE | Debounce of the digital inputs 4 and 5, value * bus cycle interval = debounce time. |
| DigitalInDebounce67 | Output | BYTE | Debounce of the digital inputs 6 and 7, value * bus cycle interval = debounce time. |
| DigitalInDebounce89 | Output | BYTE | Debounce of the digital inputs 8 and 9, value * bus cycle interval = debounce time. |
| DigitalInDebounce1011 | Output | BYTE | Debounce of the digital inputs 10 and 11, value * bus cycle interval = debounce time. |
| DigitalInDebounce1213 | Output | BYTE | Debounce of the digital inputs 12 and 13, value * bus cycle interval = debounce time. |
| DigitalInDebounce1415 | Output | BYTE | Debounce of the digital inputs 14 and 15, value * bus cycle interval = debounce time. |
| SafeState | Output | bit | Puts the micro-controller in a safe state should the controller require a restart or shutdown. True = Safe state, False = Remain On. |
| RetainDataEnable | Output | bit | Switch on retain data storage in the micro-controller. TRUE = On, FALSE = Off. |
| RetainCopy | Output | bit | If TRUE, then the current RetainDataOut bytes in the micro-controller are copied to RetainDataIn. If FALSE, the data currently stored in the flash is sent to RetainDataIn. |
| | | | |
| State | | | |

| | | | |
|--------------------|-------|------|--|
| Firmware | Input | BYTE | Firmware version of the micro-controller on the PiXtend V2 -L- board. |
| Hardware | Input | BYTE | PiXtend V2 -L- Board revision, 20 = 2.0, 21 = 2.1, etc. |
| ModelIn | Input | BYTE | Read model number from PiXtend V2 -L-. |
| Error | Input | BYTE | Error byte of the micro-controller See status bytes. |
| RetainCRCError | Input | bit | The CRC of the retain memory in the micro-controller is wrong, the stored data may be faulty. |
| RetainVoltageError | Input | bit | If TRUE, then the Retain function cannot be used, the supply voltage is below 19 volts. |
| Run | Input | bit | The communication with the micro-controller is ok. |
| BusCycleError | Input | bit | The cycle time (interval) of the bus cycle task is too fast. |
| CRCHeaderInError | Input | bit | A CRC error was detected in the SPI header, there is a communication error. |
| CRCDataInError | Input | bit | A CRC error was found in the PiXtend V2 -L- SPI data, there is no usable data. |
| ModelInError | Input | bit | Model error detected, the set PiXtend V2 -L- model in CODESYS V3 and the actual hardware do not match. |
| I2CError | Input | bit | An error was detected on the I ² C bus or the micro-controller could not find a slave MC. |
| Sensor0Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 0. The temperature and humidity values are invalid. |
| Sensor1Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 1. The temperature and humidity values are invalid. |
| Sensor2Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 2. The temperature and humidity values are invalid. |
| Sensor3Error | Input | bit | If TRUE, then the micro-controller could not read data from the sensor at GPIO 3. The temperature and humidity values are invalid. |
| | | | |
| Analog inputs | | | |
| AnalogIn0 | Input | REAL | Analog input 0 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn0Raw | Input | WORD | Analog input 0 raw value, actual 16 bit value from ADC. |
| AnalogIn1 | Input | REAL | Analog input 1 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn1Raw | Input | WORD | Analog input 1 raw value, actual 16 bit value from ADC. |
| AnalogIn2 | Input | REAL | Analog input 2 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn2Raw | Input | WORD | Analog input 2 raw value, actual 16 bit value from ADC. |
| AnalogIn3 | Input | REAL | Analog input 3 measured in volts [V], range is 0 volts to +10 volts. |
| AnalogIn3Raw | Input | WORD | Analog input 3 raw value, actual 16 bit value from ADC. |

| | | | |
|-----------------|--|------|---|
| AnalogIn4 | Input | REAL | Analog input 4 measured in milliamps [mA], range is 0 mA to +20 mA. |
| AnalogIn4Raw | Input | WORD | Analog input 4 raw value, actual 16 bit value from ADC. |
| AnalogIn5 | Input | REAL | Analog input 5 measured in milliamps [mA], range is 0 mA to +20 mA. |
| AnalogIn5Raw | Input | WORD | Analog input 5 raw value, actual 16 bit value from ADC. |
| | | | |
| Digital inputs | | | |
| DigitalInputs0 | Input | BYTE | Digital inputs 0-7, bit access is possible. |
| DigitalInputs1 | Input | BYTE | Digital inputs 8-15, bit access is possible. |
| GPIOInputs | Input | BYTE | GPIO inputs 0-3 as byte, bit access is possible. This function depends on the settings on the Parameter tab. |
| | | | |
| Digital outputs | | | |
| DigitalOutputs0 | Output | BYTE | Digital outputs 0-7 as byte, bit access is possible. |
| DigitalOutputs1 | Output | BYTE | Digital outputs 8-11 as byte, bit access is possible. |
| GPIOOutputs | Output | BYTE | GPIO outputs 0-3 as byte, bit access is possible. This function depends on the settings on the Parameter tab. |
| RelayOutputs | Output | BYTE | Relay outputs 0-3 as byte, bit access is possible. |
| | | | |
| PWM Outputs | | | |
| PWM0A | Output | WORD | PWM 0, channel A output value. The actual PWM behavior depends on the mode setting in byte PWM0Ctrl0. |
| PWM0B | Output | WORD | PWM 0, channel B output value. The actual PWM behavior depends on the mode setting in byte PWM0Ctrl0. |
| PWM1A | Output | WORD | PWM 1, channel A output value. The actual PWM behavior depends on the mode setting in byte PWM1Ctrl0. |
| PWM1B | Output | WORD | PWM 1, channel B output value. The actual PWM behavior depends on the mode setting in byte PWM1Ctrl0. |
| PWM2A | Output | WORD | PWM 2, channel A output value. The actual PWM behavior depends on the mode setting in byte PWM2Ctrl0. |
| PWM2B | Output | WORD | PWM 2, channel B output value. The actual PWM behavior depends on the mode setting in byte PWM2Ctrl0. |
| | | | |
| Humidity inputs | Air humidity value of a sensor connected to GPIO 0-3. The sensor option must be activated in the Parameter tab. The unit is %RH. | | |
| Humid0 | Input | REAL | |
| Humid1 | Input | REAL | |

| | | | |
|--------------------|---|-----------------------|--|
| Humid2 | Input | REAL | |
| Humid3 | Input | REAL | |
| | | | |
| Temperature inputs | Temperature value of a sensor connected to GPIO 0-3. The sensor option must be activated in the Parameter tab. The unit is degrees Celsius (°C). | | |
| Temp0 | Input | REAL | |
| Temp1 | Input | REAL | |
| Temp2 | Input | REAL | |
| Temp3 | Input | REAL | |
| | | | |
| Retain Data | The retain data can be used to save up to 64 bytes in the memory of the micro-controller in case of a power failure. After a restart, this data can be restored or read back. | | |
| RetainDataOut | Output | ARRAY[0 - 63] OF BYTE | |
| RetainDataIn | Input | ARRAY[0 - 63] OF BYTE | |

Table 4: CODESYS I/O Overview for PiXtend V2 -L-

7.13. FAQs – Frequently asked questions and troubleshooting

7.13.1. CODESYS Raspberry Pi Runtime and PiXtend V2

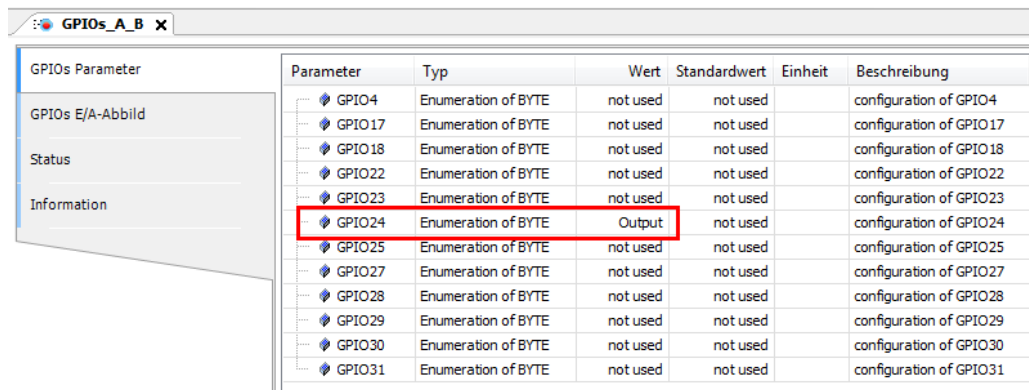
Problem

The green LED “+5V” lights up and the Raspberry Pi boots normally. However, the data exchange between the Raspberry Pi and PiXtend V2 does not seem to work (relays or outputs cannot be set).

Troubleshooting

Check whether the “SPI_EN” switch is set to “ON”. This makes communication between the Raspberry Pi and PiXtend V2 possible.

If you are working with CODESYS and have created your own project, always ensure that the GPIO24 (of the Raspberry Pi) is configured as an output and set to logical “1” (TRUE) (see figure 91). The GPIO24 is the signal which is transmitted via the “SPI_EN” switch and activates the data transmission.



The screenshot shows the 'GPIOs_A_B' window in CODESYS. On the left, there is a sidebar with 'GPIOs Parameter', 'GPIOs E/A-Abbild', 'Status', and 'Information'. The main area displays a table of GPIO configurations. The row for GPIO24 is highlighted with a red box, indicating it is configured as an output.

| Parameter | Typ | Wert | Standardwert | Einheit | Beschreibung |
|-----------|---------------------|----------|--------------|---------|-------------------------|
| GPIO4 | Enumeration of BYTE | not used | not used | | configuration of GPIO4 |
| GPIO17 | Enumeration of BYTE | not used | not used | | configuration of GPIO17 |
| GPIO18 | Enumeration of BYTE | not used | not used | | configuration of GPIO18 |
| GPIO22 | Enumeration of BYTE | not used | not used | | configuration of GPIO22 |
| GPIO23 | Enumeration of BYTE | not used | not used | | configuration of GPIO23 |
| GPIO24 | Enumeration of BYTE | Output | not used | | configuration of GPIO24 |
| GPIO25 | Enumeration of BYTE | not used | not used | | configuration of GPIO25 |
| GPIO27 | Enumeration of BYTE | not used | not used | | configuration of GPIO27 |
| GPIO28 | Enumeration of BYTE | not used | not used | | configuration of GPIO28 |
| GPIO29 | Enumeration of BYTE | not used | not used | | configuration of GPIO29 |
| GPIO30 | Enumeration of BYTE | not used | not used | | configuration of GPIO30 |
| GPIO31 | Enumeration of BYTE | not used | not used | | configuration of GPIO31 |

Figure 91: CODESYS - GPIO24 as an output

7.13.2. Serial communication

7.13.2.1 Not all characters are transmitted or arrive one after the other

If too many characters are sent, it may be that not all characters can be transmitted at once. A cycle is then not enough to send all characters. Increasing the baud rate may help.

Perform the following calculation to estimate how many characters can be transmitted:

Number of possible characters = (cycle time * baud rate) / bits to be transmitted

- ▶ the cycle time is expressed in seconds (10 ms = 0.010 s)
- ▶ the bits to be transmitted also contain, for example, a start or stop bit
- ▶ a character consists of 8 bits (= 1 byte)

7.13.2.2 Why are the GPIOs 18 and 22 switched in the program?

This applies only to PiXtend V2 -L-!

| | | |
|--------------|----|--------------------|
| GPIO_18:TRUE | => | RS485 |
| FALSE | => | RS232 |
| GPIO_22:TRUE | => | RS485 send mode |
| FALSE | => | RS485 receive mode |

The switching of sending and receiving for the RS485 has to be carried out manually, since the Raspberry Pi does not have the necessary pins (RTS / CTS).

NOTICE

The PiXtend V2 -L- also has an automatic send/receive mode switch, which is preset as a factory default. GPIO 22 is not needed in this case.

7.13.2.3 Scrolling the table does not work properly

If the mouse pointer is over the table, scrolling may not work properly. Just move the mouse pointer away from the table.

7.13.2.4 The page/user interface is not displayed correctly

The page is a canvas element that may not be displayed correctly. This can be remedied by reloading the page (F5) and using a browser that can handle HTML5.

7.13.2.5 The first message is not transmitted correctly

When two PiXtend V2 are connected, the first message sent is not transmitted correctly, but only the first character. This behavior can occur when both PiXtend V2 and Raspberry Pi devices are simultaneously started. The reason for this is the kernel message "Uncompressing Linux..." which is always displayed on the serial interface at startup, even if it is deactivated. This is a problem of the Raspbian Jessie version from 21 November 2015, which should be fixed with the next version.

7.13.2.6 Why is the "Auto send" function not available with RS485?

This applies only to PiXtend V2 -L-!

This function is prepared in the program but is not used. RS485 is a bus protocol and only one participant may transmit at a time. If the "auto send" function is used and more than one slave is transmitting at the same time, a collision may occur, and data will not be transmitted correctly.

7.13.2.7 Not all baud rates are displayed

In this example, only some baud rates in the range of 9600 to 115200 are used. If other baud rates are required, these must also be entered into the program (in the text list "Baud rates" as well as in the main program in "state 5").

If the selection list is not displayed correctly, changing the aspect ratio of the window can help or a reload of the page.

pxdev – Linux Tools and Library

This manual describes all necessary steps to install and run pxdev, the Linux development tools for the PiXtend V2 system (www.pixtend.de). For a manual installation, refer to chapter 8.4 Manual installation of pxdev.

Alternatively, you can start with our prepared SD card images. We recommend this option for beginners and for the initial installation of the PiXtend V2 - chapter: 8.3 Using the PiXtend V2 basic image.

pxdev contains the following components:

pixtend - the PiXtend/PiXtend V2 C-Library enables access to PiXtend V2 I/O hardware. The data exchange between the micro-controller and the Raspberry Pi takes place via the SPI (Serial Peripheral Interface). The wiringPi library is used for this. (<https://github.com/WiringPi/WiringPi.git>)

pixtendtool2s - A command line tool to access the PiXtend V2 -S- I/O hardware and configuration bytes using simple console commands.

pxauto2s – This is a simple console application with a graphical user interface. PiXtend V2 -S- can be continuously monitored. The application is suitable for fast installation and I/O tests.

pixtendtool2l - A command line tool to access the PiXtend V2 -L- I/O hardware and configuration bytes using simple console commands.

pxauto2l – This is a simple console application with a graphical user interface. PiXtend V2 -L- can be continuously monitored. The application is suitable for fast installation and I/O tests.

8.1. Note

8.1.1. Usage of pxtendtool2(s/l) - Single commands for PiXtend V2

The pxtendtool2(s/l) provides direct access to the PiXtend V2 hardware from the Linux command line. It is therefore suitable, among other things, for the following applications:

- Setting up individual commands via the command line: locally or also via SSH remote access over the network, e.g. to query outputs and inputs
- Execute pxtendtool2s commands from your own shell scripts
- Cyclic execution of pxtendtool2(s/l) commands to periodically log data and provide it for other services, e.g. a MySQL database.
- It be used as a reference for using the PiXtend/PiXtend V2 C library in “manual mode” (single commands)
- It can be adapted to your own needs

8.1.2. Usage of pxauto2(s/l) - GUI for PiXtend V2

The pxauto2(s/l) tool has a graphical interface and is suitable for the quick installation of the PiXtend V2 hardware. The states of the I/Os can be cyclically monitored.

- pxauto2(s/l) can be used both locally and remotely, i.e. with a SSH terminal
- I/O tests of connected hardware, sensors and actuators
- Testing the functionality of the PiXtend V2
- Basis for customized projects with GUI (Graphical User Interface)
- It be used as a reference for using the PiXtend V2 C library in “auto mode” (cyclic process image in C)

8.1.3. Usage of the PiXtend V2 C-Library

The PiXtend V2 C-library is used by all previously listed programs and forms the interface between the Raspberry Pi and the micro-controller used on the PiXtend V2. Changes to the library are usually not necessary.

8.2. Requirements

Required software

- PiXtend V2 SD image “Basic” (C / Python / Node-RED) (recommended!)
Download available at <https://www.pixtend.de>

or:

- Current Raspberry Pi OS Linux distribution on an SD card
- wiringPi (<https://github.com/WiringPi/WiringPi.git>)
- ncurses libraries libncurses5-dev libncursesw5-dev
- Optional SSH client (e.g. putty.exe – www.putty.org)

Required hardware

PiXtend V2 Board (www.pixtend.de)

Raspberry Pi Model B+ / 2 B / 3 B / 3 B+ / 4 B

8.3. Using the PiXtend V2 basic image

The easiest way for first tests and further work with pxdev is to use our pre-installed SD card and SD card image. The latest image can be downloaded free of charge at any time from our website <https://www.pixtend.de>.

See chapter 6.5 Preparing the SD card for your Raspberry Pi for instructions on how to transfer the image to an SD card

After booting, change to the pxdev folder with the following command:

```
cd pxdev
```

Or to the folder of the “pixtendtools2s” program:

```
cd pxdev/PiXtend_V2/pixtendtool2s/
```

For the PiXtend V2 -L- to the folder of the “pixtendtools2l” program:

```
cd pxdev/PiXtend_V2/pixtendtool2l/
```

Skip chapter 8.4 Manual installation of pxdev and continue directly with using the tools or the library in chapter 8.5 Using pixtendtool 2(S/L)

8.4. Manuel installation of pxdev

8.4.1. Preparation and installation

The latest version of the pxdev-package can be downloaded as a .zip-archive from our homepage in the download section or checked out from the pxdev repository with GIT. GIT is used for version management and allows the download complete repositories from the Internet with simple commands. We recommend the following procedure:

First enter the following command:

```
sudo apt-get update
```

in the Raspberry Pi command line to update the package manager database.

Pxdev internally uses the WiringPi libraries (<https://github.com/WiringPi/WiringPi> <https://github.com/WiringPi/WiringPi>) as a basis.

Install the WiringPi libraries on Raspberry Pi OS or later from Github:

```
sudo apt-get install git-core
cd ~
git clone https://github.com/WiringPi/WiringPi.git
cd WiringPi
./build
gpio -v
```

The WiringPi library should report version 2.52 or later.

pxauto2(s/l) uses the ncurses library.

Install the required packages:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Install the pxdev package into the home directory of your Raspberry Pi.

```
cd ~
```

Clone the pxdev repository as follows:

```
git clone https://git.kontron-electronics.de/sw/ked/raspberry-pi/pixtend-v2/pxdev.git pxdev
```

The command ls lists the contents of your home directory. To change to the folder pxdev use

```
cd pxdev
```

The ls -la command lists the contents of the pxdev package:

It contains a simple build script, make the script executable

```
chmod +x build
```

You can now start the build script:

```
./build
```

Now the PiXtend V2 library, pixtendtool2s, pxauto2s tool, pixtendtool2l and pxauto2l tool are created in sequence.

8.4.2. Activating the SPI bus

The communication between the Raspberry Pi and the PiXtend V2 micro-controller uses SPI. Therefore, make sure that the SPI module on the Raspberry Pi is activated. In the following, we will show you how to activate the SPI bus.

Open the raspi-config:

```
sudo raspi-config
```

and activate the SPI bus under “Interfacing Options” → SPI → “Yes” → “Ok”

If raspi-config does not offer this option, restart the Raspberry Pi manually.

```
sudo reboot
```

8.5. Using pixtendtool2(s/l)

We carry out the following instructions and examples on the basis of the PiXtend V2 -S-, and they also apply to the PiXtend V2 -L-, if the corresponding program is used. Instead of "pixtendtool2s" use "pixtendtool2l".

Change to the directory that contains pixtendtool2s:

```
cd ~/pxdev/PiXtend_V2/pixtendtool2s
```

Enter the following command:

```
sudo ./pixtendtool2s -h
```

This command prints all available calls and options/parameters:

```
PiXtend Tool V2 -S- http://www.pixtend.de - Version 0.5.5
```

```
usage: sudo ./pixtendtool2s [OPTION] [VALUE(s)]
```

Available options:

| | |
|-----------------------------------|--|
| -h | Print this help |
| -do VALUE | Set the digital output byte to VALUE[0-255] |
| -do BIT VALUE | Set the digital output BIT[0-3] to VALUE [0/1] |
| -dor | Get the digital output byte |
| -dor BIT | Get the digital output BIT[0-3] |
| -di | Get the digital input byte |
| -di BIT | Get the digital input BIT[0-7] |
| -ai CHANNEL | Get the analog input CHANNEL[0-1] raw value |
| -ai CHANNEL REF | Get the analog input CHANNEL[0-1] value based on REF[5V/10V] |
| -ao CHANNEL VALUE | Set the analog output CHANNEL[0-1] to VALUE[0-1023] |
| -rel VALUE | Set the relay output byte to VALUE[0-255] |
| -rel BIT VALUE | Set the relay output BIT[0-3] to VALUE[0/1] |
| -relr | Get the relay output byte |
| -relr BIT | Get the relay output BIT[0-3] |
| -gw VALUE | Set GPIO output byte to VALUE[0-255] |
| -gw BIT VALUE | Set GPIO output BIT[0-3] to VALUE[0/1] |
| -gr | Get GPIO input byte |
| -gr BIT | Get GPIO input BIT[0-3] |
| -gc VALUE | Set GPIO control to VALUE[0-255] |
| -tr CHANNEL TYPE | Get temperature from CHANNEL[0-3] of TYPE[DHT11/DHT22] |
| -hr CHANNEL TYPE | Get humidity from CHANNEL[0-3] of TYPE[DHT11/DHT22] |
| -srv0 CHANNEL VALUE | Set servo 0 CHANNEL[0-1] to VALUE[0-65535] |
| -srv1 CHANNEL VALUE | Set servo 1 CHANNEL[0-1] to VALUE[0-255] |
| -pwm0 CHANNEL VALUE | Set PWM 0 CHANNEL[0-1] to VALUE[0-65535] |
| -pwm1 CHANNEL VALUE | Set PWM 1 CHANNEL[0-1] to VALUE[0-255] |
| -pwm0c VALUE0 VALUE1 | Set PWM 0 control VALUE0[0-255] and VALUE1[0-65535] |
| -pwm1c VALUE0 VALUE1 | Set PWM 1 control VALUE0[0-255] and VALUE1[0-255] |
| -swc SERIALDEVICE BAUDRATE CHAR | Write a CHAR on SERIALDEVICE |
| -sws SERIALDEVICE BAUDRATE STRING | Write a STRING on SERIALDEVICE (max 255) |
| -sr SERIALDEVICE BAUDRATE | Read data from SERIALDEVICE until Ctrl^C |
| -ucc0 VALUE | Set the microcontroller control 0 to VALUE[0-255] |
| -ucc1 VALUE | Set the microcontroller control 1 to VALUE[0-255] |
| -ucs | Get microcontroller status register |
| -ucr | Reset the microcontroller |
| -ucv | Get microcontroller version |

```
-ucw                                Get microcontroller warnings
```

Note for PWM 0/1 and Servo 0/1: 0 = channel A and 1 = channel B

The output may vary if there is a new version of pixtendtool2s available in the meantime. The output shown here refers to version 0.5.5.

8.5.1. Example:

The command

```
sudo ./pixtendtool2s -di
```

gives information about the state of the digital input byte at the time of the query:

```
pi@raspberrypi:~/pxdev/PiXtend_V2/pixtendtool2s $ sudo ./pixtendtool2s -di
Digital input byte is [0]
```

Figure 92: Linux Tools - pixtendtool2s - Switch '-di'

```
sudo ./pixtendtool2s -do 15
```

Set the digital outputs to dec. 15, this represents binary 00001111 and sets the outputs 0, 1, 2 and 3.

To query analog inputs, use:

```
sudo ./pixtendtool2s -ai 0
```

```
pi@raspberrypi:~/pxdev/PiXtend_V2/pixtendtool2s $ sudo ./pixtendtool2s -ai 0
Analog input 0 value is [516]
```

Figure 93: Linux Tools - pixtendtool2s - Switch '-ai'

Analog input 0 provides us the value 516. With a reference voltage of 10V and a resolution of 10 bit ($2^{10} = 1024$), the value corresponds to $10V / 1024 * 516 = 5.04$ volts.

Serial transmissions via RS232 can be queried very easily with the pixtendtool2s.

A character string is sent with the following command via the UART interface of the Raspberry Pi:

```
sudo ./pixtendtool2s -sws /dev/ttyS0 115200 test123
```

```
pi@raspberrypi:~/pxdev/PiXtend_V2/pixtendtool2s $ sudo ./pixtendtool2s -sws /dev/ttyS0 115200 test123
Opened serial device [fd=6]
put string test123
Closed serial device
```

Figure 94: Linux Tools - pixtendtool2s - Switch '-sws'

With the Raspberry Pi 3 B (3 B+), the UART interface is called “/dev/ttyS0”. Here, the string “test123” is transmitted with a baud rate of 115,200.

PiXtend V2 -S- has a UART interface which connects to a RS232 converter.¹

¹The PiXtend V2 -L- has an additional RS485 converter that can be activated by GPIO 18. The RS232 interface will then no longer be available on the PiXtend V2 -L-.

The serial interfaces can also receive/read:

```
sudo ./pixtendtool2s -sr /dev/ttyS0 115200
```

```
pi@raspberrypi:~/pxdev/PiXtend_V2/pixtendtool2s $ sudo ./pixtendtool2s -sr /dev/ttyS0 115200
Opened serial device [fd=6]
Recv: Hallo PiXtend V2 -S-!
```

Figure 95: Linux Tools - pixtendtool2s - Switch '-sr'

Additional notes on the UART interface of the Raspberry Pi:

By default, the serial interface of the Raspberry Pi can be used as a remote console. If the interface is to be used exclusively for your own data transfers, the raspi-config program must be used:

```
sudo raspi-config
```

Select:

Interfacing Options --> Serial --> <No> --> <Yes> --> <Ok>

A reboot is then necessary:

```
sudo reboot
```

8.5.2. Further steps

Experiment with the options and parameters.

Once you are familiar with the use and parameters of pixtendtool2(s/l), you can write your own shell scripts using the pixtendtool2(s/l) queries.

You can redirect all output of pixtendtool2(s/l), e.g., directly into a text file:

PiXtend V2 -S-:

```
sudo ./pixtendtool2s -h > Help.txt
```

PiXtend V2 -L-:

```
sudo ./pixtendtool2l -h > Help.txt
```

or append the data to an existing file:

PiXtend V2 -S-:

```
sudo ./pixtendtool2s -ai 0 >> analog0.log
```

PiXtend V2 -L-:

```
sudo ./pixtendtool2l -ai 0 >> analog0.log
```

Set up a cronjob² that executes a script cyclically and stores the states of all analog inputs in a file.

- Another option is a script that enters the analog values into a MySQL database.
- Or a script that turns on a relay at a certain time every day, and off at another time.

Perhaps you are wondering how to operate the control registers or how to process certain values. For this purpose, there is further information in the appendix, where

you will find bit-specific information and all possibilities that the PiXtend V2 -S- and PiXtend V2 -L- offer.

²<https://en.wikipedia.org/wiki/Cron>

8.6. Using pxauto2(s/l)

The following instructions and examples are based on the PiXtend V2 -S-, but they also apply to the PiXtend V2 -L- if the corresponding program is used. Instead of “pxauto2s” use “pxauto2l”.

Change to the directory that contains pxauto2s:

```
cd ~/pxdev/PiXtend_V2/pxauto2s
```

Enter the command

```
sudo ./pxauto2s
```

to start pxauto2s.

NOTICE

Here the use of the “sudo” command is very important. If pxauto2(s/l) is started without “sudo”, unexpected behavior may occur because the hardware may not be accessible. Furthermore, the console output could be disturbed.

pxauto2s currently has the following menu options:

- DIIn – Digital inputs
- AIIn – Analog inputs/temperature/humidity - DHT11/22 setting
- GPIO – General purpose input/output
- DOut – Digital outputs
- AOut – Analog outputs
- PWM – PWM outputs
- Ctrl – Control, debounce and jumper bytes
- Stat – Status bytes
- RetIn – Retain input (refer to chapter 6.4 Retain memory)
- RetOut – Retain output (refer to chapter 6.4 Retain memory)

[illegible]

Figure 96: Linux-Tools - pxauto2s - Main menu

8.6.1. Navigation

After starting pxauto2(s/l), you are initially in the menu (MENU-MODE).

- Use the UP and DOWN arrow keys to select another menu item.
- Press RETURN to go to the currently selected window (EDIT-MODE).
- Color-highlighted fields (yellow) can be edited by typing a value.
- To delete values, use the keyboard keys DEL/BACKSPACE.
- Boolean values (e.g., digital outputs) and selections can be selected with the LEFT/RIGHT arrow keys.
- To accept changes, leave the field with the UP/DOWN arrow keys.
- To return to the menu, press RETURN again.
- This operation automatically applies the value in the current field.

Press the “q” key on the main menu to exit the program.

8.6.2. Editing fields

The highlighted fields (yellow) can be changed. Use the arrow keys to navigate to the desired field and use LEFT/RIGHT to select between the available values or type a new value directly.

(Note: The numeric keypad is usually not supported for access via SSH terminal.)

www.kontron-electronics.de

C-Library “pxdev”

With the PiXtend V2 C-Library ("pxdev"), you will be able to program your own Linux programs for your PiXtend V2 board or integrate its functionalities into your own programs.

You will have access to all inputs and outputs available on PiXtend V2.

This chapter explains the steps required to create your own programs. In addition, you will get an overview of the possibilities offered by the C-Library. At the end of this chapter, you will find two documented sample programs that you can test or use as a basis for your own developments.

Perhaps you have already tried the programs `pxauto2(s/l)` and `pixtendtool2(s/l)`. These programs were created by Kontron Electronics GmbH using the C-Library.

[illegible]

Figure 98: Linux Tools - pxauto2s

```

pi@xena: Tool V2 -> $@ http://www.piktech.de - Version 0.5.5
usage: sudo -i ./piktechdcols [OPTION] [VALUE(s)]

Available options:
-d
    -do VALUE          Set the digital output byte to VALUE[0-255]
    -do BIT VALUE      Set the digital output BIT[0-3] to VALUE [0/1]
-dor
    -dor               Get the digital output byte
    -dor BIT           Get the digital output BIT[0-3]
-di
    -di               Get the digital input byte
    -di BIT           Get the digital input BIT[0-7]
-a1
    -a1 CHANNEL        Get the analog input CHANNEL[0-1] raw value
    -a1 CHANNEL REF    Get the analog input CHANNEL[0-1] value based on REF[50V/10V]
    -a1 CHANNEL VALUE  Get the analog output CHANNEL[0-1] to VALUE[0-1023]
-rel
    -rel VALUE         Set the relay output byte to VALUE[0-255]
    -rel BIT VALUE     Set the relay output BIT[0-3] to VALUE[0/1]
-relr
    -relr              Get the relay output byte
    -relr BIT          Get the relay output BIT[0-3]
-rp
    -rp VALUE          Set GPIO output byte to VALUE[0-255]
    -rp BIT VALUE      Set GPIO output BIT[0-3] to VALUE[0/1]
-gr
    -gr               Get GPIO input byte
    -gr BIT            Get GPIO input BIT[0-3]
-rc
    -rc VALUE          Set GPIO control to VALUE[0-255]
    -rc CHANNEL        Get temperature from CHANNEL[0-3] of TYPE[DHT11/DHT22]
    -rc CHANNEL TYPE   Get humidity from CHANNEL[0-3] of TYPE[DHT11/DHT22]
-rsv0
    -rsv0 CHANNEL VALUE Set servo 0 CHANNEL[0-1] to VALUE[0-65535]
-rsv1
    -rsv1 CHANNEL VALUE Set servo 1 CHANNEL[0-1] to VALUE[0-65535]
-pwm0
    -pwm0 CHANNEL VALUE Set PWM 0 CHANNEL[0-1] to VALUE[0-65535]
    -pwm1 CHANNEL VALUE Set PWM 1 CHANNEL[0-1] to VALUE[0-255]
    -pwm0n0 VALUE PWM1 Set PWM 0 control VALUE[0-255] and PWM1[0-65535]
    -pwm1n1 VALUE PWM1 Set PWM 1 control VALUE[0-255] and PWM1[0-65535]
-swr
    -swr SERIALDEVICE Write a USB on SERIALDEVICE
    -swr SERIALDEVICE BAUDRATE Write a STRING on SERIALDEVICE (max 255)
    -swr SERIALDEVICE BAUDRATE Read data from SERIALDEVICE until Ctrl+C
    -swcr0 VALUE        Set the microcontroller control 0 to VALUE[0-255]
    -swcr1 VALUE        Set the microcontroller control 1 to VALUE[0-255]
    -swcr                Get microcontroller status register
    -swcr                Reset the microcontroller
    -swcr                Get microcontroller version
    -swcr                Get microcontroller warnings

```

Figure 99: Linux Tools - pixtendtool2s

9.1. Requirements

This chapter assumes that you have already installed the C-Library pxdev. If you have not already installed it, see Chapter 8 pxdev – Linux Tools and Library for more information.

The easiest way is to use our SD card image “PiXtend V2 Basic Image”, which you can download free of charge from our website. In this image, pxdev and the required tools are already installed.

You should also familiarize yourself with chapter 12 Appendix. Select the chapters according to your PiXtend V2, basic knowledge of C programming is assumed. The handling of different number systems (decimal, binary, hexadecimal) and the conversion between these must also be known.

You can always find the latest source code of the PiXtend V2 library at <https://git.kontron-electronics.de/sw/ked/raspberry-pi/pixtend-v2/pxdev>. If you are programming on Linux, it can be useful to display the source code on a PC or to print them out. This avoids a constant change between your program code and the source code of pxdev.

9.2. Preparation

The data exchange between the PiXtend V2 and Raspberry Pi is done cyclic via the SPI bus. This makes it possible to exchange all input and output values at once. More information on the SPI and cycle times can be found in chapter 6.2 SPI Communication, Data Transmission and Cycle Time.

The Auto mode offers advantages such as an optional watchdog timer and a check for transmission errors (CRC checksum). Our example programs pxauto2(s/l) use the Auto mode.

We will look at the procedure of the Auto mode and create folders and files. In chapter 9.6 Compiling and running the program, we make an executable program from our code.

Note:

All instructions and examples are shown here using a PiXtend V2 -S-, but they work just as well with a PiXtend V2 -L-. At the end of a statement or command simply replace the “S” with an “L” to be able to use a PiXtend V2 -L-.

We always work in the “pi” user's home directory. This avoids problems with read and write permissions, which can occur in other folders.

After logging on via SSH or after booting and using keyboard and mouse, connected directly to the Raspberry Pi, you always end up automatically in this directory. If you have changed the directory, return to it with the following command:

```
cd /home/pi
```

9.2.1. Creating a new folder and C file

```
mkdir pixCEExample  
cd pixCEExample
```

Now create a new C file pixCEExample and edit it with the editor “nano”:

```
touch pixCEExample.c  
nano pixCEExample.c
```

The nano editor will start and you will see that the pixCEExample.c file is empty.

9.2.2. Adding libraries

Now we can integrate the required PiXtend/PiXtend V2 C-Library (pxdev) into our C file:

```
#include <pixtend.h>
```

You should also include the following standard C-libraries:

```
#include <stdlib.h>
#include <stdio.h>
```

Now the actual programming can be started.

9.3. Programming

In this section the various C functions, which pxdev offers, are presented and explained in more detail. The naming convention of the transfer parameters used in this documentation differ in part from the header file of the PiXtend V2 library. This is for the sake of clarity.

The SPI Auto Mode function allows cyclic and fast processing of the various PiXtend V2 C functions. The cyclic redundancy check (cyclic redundancy check CRC) checks the transmission of the SPI for errors. Furthermore, the use of the function offers the advantage that the watchdog of the PiXtend V2 micro-controller can be used. In case of a software error, this prevents an undefined state of the system. The software regularly notifies the watchdog that it is running properly. If the current program does not report after a certain time, the watchdog triggers a reset of the micro-controller and the program stops (see chapter 6.1 Definition “Safe State”).

In order to set the cycle time and ensure the permanent transfer between the Raspberry Pi and PiXtend V2, an infinite loop must be implemented in the Auto mode. The loop is executed again depending on a condition or after a certain time. Different waiting times values can be used to set the cycle time. An overview of possible values can be found in chapter 6.2 SPI Communication, Data Transmission and Cycle Time.

Functions used:

```
int Spi_AutoModeV2S (struct pixtOutV2S *OutputData, struct pixtInV2S *InputData)
```

```
int Spi_AutoModeDAC (struct pixtOutDAC *OutputDataDAC)
```

Requirements:

The function Spi_SetupV2 (0) must be executed for communication with the micro-controller.

The function Spi_SetupV2 (1) must be executed for communication with the DAC (for analog outputs).

The program of the micro-controller starts automatically.

If the watchdog timer is to be used as well, the function must be executed as follows:

```
OutputData.byUcCtrl0 = 0x07; //Watchdog set to 1 sec.
```

The function Spi_AutoModeV2S (struct OutputData, struct InputData) initiates a complete data exchange between the PiXtend V2 - S- and Raspberry Pi. In order to use the function, a new structure of the type pixtInV2S and pixtOutV2S has to be created.

```
struct pixtInV2S InputData;
struct pixtOutV2S OutputData;
```

NOTICE

For the output data, here in this example "OutputData", the variable OutputData.byModelOut must always be set to the value of the applicable PiXtend V2 model. The numbers are derived from the ASCII table for the letters "S" and "L".

- "S" has the ASCII decimal number "83", this is for the PiXtend V2 -S-:
OutputData.byModelOut = 83
- "L" has the ASCII decimal number "76", this is for the PiXtend V2 -L-:
OutputData.byModelOut = 76

The following variables are available in the pixtOutV2S structure for the PiXtend V2 -S-. For more information on the individual bits and bytes, please refer to the corresponding chapters in the appendix.

```

OutputData.byModelOut;           //One byte model as handshake
OutputData.byUCMode              //Reserved, not used
OutputData.byUCCtrl0;            //One byte for uC Control 0
OutputData.byUCCtrl1;            //One byte for uC Control 1
OutputData.byDigitalInDebounce01; //One byte for digital inputs 0 and 1 debounce
OutputData.byDigitalInDebounce23; //One byte for digital inputs 2 and 3 debounce
OutputData.byDigitalInDebounce45; //One byte for digital inputs 4 and 5 debounce
OutputData.byDigitalInDebounce67; //One byte for digital inputs 6 and 7 debounce
OutputData.byDigitalOut;         ///One byte for setting digital outputs
OutputData.byRelayOut;           //One byte for setting the relay
OutputData.byGPIOCtrl;           //One byte for GPIO control
OutputData.byGPIOOut;            //One byte for setting the GPIO outputs
OutputData.byGPIODebounce01;     //One byte for GPIO inputs 0 and 1 debounce
OutputData.byGPIODebounce23;     //One byte for GPIO inputs 2 and 3 debounce
OutputData.byPWM0Ctrl0;           //One byte for PWM 0 Control 0
OutputData.wPWM0Ctrl1;            //Two bytes for PWM 0 control 1
OutputData.wPWM0A;                //Two bytes for PWM 0 channel A value
OutputData.wPWM0B;                //Two bytes for PMW 0 channel B value
OutputData.byPWM1Ctrl0;           //One byte for PWM 1 Control 0
OutputData.byPWM1Ctrl1;           //One byte for PWM 1 Control 1
OutputData.byPWM1A;               //One byte for PWM 1 channel A value
OutputData.byPWM1B;               //One byte for PWM 1 channel B value
OutputData.byJumper10V;           //One byte for jumper setting Analog In 0,1
OutputData.byGPIO0Dht11;          //One byte DHT11 / 22 selection for GPIO 0
OutputData.byGPIO1Dht11;          //One byte DHT11 / 22 selection for GPIO 1
OutputData.byGPIO2Dht11;          //One byte DHT11 / 22 selection for GPIO 2
OutputData.byGPIO3Dht11;          //One byte DHT11 / 22 selection for GPIO 3
OutputData.abRetainDataOut[32];    //One array with 32 bytes Retain Data Output
    
```


The following variables are available in the pixtOutV2L structure for the PiXtend V2 -L-. For more information on the individual bits and bytes, please refer to the corresponding chapters in the appendix.

```

OutputData.byModelOut;           //One byte model as handshake
OutputData.byUCMode              //Reserved, not used
OutputData.byUCCtrl0;            //One byte for uC Control 0
OutputData.byUCCtrl1;            //One byte for uC Control 1
OutputData.byDigitalInDebounce01; //One byte for digital inputs 0 and 1 debounce
OutputData.byDigitalInDebounce23; //One byte for digital inputs 2 and 3 debounce
OutputData.byDigitalInDebounce45; //One byte for digital inputs 4 and 5 debounce
OutputData.byDigitalInDebounce67; //One byte for digital inputs 6 and 7 debounce
OutputData.byDigitalInDebounce89; //One byte for digital inputs 8 and 9 debounce
OutputData.byDigitalInDebounce1011; //One byte for digital inputs 10 and 11 debounce
OutputData.byDigitalInDebounce1213; //One byte for digital inputs 12 and 13 debounce
OutputData.byDigitalInDebounce1415; //One byte for digital inputs 14 and 14 debounce
OutputData.byDigitalOut0;         //One byte for setting digital outputs 0 - 7
OutputData.byDigitalOut1;         //One byte for setting digital outputs 8 - 11
OutputData.byRelayOut;            //One byte for setting the relay
OutputData.byGPIOCtrl;           //One byte for GPIO control
OutputData.byGPIOOut;            //One byte for setting the GPIO outputs
OutputData.byGPIODebounce01;     //One byte for GPIO inputs 0 and 1 debounce
OutputData.byGPIODebounce23;     //One byte for GPIO inputs 2 and 3 debounce
OutputData.byPWM0Ctrl0;          //One byte for PWM 0 Control 0
OutputData.wPWM0Ctrl1;           //Two bytes for PWM 0 control 1
OutputData.wPWM0A;               //Two bytes for PWM 0 channel A value
OutputData.wPWM0B;               //Two bytes for PMW 0 channel B value
OutputData.byPWM1Ctrl0;          //One byte for PWM 1 Control 0
OutputData.wPWM1Ctrl1;           //Two bytes for PWM 1 Control 1
OutputData.wPWM1A;               //Two bytes for PWM 1 channel A value
OutputData.wPWM1B;               //Two bytes for PWM 1 channel B value
OutputData.byPWM2Ctrl0;          //One byte for PWM 2 Control 0
OutputData.wPWM2Ctrl1;           //Two bytes for PWM 2 Control 1
OutputData.wPWM2A;               //Two bytes for PWM 2 channel A value
OutputData.wPWM2B;               //Two bytes for PWM 2 channel B value
OutputData.byJumper10V;          //One byte for jumper setting Analog In 0,1
OutputData.byGPIO0Dht11;         //One byte DHT11 / 22 selection for GPIO 0
OutputData.byGPIO1Dht11;         //One byte DHT11 / 22 selection for GPIO 1
OutputData.byGPIO2Dht11;         //One byte DHT11 / 22 selection for GPIO 2
OutputData.byGPIO3Dht11;         //One byte DHT11 / 22 selection for GPIO 3
OutputData.abRetainDataOut[64];   //One array with 64 bytes Retain Data Output
    
```

With `OutputData.byJumper10V`, the jumper position of the analog inputs AI0 and AI1 can be set to 5V or 10V. For voltages lower than 5V it is useful to use the 5V jumper setting, because measurements can be made with a higher resolution³.

If you want to set the AI0 to 10V, the variable must be filled as follows:

```
OutputData.byJumper10V = 0x01;
```

If AI1 is to be set to the 10V jumper position, use

```
OutputData.byJumper10V = 0x02;
```

If both analog inputs are to be set to 10V:

```
OutputData.byJumper10V = 0x03;
```

For the jumper position 5V, the bits do not have to be set or written with the value "0".

For more information, please refer to the appendix for your particular PiXtend V2 device.

³Please remember to set the physical jumper on the PiXtend board before changing from 10V to 5V; otherwise, you will get wrong values later in the program.

The input data of the PiXtend V2 -S- can be extracted from the following variables of the pixtInV2S structure:

```

InputData.byFirmware;           //One byte for the micro-controller firmware version
InputData.byHardware;           //One byte for the hardware status
InputData.byModelIn;            //One byte model for the handshake
InputData.byUCState;            //One byte for the micro-controller status
InputData.byUCWarnings;         //One byte for the warnings from the micro-controller
InputData.byDigitalIn;          //One byte for the digital inputs
InputData.byGPIOIn;             //One byte for the GPIO inputs
InputData.wAnalogIn0;           //Two bytes for analog input 0
InputData.wAnalogIn1;           //Two bytes for analog input 1
InputData.wTemp0                //Two bytes for temperature value DHT11/22 on GPIO0
InputData.wTemp1                //Two bytes for temperature value DHT11/22 on GPIO1
InputData.wTemp2                //Two bytes for temperature value DHT11/22 on GPIO2
InputData.wTemp3                //Two bytes for temperature value DHT11/22 on GPIO3
InputData.wHumid0               //Two bytes for humidity value DHT11/22 on GPIO0
InputData.wHumid1               //Two bytes for humidity value DHT11/22 on GPIO1
InputData.wHumid2               //Two bytes for humidity value DHT11/22 on GPIO2
InputData.wHumid3               //Two bytes for humidity value DHT11/22 on GPIO3
InputData.rAnalogIn0;           //float variable for voltage input 0
InputData.rAnalogIn1;           //float variable for voltage input 1
InputData.rTemp0;               //float variable for temperature on GPIO0
InputData.rTemp1;               //float variable for temperature on GPIO1
InputData.rTemp2;               //float variable for temperature on GPIO2
InputData.rTemp3;               //float variable for temperature on GPIO3
InputData.rHumid0;              //float variable for humidity on GPIO0
InputData.rHumid1;              //float variable for humidity on GPIO1
InputData.rHumid2;              //float variable for humidity on GPIO2
InputData.rHumid3;              //float variable for humidity on GPIO3
InputData.abRetainDataIn[32];   //Array with 32 bytes Retain data input

```

The input data of the PiXtend V2 -L- can be extracted from the following variables of the pixtInV2L structure:

```

InputData.byFirmware;           //One byte for the micro-controller firmware version
InputData.byHardware;           //One byte for the hardware status
InputData.byModelIn;            //One byte model for the handshake
InputData.byUCState;            //One byte for the micro-controller status
InputData.byUCWarnings;         //One byte for the warnings from the micro-controller
InputData.byDigitalIn0;         //One byte for setting digital outputs 0 - 7
InputData.byDigitalIn1;         //One byte for setting digital outputs 8 - 15
InputData.byGPIOIn;             //One byte for the GPIO inputs
InputData.wAnalogIn0;           //Two bytes for analog input 0
InputData.wAnalogIn1;           //Two bytes for analog input 1
InputData.wAnalogIn2;           //Two bytes for analog input 2
InputData.wAnalogIn3;           //Two bytes for analog input 3
InputData.wAnalogIn4;           //Two bytes for analog input 4
InputData.wAnalogIn5;           //Two bytes for analog input 5
InputData.wTemp0                //Two bytes for temperature value DHT11/22 on GPIO0
InputData.wTemp1                //Two bytes for temperature value DHT11/22 on GPIO1
InputData.wTemp2                //Two bytes for temperature value DHT11/22 on GPIO2
InputData.wTemp3                //Two bytes for temperature value DHT11/22 on GPIO3
InputData.wHumid0               //Two bytes for humidity value DHT11/22 on GPIO0
InputData.wHumid1               //Two bytes for humidity value DHT11/22 on GPIO1
InputData.wHumid2               //Two bytes for humidity value DHT11/22 on GPIO2
InputData.wHumid3               //Two bytes for humidity value DHT11/22 on GPIO3
InputData.rAnalogIn0;           //float variable for voltage input 0
InputData.rAnalogIn1;           //float variable for voltage input 1
InputData.rAnalogIn2;           //float variable for voltage input 2
InputData.rAnalogIn3;           //float variable for voltage input 3
InputData.rAnalogIn4;           //float variable for current input 4
InputData.rAnalogIn5;           //float variable for current input 5
InputData.rTemp0;               //float variable for temperature on GPIO0
InputData.rTemp1;               //float variable for temperature on GPIO1
InputData.rTemp2;               //float variable for temperature on GPIO2
InputData.rTemp3;               //float variable for temperature on GPIO3
InputData.rHumid0;              //float variable for humidity on GPIO0
InputData.rHumid1;              //float variable for humidity on GPIO1
InputData.rHumid2;              //float variable for humidity on GPIO2
InputData.rHumid3;              //float variable for humidity on GPIO3
InputData.abRetainDataIn[64];   //Array with 64 bytes Retain data input
    
```

To transfer the input and output structures to the function, the following line is used:

```
Spi_AutoModeV2S(&OutputData, &InputData);
```

The OutputData structure contains the values for the control bytes and process data (output values from the point of view of the Raspberry Pi to the PiXtend V2 -S-).

The values of the inputs are written (by the PiXtend V2 -S- micro-controller) to the InputData structure during program execution).

A small example:

With the following code, the temperature can be read via GPIO0 from a DHT sensor and output to the console:

```
dht0Temp=InputData.rTemp0;
printf("Temperatur C: %.2f\n", dht0Temp);
```

The function Spi_AutoModeDAC(struct OutputDataDAC) is used exclusively for communication with the digital-to-analog converter (DAC). The function requires a structure of type pixtOutDAC as the transfer parameter. This structure can be created as follows:

```
struct pixtOutDAC OutputDataDAC
```

The structure must contain the values for the two analog outputs.

```
OutputDataDAC.wAOut0 = 1023; //1023 corresponds to the maximum value, 0 minimum value
OutputDataDAC.wAOut1 = 1023;
```

Here the analog outputs are set to 100%.

The following query passes the structure OutputDataDAC to the function.

```
Spi_AutoModeDAC(&OutputDataDAC);
```

9.4. Example program for PiXtend V2 -S-

```
//Include the libraries
#include <pixtend.h>
#include <stdio.h>

//Function prototype
int GetPixData();

//Define some variables
float dht1Temp=0.0;
float dht1Hum=0.0;

//PiXtend V2 -S- input data
struct pixtInV2S InputData;

//PiXtend V2 -S- output data
struct pixtOutV2S OutputData;

//PiXtend V2 -S- DAC Output Data
struct pixtOutDAC OutputDataDAC;

//Read Temperature and Humidity from GPIO1 (DHT22)
int GetPixData()
{
    dht1Temp=InputData.rTemp1;
    dht1Hum=InputData.rHumid1;

    //Only print out valid data
    if(dht1Hum>0.0)
    {
        //Write out the data on the linux console
        printf("Temperature [°C]: %.2f\n", dht1Temp);
        printf("Humidity [%%]: %.2f\n", dht1Hum);
    }

    return 0;
}

int main(void)
{
    //Configure SPI
    Spi_SetupV2(0);
    Spi_SetupV2(1);

    //Write Data in Structure OutputData
    OutputData.byModelOut = 83; // Set model as handshake, PiXtend V2 -S- = 83
    OutputData.byGPIOCtrl = 32; //GPIO1 is used with DHT22 sensor
    OutputDataDAC.wAOut0 = 1023; //pre-load analog outputs
    OutputDataDAC.wAOut1 = 1023;

    //Auto Mode in infinity loop
    while(1)
    {
        //Do the data transfer!
        Spi_AutoModeV2S(&OutputData, &InputData);
        Spi_AutoModeDAC(&OutputDataDAC);

        //Set all Relays if at least one digital input is High
        if(InputData.byDigitalIn>0)
        {
            printf("Input detected - setting Relays!\n");
            OutputData.byRelayOut = 15;
        }
        else
        {
            OutputData.byRelayOut = 0;
        }

        //Toggle Analog Outputs
        if(OutputDataDAC.wAOut0 == 1023 || OutputDataDAC.wAOut1 == 1023)
        {
            OutputDataDAC.wAOut0 = 512;
            OutputDataDAC.wAOut1 = 512;
        }
    }
}
```

```
    }  
    else  
    {  
        OutputDataDAC.wAOut0 = 1023;  
        OutputDataDAC.wAOut1 = 1023;  
    }  
  
    //Read data and print it out  
    GetPixData();  
  
    //Take a 100 ms break between the cycles  
    delay(100);  
}  
return 0;  
}
```

9.5. Example program for PiXtend V2 -L-

```

//Include the librarys
#include <pixtend.h>
#include <stdio.h>

//Function prototype
int GetPixData();

//Define some variables
float dht1Temp=0.0;
float dht1Hum=0.0;

//PiXtend V2 -L- input data
struct pixtInV2L InputData;

//PiXtend V2 -L- output data
struct pixtOutV2L OutputData;

//PiXtend V2 -L- DAC Output Data
struct pixtOutDAC OutputDataDAC;

//Read Temperature and Humidity from GPIO1 (DHT22)
int GetPixData()
{
    dht1Temp=InputData.rTemp1;
    dht1Hum=InputData.rHumid1;

    //Only print out valid data
    if(dht1Hum>0.0)
    {
        //Write out the data on the linux console
        printf("Temperature [°C]: %.2f\n", dht1Temp);
        printf("Humidity [%%]: %.2f\n", dht1Hum);
    }

    return 0;
}

int main(void)
{
    //Configure SPI
    Spi_SetupV2(0);
    Spi_SetupV2(1);

    //Write Data in Structure OutputData
    OutputData.byModelOut = 76; // Set model as handshake, PiXtend V2 -L- = 76
    OutputData.byGPIOCtrl = 32; //GPIO1 is used with DHT22 sensor
    OutputDataDAC.wAOut0 = 1023; //pre-load analog outputs
    OutputDataDAC.wAOut1 = 1023;

    //Auto Mode in infinity loop
    while(1)
    {
        //Do the data transfer!
        Spi_AutoModeV2L(&OutputData, &InputData);
        Spi_AutoModeDAC(&OutputDataDAC);

        //Set all Relays if at least one digital input is High
        if(InputData.byDigitalIn0>0 || InputData.byDigitalIn1>0)
        {
            printf("Input detected - setting Relays!\n");
            OutputData.byRelayOut = 15;
        }
        else
        {
            OutputData.byRelayOut = 0;
        }

        //Toggle Analog Outputs
        if(OutputDataDAC.wAOut0 == 1023 || OutputDataDAC.wAOut1 == 1023)
        {
            OutputDataDAC.wAOut0 = 512;
            OutputDataDAC.wAOut1 = 512;
        }
        else
        {
            OutputDataDAC.wAOut0 = 1023;

```



```
        OutputDataDAC.wAOut1 = 1023;
    }

    //Read data and print it out
    GetPixData();

    //Take a 100 ms break between the cycles
    delay(100);
}
return 0;
}
```

9.6. Compile and run the program

To execute the program, it must first be compiled, enter the following line:

```
sudo gcc -Wall -o "pixCEExample" "pixCEExample.c" -lpixtend -lwiringPi
```

Now pixCEExample.c is compiled and saved in pixCEExample.

With the command

```
sudo ./pixCEExample
```

the program is executed.

If you do not want to use sudo, you can give the compiled file pixCEExample the required rights. Use the following line for this:

```
sudo chmod -R 0777 pixCEExample
```

The sudo⁴ command can now be omitted when executing the C program.

To exit a program, use this key combination Ctrl+C.

⁴sudo is only necessary here because the SPI interface is accessed via the wiringPi library (<https://github.com/WiringPi/WiringPi.git>). Depending on the version, this access still requires superuser rights.

9.7. Frequently Asked Questions (FAQs)

What do I need to take into account when using the PiXtend/PiXtend V2 C library?

The SPI interface must be configured with the function `Spi_SetupV2 (int device)`. The `Spi_SetupV2` function can be executed up to two times (once for device 0, once for device 1). Therefore, the executions should be moved to an initialization routine that is executed only once (immediately after the program is started).

The handshake must be communicated by the PiXtend V2 via the output byte `byModelOut`, this is always the decimal number 83 for the PiXtend V2 -S- and 76 for the PiXtend V2 -L-.

There is sometimes a transfer problem between the PiXtend and Raspberry Pi. What has to be done?

In your program, check whether you execute the function `Spi_SetupV2 (int device)` several times or cyclically. If this is the case, this is the reason for the transmission errors. If there is no transmission at all, check whether the switch `SPI_EN` on the PiXtend V2 is ON. If DHT sensors are used, the cycle time must be at least 30 ms. Generally, the cycle time should not be less or faster than 2.5 ms (PiXtend V2 -S-) or 5 ms (PiXtend V2 -L-).

Where is the cycle time set?

With C programs, using the PiXtend Library, the user must take care of the cycle time. In the following example, the PiXtend V2 -L- AutoMode is executed in an infinite loop every 100 ms, or at the end of a run program is paused for 100 ms before it continues.

```
//Auto Mode in infinity loop
while(1)
{
    //Do the data transfer!
    Spi_AutoModeV2L(&OutputData, &InputData);
    Spi_AutoModeDAC (&OutputDataDAC);

    //Take a 100 ms break between the cycles
    delay(100);
}
return 0;
```

Figure 100: C program - 100ms cycle time

FHEM

FHEM is a PERL-based server program for home automation, which enables the user to integrate different sensors and actuators into a single system and to automate processes. These sensors and actuators can, for example, be switches, lamps, heaters or temperature and humidity sensors.

Therefore, this FHEM module gives you full access to the functions of the PiXtend V2 within the FHEM environment.

The FHEM module developed by Kontron Electronics GmbH for the PiXtend V2 is a stand-alone module. It was developed for the Raspberry Pi and has no software dependencies on other modules or utilities.

10.1. Requirements

For the installation of FHEM on the PiXtend V2 controller, basic knowledge of the Raspberry Pi and its Linux operating system is required. In addition to a PiXtend V2 and a Raspberry Pi, the following is required:

- SD card with the latest Raspbian image. See chapter 6.5 Preparing a SD card for your Raspberry Pi to create such an SD card.
- SSH client software (e.g. `putty.exe` – www.putty.org)
- FTP client software (e.g. WinSCP – www.winscp.net)

For the subsequent application, it is useful to get an overview of the available functions and possibilities of the PiXtend V2.

NOTICE

Also refer to the notes on software compatibility in section 6.10 Compatibility of the Software Components.

10.2. Installation

For the installation of FHEM and the FHEM module for the PiXtend V2, the following steps have to be taken. If FHEM is already installed on the PiXtend V2 board, the first step can be skipped and you can start directly with the integration of the module.

10.2.1. Installation of FHEM

The latest instructions for installing FHEM on the Raspberry Pi can be found here: <https://debian.fhem.de/>.

It is recommended to perform the installation according to “The easy way: use apt-get” (listed in the left menu). For the version 21 September 2017 - Rev 5.8.15110, the installation is as follows.

The repository key for FHEM is updated with the following command:

```
wget -qO - http://debian.fhem.de/archive.key | sudo apt-key add -
```

Now the address of the repository must be entered in the **sources.list** file. Open the file with the following command:

```
sudo nano /etc/apt/sources.list
```

and add the next line at the end. Save the file (Ctrl+X and confirm with Y and then Enter). The entry is automatically deleted again when FHEM is installed.

```
deb http://debian.fhem.de/nightly/ /
```

Then the installed packages must be updated with the new repository:

```
sudo apt-get update
```

Afterwards FHEM can be installed (confirm the request if FHEM should be installed with 'Y'):

```
sudo apt-get install fhem
```

10.2.2. Integration of the module

To integrate the FHEM module for the PiXtend V2 into the FHEM environment, the module has to be copied into the FHEM directory. The module can be found as a ZIP file in the Download section of our website. Since the FHEM directory belongs to the user “fhem”, no files can be stored here without granting permission. As a result, the users of the group temporarily require the necessary write permissions:

```
sudo chmod g+w /opt/fhem/FHEM
```

Afterwards the module “97_PiXtendV2.pm” can be copied to the directory “/opt/fhem/FHEM” on the Raspberry Pi with the help of a FTP client. At the end, the rights of the group should be reset:

```
sudo chmod g-w /opt/fhem/FHEM
```

10.2.3. Configuration of the system

The SP interface is used for the data transfer between the Raspberry Pi and the PiXtend V2 board. Therefore, SPI has to be enabled in the settings of the Raspberry Pi.

```
sudo raspi-config
```

You open the menu and activate the interface under “5 Interfacing Options” with “P4 SPI”. Exit the menu again after that.

By default, the user “fhem” has no rights for the SPI interface and the GPIOs. If the user needs these, the group rights are added:

```
sudo adduser fhem gpio; sudo adduser fhem spi; sudo reboot
```

This line also performs the required restart.

After restarting the Raspberry Pi, you can access the FHEM interface page of the Raspberry Pi in a browser with the following address:

```
http://<RaspberryIP>:8083/fhem
```

Replace <RaspberryIP> with your device’s IP address. You get the current IP address with the following command:

```
ifconfig
```

The installation of FHEM and the integration of the module are done and you can start with your home automation.

10.3. Description of the module

The PiXtend V2 module for FHEM has a variety of functions which can be used. The following pages describe how such a module can be created and how the functions can be used. The functions can be executed either via the FHEM web interface of the module or by writing the appropriate command to the command line of FHEM.

10.3.1. Creating a new PiXtend V2 device

A new PiXtend V2 -S- module is created by entering the command

```
define <name> PiXtendV2
```

A new PiXtend V2 -L- module is created by entering the command

```
define <name> PiXtendV2 L
```

Replace <name> with a name of your choice.

We do not recommend creating multiple PiXtend V2 modules in FHEM for an actual physical device. It does not make sense and can cause problems because the same hardware functions are accessed from several places.

10.3.2. Internals

The internals which belong exclusively to this module are the “RetainSize” and the field “STATE”. RetainSize represents the size of retain data in bytes. The value of STATE can be “defined”, “active” or “error”. The value “defined” is only displayed shortly after the module is created and then immediately changes to “active”. If an error occurs, the value changes to “error” and the field “ErrorMsg” contains more accurate error information of the failure that caused the error. In most cases, this is due to a faulty communication between PiXtend V2 and the Raspberry Pi, e.g. If PiXtend V2 is in safe state.

10.3.3. Set commands

The PiXtend V2 module has a variety of set commands to control digital outputs or relays. Commands to perform the basic configuration for the PiXtend start with an underscore (‘_’) and can be stored in the attribute “PiXtend_Parameter”.

If a command supports multiple channels the pound “#” sign has to be replaced with the channel number. All set commands are case insensitive to guarantee easy use.

`_GPIO#Ctrl` [input, output, DHT11, DHT22]

With this setting the GPIO can be configured as [input], [output] or as [DHT11] or [DHT22] as well, if a DHT sensor is connected to this GPIO. If a DHT is selected and connected, the GPIO cannot simultaneously be used as a normal GPIO.

`_GPiOPullupsEnable` [yes, no]

This setting enables [yes] or disables [no] the possibility to set the internal pull-up resistors via the `GPIOOut` setting for all GPIOs.

`_JumperSettingAI#` [5V, 10V]

This setting affects the calculation of the voltage on the analog inputs and refers to the actual setting of the physical jumper on the PiXtend V2 board [5V,10V]. The default value is [10V] if no jumper is used.

`_StateLEDDisable` [yes, no]

This setting disables [yes] or enables [no] the status LED on the PiXtend V2. If the LED is disabled, it will not light up when there is an error.

`WatchdogEnable` [disable,125ms,1s,8s]

This setting allows to configure the watchdog timer. If the watchdog is configured, the PiXtend V2 switches to the safe state if no valid transmission between the Raspberry Pi and the PiXtend V2 has taken place within the set time. New communication is only possible again after resetting the PiXtend V2.

`AnalogOut#` []

This sets the analog output to the selected voltage. The value can be a voltage between 0V and 10V or a raw value between 0 and 1023. To set the value to a voltage, the value has to include a “.” even if it is an even number.

`DigitalDebounce#` [0-255]

This allows for the denouncing of the digital inputs. The setting always affects two channels, `DigitalDebounce01` affects `DigitalIn0` and `DigitalIn1`. The resulting delay is calculated as (selected value) * (100 ms). The selected value can be any number between 0 and 255. Debouncing is useful if switches or buttons are connected to the inputs.

`DigitalOut#` [on, off, toggle]

This sets the digital output to HIGH [on] or LOW [off] or [toggle] it.

`GPiODEbounce#` [0-255]

This allows for the denouncing of the GPIO inputs. The setting always affects two channels, `GPiODEbounce01` affects `GPiOIn0` and `GPiOIn1`. The resulting delay is calculated as (selected value)*(100 ms). The selected value can be any number between 0 and 255. Debouncing is useful if switches or buttons are connected to the inputs.

`GPIOOut#` [on, off, toggle]

This sets the GPIO to HIGH [on] or LOW [off] or [toggle] it, if it is configured as an output. If it is configured as an input, this command can enable [on] or disable [off] or [toggle] the internal pull-up resistor for that GPIO. However, this option must be activated globally with “`_GPiOPullupsEnable`”.

PWM

PiXtendV2 supports several PWM modes, which can be configured with the commands starting with “PWM”. Chapter 14 explains the exact meaning of the values and how to set the PWM modes.

`RelayOut#` [on, off, toggle]

This sets the relay to HIGH [on] or LOW [off] or toggles [toggle] it.

Reset

This resets the PiXtend V2 controller, for example if it is in safe state and allows to access it again.

RetainCopy [on, off]

If RetainCopy is enabled [on] the RetainDataOut that is written to the PiXtend V2 will be received in RetainDataIn again. This can be useful in some situations to see which data was send to the PiXtend V2. If it is disabled [off] the last data stored in RetainDataIn will be received.

RetainDataOut [0-(RetainSize-1)] [0-255]

PiXtend V2 supports the storage of Retain data. The retain data is organized in bytes and each byte can be written individually with a value between 0 and 255.

As the first parameter, the command needs the byte index which is between 0 and (RetainSize-1). The RetainSize is shown in the "Internals". The value to be stored is expected as the second parameter.

RetainEnable [on, off]

The function of storing Retain data on the PiXtend V2 has to be enabled [on]; otherwise, no data is stored. The memory in which the data is stored supports 10,000 write-cycles. Only activate this function if it is actually required.

SafeState

This setting allows to force the PiXtend V2 to enter the safe state. If retain storage is enabled, the data will be stored. In safe state the PiXtend V2 will not communicate with FHEM and cannot be configured. A reset has to be performed to restart a PiXtend V2.

10.3.4. Get commands

Get commands query the readings or the system state. The format of the return value can be set by an attribute ("PiXtend_GetFormat") to a text or the "pure" value. If the output is active as text, the value is in square brackets.

If a command supports multiple channels, the pound "#" sign has to be replaced with the channel number. All get commands are case insensitive to guarantee easy use.

AnalogIn#

This returns the value of the selected analog input. The result depends on the selected _JumperSettingAI# and the actual physical jumper position on the board.

DigitalIn#

This returns the status on (HIGH) or off (LOW) of the digital input.

GPIOIn#

This returns the status on (HIGH) or off (LOW) of the GPIO, independent of its configuration (input or output).

RetainDataIn [0-(RetainSize-1)]

This returns the value of the selected RetainDataIn byte.

Sensor# [temperature, humidity]

If a DHT sensor is connected to the corresponding GPIO and the _GPIO#Ctrl is set to DHT11 or DHT22, the temperature and humidity are measured and can be read.

SysState

This returns the system status [defined, active, error] of the FHEM module.

UCState

This returns the status of the PiXtend V2. If it is 1, everything is fine. If it is greater than 1 an error occurred or is present, and the PiXtend V2 cannot be configured. More information can be found in sections 14.1.2.4 and 14.2.2.4.

UCWarnings

This returns a value that represents the PiXtend V2 warnings. More information can be found in sections 14.1.2.4 and 14.2.2.4

Version

This returns the FHEM module version and the micro-controller version [Model-Hardware-Firmware].

10.3.5. Readings

Readings are values provided by the controller, e.g. sensor values or the state of inputs. The meaning of the readings is similar to the get commands. Most of these readings trigger an event which allows FHEM and the user to react to this change.

AnalogIn#

This shows the result of the measurement on the analog inputs in volts.

DigitalIn#

This shows the status on (HIGH) or off (LOW) of the digital inputs.

Firmware

This shows the firmware version.

GPIOIn#

This shows the status on (HIGH) or off (LOW) of the GPIO, independent of its configuration (input or output).

Hardware

This shows the hardware version.

Model

This shows the model.

RetainDataIn

This shows the values of the RetainDataIn data. The values of RetainDataIn are combined in one row. The most left value represents Byte0/RetainDataIn0. Each value is separated by a blank and thus can be parsed very easy in Perl:

```
my ($str) = ReadingsVal(pix, "RetainDataIn", "?")
if($str ne "?"){
    my @val = split(/ /, $str);    => $val[0] now contains Byte0,
                                   $val[1] Byte1, etc.
    ...
}
```

Sensor#T/H

This shows the temperature (T) in °C and the humidity (H) in % of the sensor that is connected to the corresponding GPIO.

UCState

This shows the status of the PiXtend V2. If it is 1, everything is fine. If it is greater than 1 an error occurred or is present, and the PiXtend V2 cannot be configured. More information can be found in sections 14.1.2.4 and 14.2.2.4.

UCWarnings

This displayed value represents the PiXtend V2 warnings.

10.3.6. Attribute

The attribute name is case-sensitive.

PiXtend_GetFormat [text, value]

This changes the style in which the values of the get commands are returned. They can be returned as a message [text] or as a raw [value]. The default output is as text.

PiXtend_Parameter

With this attribute, the base configuration (set commands with a leading “_”) can be saved as an attribute. In contrast to set commands, attributes are stored in the Config file of FHEM and are automatically loaded at startup. The value of each command is transferred after a colon and several commands are separated by a space, for example:

```
attr pix PiXtend_Parameter _gpio0ctrl:dht11 _gpio3ctrl:dht22
```

10.4. Example:

The starting point for the following examples always is a PiXtend V2 -S- or PiXtend V2 -L- device with the name “pix” that has already been added. The module is defined by this command:

```
define pix PiXtendV2
```

or

```
define pix PiXtendV2 L
```

10.4.1. Display of sensor values

In FHEM, values can be displayed in a diagram (plot). This allows the graphical processing of sensor values which are connected to the PiXtend V2. In this example, a DHT11 sensor is connected to GPIO0 and a DHT22 sensor is connected to GPIO3. Power is supplied via a 5V cable provided at the terminals.

To activate the sensor functionality, the device “pix” has to be opened in FHEM. After that, select the set command “_GPIO0Ctrl” and on the second field “DHT11” and confirm the setting by pressing the set button. Alternatively, the following command can be written directly to the command line (see figure 101):

```
set pix _gpio0ctrl dht11
```

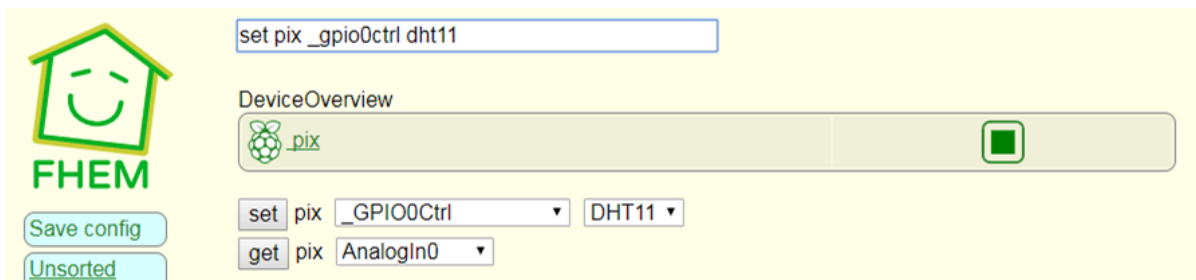


Figure 101: FHEM - Activation of the sensor functionality

Repeat this step for the DHT22 sensor and GPIO3.

Optionally, this configuration (and other settings) can be loaded automatically when FHEM starts. Therefore, the commands have to be set via the attribute “PiXtend_Parameter”.

The commands are stored in the command line as follows:

```
attr pix PiXtend_Parameter _gpio0ctrl:dht11 _gpio3ctrl:dht22
```

The value of a command is transferred after a colon and several commands are separated by a space. With this method, all set commands that begin with an underscore (‘_’) can be stored in the FHEM startup.

After configuration, the measured values for sensor0 and sensor3 are now displayed in the “Readings” area of the “pix” device.

Most of the measured values are displayed according to this tutorial:

https://wiki.fhem.de/wiki/Buderus_Web_Gateway#Mit_FileLog

In the first step, a LogFile is generated, only the required measured values from the readings are saved here:

```
define pixlog FileLog ./log/pix-%Y-%m.log
pix:Sensor0H:.*|pix:Sensor0T:.*|pix:Sensor3H:.*|pix:Sensor3T:.*
```

The second step is to create an SVG plot in this LogFile by clicking on the corresponding button (see figure 102).

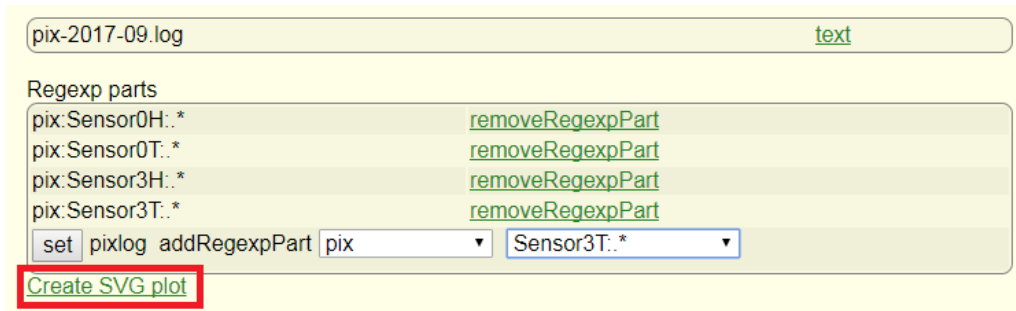


Figure 102: FHEM - Creating a plot via the LogFile

Now the plot can be configured, e.g. like the one in figure 103. After the configuration, press the button “write .gplot file” to create the plot.

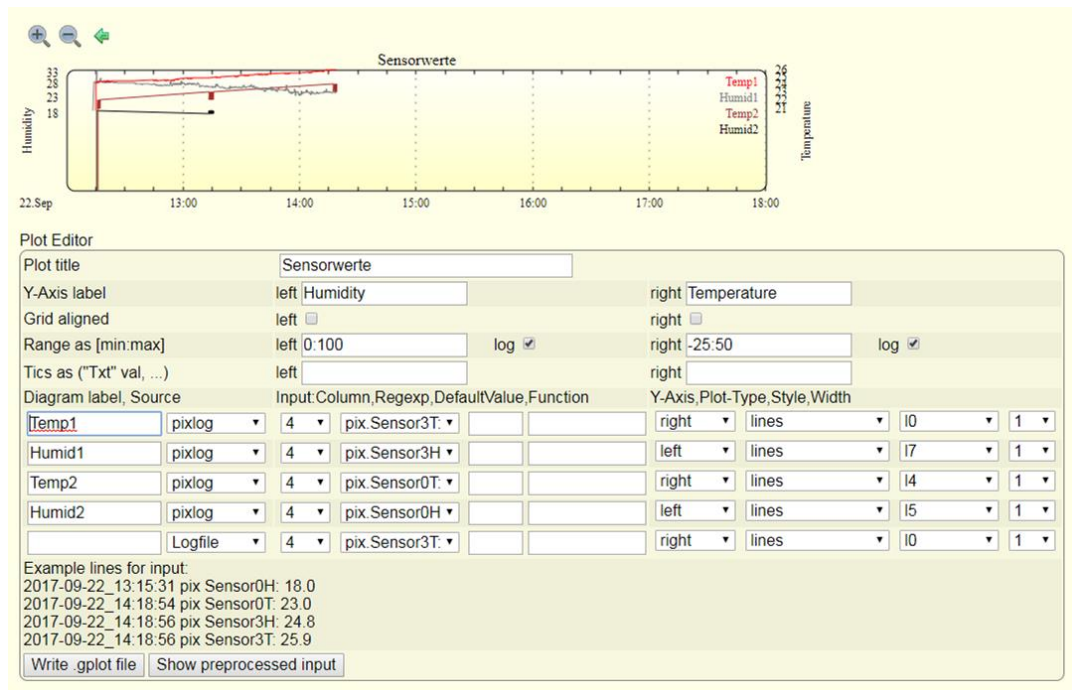


Figure 103: FHEM - Diagram of the measured sensor values

NOTICE

The DHT11 sensor (Temp2 and Humid2) only has an accuracy of 1°C or 1%. A new measuring point is only created if the readings, i.e. the measured value changes.

10.4.2. Debouncing and combining of inputs and outputs

For the first step, a digital input (DI0) should be debounced and turn on a relay (Relay0) afterwards. The following command is used to debounce the input. Digital inputs DI0 and DI1 must have a constant value over a time of $10 \cdot 100 \text{ ms} = 1 \text{ s}$, i.e. they are debounced.

```
set pix digitaldebounce01 10
```

To turn on a relay, create a “notify” with the following command:

```
define n_relay notify pix:DigitalIn0:.* set pix RelayOut0 $EVTPART1
```

The relay is set to “on” or “off” according to the state (\$EVTPART1) of DigitalIn0, the digital input reacts with a delayed of 1 s in the ideal case.

In the second step, an off delay is implemented using digital input DI2 and relay1. First the digital input is debounced, this time only 100 ms.

```
set pix digitaldebounce23 1
```

Now a DOIF module with the name “TimedSW” is created:

```
define TimedSW DOIF ([pix:DigitalIn2:"(on)"]) (set pix relayout1 on,set pix digitaldebounce23 50)
DOELSE (set pix relayout1 off,set pix digitaldebounce23 1)
```

If DigitalIn2 is “on”, in the second step the Relay1 is set to “on” and additionally the debounce value is set to 50 (= 5 s). If DigitalIn2 is “off”, Relay1 is set to “off” and the debounce value is reset (0.1 s). This ensures that the relay is switched on immediately after a short debouncing, but switched off after a delay of 5 seconds.

In the third step, two conditions are linked together. Relay2 is only switched if a voltage greater than 5V is measured at analog input AI0 and digital input DI4 is active at the same time. We create a DOIF module with the following command:

```
define Condi DOIF ([pix:DigitalIn4:"(on)"] and [pix:AnalogIn0] > 5.0) (set pix relayout2 on) DOELSE
(set pix relayout2 off)
```

10.4.3. Controlling servo motors

This example is intended to illustrate the control of a servo motor in FHEM. By default, the servo mode is active in the PWM settings and therefore does not have to be configured. However, the outputs of the servo channels must be activated so that a signal is sent. According to the description in chapter 14 for configuration of the PWM channels, the value “24” must be written to PWMxCtrl0. For example, use

```
set pix pwm0ctrl0 24
```

to activate the two PWM channels PWM0A and PWM0B. The servo motor can be controlled via the fields PWM0A and PWM0B.

Please note that the value 0 corresponds to the minimum position and the value 16000 to the maximum position, for example:

```
set pix pwm0a 9000
```

As a default, value 0 is used for PWMxA/B in servo mode. You would like to start with a different value? Then set this value first and then activate the corresponding channel.

10.5. Frequently Asked Questions (FAQs)

Why do the Readings for sensor values shows "Sensor not connected" or "Function not enabled"?

In this case, the function to read the sensor values was not enabled for the corresponding GPIO channel ("_GPIOxCtrl") or this setting was correct, but the sensor is not properly connected. Enable the sensor functionality or check the wiring of the sensor.

While adding the device, the error "ioctl.ph is not available but needed" appears.

The PiXtend V2 module for FHEM requires ioctl.ph to access the SPI, but this file was not found. There is probably an error in the installation of Raspbian. For this reason, we recommend using the images tested by us.

While adding the device, the following appears: "Error!

Device not xxx. Please change access rights for fhem user ...".

The user "fhem" needs the rights for the SPI interface and the GPIOs of the Raspberry Pi. The user is assigned the rights by adding it to the appropriate groups:

```
sudo adduser fhem gpio; sudo adduser fhem spi; sudo reboot
```

Only "error" is displayed as "STATE", what is the problem?

The communication between the PiXtend V2 and Raspberry Pi is faulty. Check if these two devices are connected properly, the switches on the PiXtend V2 are set to the correct position. And when the PiXtend V2 is not in the safe state and the SPI is enabled on the Raspberry Pi?

PiXtend Python Library V2 (PPLV2)

The programming language Python is a universal and interpretable language, its form is concise and readable and thus facilitates programming. A major feature of Python is, for example, that the program code is structured by indenting the code⁵. The large library is another advantage, there is documentation for almost every application.



Figure 104: Python - image source: <https://www.python.org/>, The Python Brochure

Python is already integrated in the Raspbian image⁶ and after the first start you can immediately start programming Python on the Raspberry Pi.

The access to the on-board GPIOs or the SPI bus works “out of the box”.

The PPLV2 is open source and can be used, modified and extended by anyone.

The latest versions of all documents and software components can be found in the download section of our website at <https://www.pixtend.de>.

11.1. Requirements

The driver support from the PPLV2 refers to the PiXtend V2 for Python 2.7.9 and Python 3. We recommend one of these Raspberry Pi models: B+, 2 B, 3 B, 3 B+, 4 B.

Download the PiXtend Basis 2.x.x.x SD card image from our Download section and use it as a starting point for your projects.

Alternatively, you can use an original Raspbian Buster image.

You can access the Raspberry Pi either directly with a keyboard and monitor or via SSH (TeraTerm / Putty) from a PC. If you are using an original Raspbian image, the Raspberry Pi needs an active Internet connection so you can follow the steps in this chapter.

We recommend reading the corresponding PiXtend V2 chapter Process Data and Control and Status Bytes in the Appendix. These contain information on the configuration of the PWM outputs and the analog inputs, as well as useful information about the GPIOs, the digital inputs and outputs, and the relays on the PiXtend V2.

11.2. Installation with the PiXtend V2 image

In the PiXtend Basis 2.x.x.x SD card image, the PiXtend Python Library V2 is already pre-installed and can be used immediately. For

⁵Source Wikipedia April 2017: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

⁶<https://www.raspberrypi.com/documentation/computers/getting-started.html#setting-up-your-raspberry-pi>

more information on creating an SD card for the Raspberry Pi, see Chapter 6.5 Preparing an SD card for your Raspberry Pi.

On the Raspberry Pi, you will find all files in the folder `/home/pi/pplv2/`.

11.3. Installation on the original Raspbian

11.3.1. Preparation

We start with an original Raspbian Buster image (Release: 26 September 2019). With this image, the PIXEL interface automatically starts after the first boot.

Since we do not need it here, we will disable it. A monitor, keyboard and mouse are required for the following steps. Logging on via Ethernet using SSH does not work, because the SSH server is disabled by default.

Click on the terminal icon in the PIXEL interface's menu bar and open a console window. Maximize the window for best visibility.



Figure 105: Python - Raspbian PIXEL- Terminal icon in the task bar

Open the configuration on the Raspberry Pi:

```
sudo raspi-config
```

The configuration program for the Raspberry Pi is started. In under “Boot Options” menu, select “Desktop / CLI” and then in the sub-menu “Console Autologin”. The selection is as follows:

“Boot Options” → “Desktop / CLI” → “Console Autologin”

Activate the SPI bus in the raspi-config configuration program:

“Interfacing Options” → “SPI” → <Yes> → <Ok>

The SSH server must be enabled to access the Raspberry Pi over the network via SSH and to run your own Python programs:

“Interfacing Options” → “SSH” → <Yes> → <Ok>

To exit the configuration program, press the Tab key twice until the word <Finish> is highlighted in red, then press Enter.

After these changes, a reboot must be performed. If raspi-config does not automatically offer a reboot when exiting the program, enter the following command:

```
sudo reboot
```

The next step is to download and install the necessary components, for this Raspberry Pi needs an active Internet connection.

11.3.2. PiXtend Python Library V2 Installation

For the installation of the PPLV2, we first create a directory for the PPLV2 package. Download the package and unzip it into the created directory, then install the PPLV2 package. Now the PiXtend Python Library V2 can be used globally in all Python 2.7.9 and Python 3 programs.

Installing a Python package can cause a lot of output on the console, this is perfectly normal.

Perform the following steps in sequence:

```
mkdir pplv2
wget https://files.kontron-electronics.de/pixtend/v2/software/pplv2_v0.1.x.zip
unzip pplv2_v0.1.x.zip -d ./pplv2/
cd pplv2
sudo python setup.py install
sudo python3 setup.py install
```

If the last command was successfully executed, the PiXtend Python Library V2 is now installed and can be used in your own Python programs. The installation of the PPLV2 package can be checked with `pip freeze` and `pip3 freeze`. The list of installed packages for both Python versions now contains the entry `pixtendlbv2 == 0.1.x`. Which version was installed exactly can be found in the "version_XXX.txt" file inside of the Zip file.

11.4. Programming

Our example program is already on the Raspberry Pi and is part of the PiXtend Python Library V2, it is in the folder `/home/pi/pplv2/examples`.

Those who work directly at Raspberry Pi can get started right away. If you connect to the Raspberry Pi via network, then you should first download an SSH client program "Putty".

11.4.1. Example directory

After the login, we are in the pi user home directory (`/home/pi`). Here you will find the folder `pplv2` which was created in the beginning, and which already exists on our SD card. It contains the source code of the PiXtend Python Library V2 and the sub-folder with the examples.

We change to the directory with the examples:

```
cd ~/pplv2/examples/
```

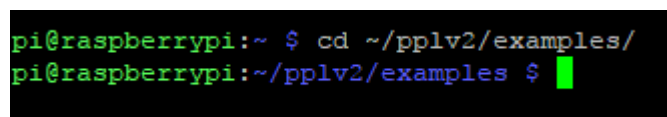


Figure 106: Python - Console - PPLV2 directory

11.4.2. Executing an example

We will use a PiXtend V2 -S- for this example and start the Python PiXtend V2 demo program. All digital inputs are read and all digital outputs are switched alternately. The clacking of the relays is easy to hear and the LEDs indicate the state.

The following command starts the example:

```
sudo python pixtendv2s_demo.py
```

If you have a PiXtend V2 -L-, try the following:

```
sudo python pixtendv2l_demo.py
```

```
pi@raspberrypi:~/pplv2/examples $ sudo python pxtendv2_demo.py
```

```
PiXtend Python Library v2 (PPLv2) demo.
```

```
Running Main Program - Hit Ctrl + C to exit  
One time configuration: Setting the relays and digital outputs in an alternating pattern  
The value False = OFF and the value True = ON
```

```
Cycle No.: 22
```

```
PiXtend V2 -S- Info:  
Firmware:      2  
Hardware:     20  
Model:        S
```

```
Digital Inputs:  
DigitalIn0:   False  
DigitalIn1:   False  
DigitalIn2:   False  
DigitalIn3:   False  
DigitalIn4:   False  
DigitalIn5:   False  
DigitalIn6:   False  
DigitalIn7:   False
```

```
Digital Outputs:  
DigitalOut0:  False  
DigitalOut1:  True  
DigitalOut2:  False  
DigitalOut3:  True
```

```
Relays:  
Relay0:       False  
Relay1:       True  
Relay2:       False  
Relay3:       True
```

```
^C
```

```
PiXtend Python Library v2 (PPLv2) demo finished.  
pi@raspberrypi:~/pplv2/examples $ █
```

Figure 107: Python - PiXtend Python Library V2 demo program

After starting the program, the digital outputs and the relays change their states about every second. The console displays the current status in text form, "False" means off and "True" means on. There is a cycle counter that increases by one every second.

The Python program can be terminated with the keys Ctrl+C.

11.4.3. Creating your own program

Open an editor or program with which you can write or edit Python programs. Create a new empty Python file. Save this file, e.g., under the name "programm1.py".

We are using a PiXtend V2 -S- as an example here, but the same functions are also available on the PiXtend V2 -L-. Replace the "S" with an "L".

In our first program, relay 0 switched every second.

The program could look like this:

```
#!/usr/bin/env python
```

```
from pxtendv2s import PiXtendV2S
```

```
import time
```

```
p = PiXtendV2S()
```

```
while True:
```

```
    if p.relay0 == p.OFF:
```

```
        p.relay0 = p.ON
```

```
    else:
```

```
        p.relay0 = p.OFF
```

```
    time.sleep(1)
```

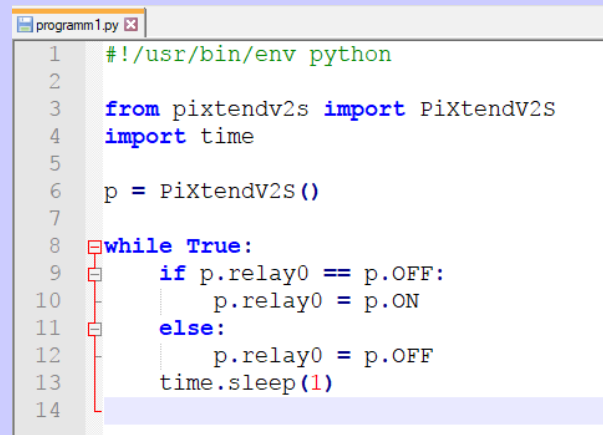


Figure 108: Python - simple demo program

Program explanation:

- In the first line, we write that it is a Python program.
- After that the PiXtendV2S class has to be imported to get access to the features and functions of the PiXtend V2 -S-. This also makes communication with the micro-controller and DAC possible.
- To wait at the end of the program, we import the time class.
- With `p = PiXtendV2S`, an object is created with which we can work with, the communication with the micro-controller is automatically established in the background.
- In a while loop, we check whether relay 0 is On or Off and respond accordingly.
- The constants ON and OFF can be replaced with True and False. Working with names is intended help to clarify the actual procedure.
- At the end, we wait a second to reduce the strain on relay 0.

Transfer the Python program to the Raspberry Pi and run it with the following command:

```
sudo python programm1.py
```

After starting, the relay 0 is switched on and off, as programmed by us and this happens now every second.

To exit the program, use the the key combination Ctrl+C twice. The first time exits the while loop, and the second time terminates the background communication with the PiXtend V2 -S- board. In your own programs, you can execute the "close" function of the PiXtendV2S class before closing the program. The communication with the micro-controller is terminated, the SPI driver is closed, and the asynchronous communication is stopped.

NOTICE

Please note that on a Raspberry Pi there may only be one single instance of the PiXtendV2S or PiXtendV2L class. It does not matter how many Python programs are created. This restriction is related to the SPI bus of the Raspberry Pi, it may only be used by one program at a time.

11.4.4. Short Overview

The PPLV2 consists of multiple Python classes: PiXtendV2Core, PiXtendV2S and PiXtendV2L. The PiXtendV2Core class represents a basic class, which contains many basic functions and properties, but is not executable by itself. The PiXtendV2S/ PiXtendV2L classes inherit these properties and functions and forms all the remaining properties of the PiXtend V2.

After instantiation of the PiXtendV2S/PiXtendV2L class, the communication with the micro-controller on the PiXtend V2 board is automatically executed in the background. The user can easily use all digital and analog inputs and outputs without having to worry about the communication. An explicit wait after each cycle is not required.

In the zip file (pplv2_v0.1.x.zip) or in the folder pplv2 on the Raspberry Pi, the subfolder "doc" contains two HTML files that contain the API documentation for the Python classes mentioned. The help file for the PiXtendV2S/PiXtendV2L classes contains all usable functions and properties. Look there if you want to use the GPIOs, the temperature and humidity inputs or the PWM outputs.

The following table provides a brief overview of the most important and common features and properties of the Python class PiXtendV2S⁷:

| Name | Type | Description |
|-------------------|---------------|--|
| | | Instantiation of the PiXtendV2S class and start of the communication with the PiXtend V2 -S- micro-controller. Example: <code>p = PiXtendV2S()</code> Instantiation of the PiXtendV2L class and start of the communication with the PiXtend V2 -L- micro-controller. Example: <code>p = PiXtendV2L()</code> |
| close | Function | To terminate the Python program, execute the close function and then the PiXtendV2S instance can be deleted. The close function resets all internal variables and objects, closes the SPI driver and ends the background communication. Example: <code>p.close()</code> <code>p = None</code> |
| digital_in0 .. 7 | Property (r) | With these eight read-only properties (digital_in0 to digital_in7), the states of the digital inputs can be read by the PiXtend V2 -S-. The properties deliver either the value False for off or the value True for on. |
| digital_out0 .. 3 | Property (rw) | The four digital outputs can be both read and written via the properties digital_out0 to digital_out3. If the property is assigned the value False, the corresponding digital output switches off or if the value True is assigned, the output switches on. |
| relay0 .. 3 | Property (rw) | The four relays can be both read and written via the properties relay0 to relay3. If the property is assigned the value False, the corresponding relay switches off or if the value True is assigned, the relay switches on. |
| analog_in0 .. 1 | Property (r) | With these two properties, the PiXtend V2 -S- analog inputs can be read. |
| hardware | Property (r) | With this read only property, the hardware version can be queried. For example, the value 21 stands for board version 2.1. |
| firmware | Property (r) | The firmware version of the micro-controller can be determined via this read only property. |
| ON | Constant | This corresponds to the value True. |
| OFF | Constant | This corresponds to the value False. |

11.4.5. Starting Python automatically

After a reboot/power-up of the Raspberry Pi, a Python program does not start automatically. The Python program can be started with the following command, replace myprogram.py with your own program name

```
sudo python myprogram.py
```

We can also run this command automatically when the Linux system boots.

The change is done quickly in the Linux console:

```
sudo nano /etc/rc.local
```

⁷The letters (r) and (rw) after a property refer to r = read only and rw = read and write.

Refer to the API documentation in the doc folder for the functions and properties for the PiXtend V2 -L-.

The file “rc.local” opens with some content. We add two new lines before the line “exit 0” with the assumption that the program to be started is located in the home directory:

```
cd /home/pi/  
sudo python myprogram.py &
```

The “&” is not a typo. This starts the Python program as a process in the background.

Save and exit with CTRL+X → Reboot

CAUTION

If the program contains an error, the digital outputs are uncontrolled or very fast, this can damage the connected peripherals. Since the faulty program is set up as an “autostart program”, rebooting will not help to solve the problem. Always make a backup copy of your work.

11.4.6. Using the serial interface

The use of the serial port is not included in the PPLV2 package, use pySerial, it concerns the Raspberry Pi and Python. With the Raspbian version mentioned in this documentation, the serial interface runs, for example, via the Linux device `/dev/ttyS0`.

Please the output of the Linux console is displayed. Before use, edit `/boot/cmdline.txt` and remove the `console=serial0,115200` part from the first line, or use `raspi-config` to turn off console output.

11.4.7. Frequently Asked Questions (FAQs)

The program cannot be started. There is a Python error message about an invalid iteration.

There is probably a problem with the indentation of the different lines of code in the program. Check again that the indentation is correct for each line. We always recommend four spaces per indentation level, if a space is missing or spaces and tabs are mixed, Python will display an error message. In Notepad ++ e.g. you can see the “non-printable” characters and you see very quickly where something is wrong, Python also tells you in which line it found the error.

Can I run my program without sudo?

This is actually possible with Python in the release of Raspbian used here, but we still recommend using sudo to allow Python to perform system functions. Otherwise, problems may arise whose cause cannot be easily explained.

Why do the outputs or relays turn off by themselves or blink when they are switched on in the program?

On a Raspberry Pi, the PiXtendV2S or PiXtendV2L class may only be used once, depending on the PiXtend board. Multiple uses are not possible because of the SPI bus, no matter how many Python programs you want to create. There may only ever be one master or main program with a single PiXtend V2 object that communicates with the PiXtend V2 board, not several.

Where and how can I set the cycle time for the PPLv2?

The cycle time, as already mentioned in chapter 6.2 SPI Communication, Data Transmission and Cycle Time, can be changed by the user. The cycle time indicates at which intervals the PPLv2 exchanges data with the PiXtend V2 board.

To set the optimal cycle time for each Python project, the base class of PPLv2 has the parameter “com_interval”. Depending on the existing PiXtend V2 Board, you can easily adjust the cycle time when instantiating a PiXtend V2 Board object. By default, the PPLv2 works with a cycle time of 30 ms (0.03 seconds), see the API documentation in the “doc” subfolder of the “pplv2” folder.

Example for the PiXtend V2 -L-, 100 ms cycle time:

```
#!/usr/bin/env python
from pixtendv2s import PiXtendV2S
import time
p = PiXtendV2S(com_interval=0.1)
while True:
    if p.relay0 == p.OFF:
        p.relay0 = p.ON
    else:
        p.relay0 = p.OFF
    time.sleep(1)
```

Example for PiXtend V2 -L-, 50 ms cycle time:

```
#!/usr/bin/env python
from pixtendv2l import PiXtendV2L
import time
p = PiXtendV2L(com_interval=0.05)
while True:
    if p.relay0 == p.OFF:
        p.relay0 = p.ON
    else:
        p.relay0 = p.OFF
    time.sleep(1)
```

Appendix

This chapter shows which process data is exchanged between the PiXtend micro-controller and Raspberry Pi (RPI). We speak of process data, since it is the data from the inputs and outputs on the PiXtend board.

In the PiXtend example programs for CODESYS, the data is correctly transferred, evaluated and, if necessary, prepared.

With the Linux tools and the use of the PiXtend library (pxdev) as well as the PiXtend Python library, it is important to know how the data can be influenced and evaluated.

The process data is defined in the PiXtend micro-controller and therefore independent of the programming language or Linux software used.

The latest versions of all documents and software components can be found in the download section of our website at <https://www.pixtend.de>.

12.1. PiXtend V2 -S-

12.1.1. Process data

12.1.1.1 Overview of the process data

There are two types of process data:

- Process data transmitted from the Raspberry Pi to the PiXtend
- Process data transferred from the PiXtend to the Raspberry Pi

12.1.1.1.1 Process data transmitted from the Raspberry Pi to the PiXtend

Data for digital outputs, relays and GPIOs (as outputs) PWM data, Retain data

- DigitalOut
- RelayOut
- GPIOOut
- PWM0X (L/H)
- PWM1X (L/H)
- RetainDataOutX

12.1.1.1.2 Process data transmitted from the Raspberry Pi to the PiXtend DAC

Date for analog outputs

- AnalogOutX (L/H) –

12.1.1.1.3 Process data transferred from the PiXtend to the Raspberry Pi

Data of the digital and analog inputs, GPIOs (as inputs)

Temperature and humidity of DHT11/22 or AM2302 sensors, Retain data

- DigitalIn
- AnalogInX (L/H)
- GPIOIn
- TempX (L/H)
- HumidX (L/H)
- RetainDataInX

12.1.1.2 SPI bus protocol overview

| INPUT | | | | OUTPUT | | |
|-------------------------|---------------------|-----------|----------|-------------------------|----------------------|-----------|
| MC (Out) --> RasPi (In) | | | | RasPi (Out) --> MC (In) | | |
| | Name | used Bits | Byte Idx | | Name | used Bits |
| HeaderIn | Firmware | 8 | 0 | HeaderOut | ModelOut | 8 |
| | Hardware | 8 | 1 | | UCMode | 8 |
| | ModelIn | 8 | 2 | | UCCtrl0 | 8 |
| | UCState | 8 | 3 | | UCCtrl1 | 8 |
| | UCWarnings | 8 | 4 | | Reserved | 0 |
| | Reserved | 0 | 5 | | Reserved | 0 |
| | Reserved | 0 | 6 | | Reserved | 0 |
| | CRCHeaderInL | 8 | 7 | | CRCHeaderOutL | 8 |
| | CRCHeaderInH | 8 | 8 | | CRCHeaderOutH | 8 |
| DataIn | DigitalIn | 8 | 9 | DataOut | DigitalInDebounce01 | 8 |
| | AnalogIn0L | 8 | 10 | | DigitalInDebounce23 | 8 |
| | AnalogIn0H | 2 | 11 | | DigitalInDebounce45 | 8 |
| | AnalogIn1L | 8 | 12 | | DigitalInDebounce67 | 8 |
| | AnalogIn1H | 2 | 13 | | DigitalOut | 4 |
| | GPIOIn | 4 | 14 | | RelayOut | 4 |
| | Temp0L | 8 | 15 | | GPIOCtrl | 8 |
| | Temp0H | 8 | 16 | | GPIOOut | 4 |
| | Humid0L | 8 | 17 | | GPiodebounce01 | 8 |
| | Humid0H | 8 | 18 | | GPiodebounce23 | 8 |
| | Temp1L | 8 | 19 | | PWM0Ctrl0 | 8 |
| | Temp1H | 8 | 20 | | PWM0Ctrl1L | 8 |
| | Humid1L | 8 | 21 | | PWM0Ctrl1H | 8 |
| | Humid1H | 8 | 22 | | PWM0AL | 8 |
| | Temp2L | 8 | 23 | | PWM0AH | 8 |
| | Temp2H | 8 | 24 | | PWM0BL | 8 |
| | Humid2L | 8 | 25 | | PWM0BH | 8 |
| | Humid2H | 8 | 26 | | PWM1Ctrl0 | 8 |
| | Temp3L | 8 | 27 | | PWM1Ctrl1L | 8 |
| | Temp3H | 8 | 28 | | PWM1Ctrl1H | 8 |
| | Humid3L | 8 | 29 | | PWM1AL | 8 |
| | Humid3H | 8 | 30 | | PWM1AH | 8 |
| | Reserved | 0 | 31 | | PWM1BL | 8 |
| | Reserved | 0 | 32 | | PWM1BH | 8 |
| | RetainDataIn0 .. 31 | all | 33 | | RetainDataOut0 .. 31 | all |
| | CRCDataInL | 8 | 65 | | CRCDataOutL | 8 |
| | CRCDataInH | 8 | 66 | | CRCDataOutH | 8 |

Table 5: PiXtend V2 -S- SPI bus protocol overview

This chapter contains a description of every byte of the SPI protocol, partly with application notes and calculation examples, see PWMs. The SPI protocol for the PiXtend V2 -S- includes a total of 67 bytes, table 7 shows a shortened form, here the 32 bytes of the Retain memory were summarized.

If an entry is a single byte, the SPI protocol only contains the name or designation of the byte. With 16-bit values (2 bytes), for example temperature, the value is divided into two individual bytes. This can be recognized by the addition's "L" for the low byte and "H" for the high byte. If you want to implement the SPI protocol, make sure that you assign the bytes for high and low accordingly.

The CRC calculation for the header and data is not explicitly listed, please refer to the source code for our Linux tools (pxdev), available on SourceForge, our website in the download section or in our Python modules for further details. We have provided a corresponding implementation that you can use it as a template for your programming language.

12.1.1.3 SPI bus configuration

To address PiXtend V2 SPI devices from own applications, the following SPI bus settings should be considered:

- SPI-Mode 0 (CPOL = 0, CPHA = 0)
- SPI speed: 700 kHz
- SPI ChipSelect 0 (CS0): PiXtend V2 micro-controller
- SPI ChipSelect 1 (CS1): PiXtend DAC or CAN interface (for V2 -L-)
- GPIO24, wiringPi BCM numbering (wiringPi's own numbering GPIO 5) of the Raspberry Pi must be set to "high" / logical "1" → SPI Enable.

Cycle time limit:

- Minimum cycle time V2 -S-: 2.5 ms
- Minimum cycle time V2 -L-: 5 ms

This is the minimum time interval between two communication processes with the micro-controller. Longer cycle times are better.

We recommend reading over the source code of the PiXtend library and/or the Python modules to get a better feeling for using the SPI bus.

12.1.1.4 Output data

Output data is the data that the Raspberry Pi transmits to the PiXtend micro-controller, such as digital outputs or relays.

12.1.1.4.1 DigitalOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|-----|-----|-----|-----|
| Name | - | - | - | - | DO3 | DO2 | DO1 | DO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 6: Bits of the DigitalOut bytes

Bit 0 - 3

The four digital outputs are organized in one byte. The least significant bit (LSB) contains the status of DO0. Bit 7 (MSB), bit 6, bit 5 and bit 4 are not assigned. The start value after the power-up or reset of the micro-controller is “0” for the entire DigitalOut byte. The outputs are deactivated at the start and in SafeState mode.

By writing a “1” to one of the bits (0 - 3), the corresponding output is activated. The corresponding LED on the PiXtend board lights up.

The digital outputs on the PiXtend V2 have a separate power supply. You can see from the LED “VCC DO” whether the power supply is connected and supplied with voltage.

The LEDs of the digital outputs also light up if the VCC-DO supply has not been connected, but no voltage is “visible” on the DO pins.

To test the function of the outputs, the value 255 (decimal) or 0xFF (hex) can be written on the DigitalOut byte. All four outputs are activated. The fact that bits 4 to 7 are also set here has no effect.

12.1.1.4.2 RelayOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|--------|--------|--------|--------|
| Name | - | - | - | - | RELAY3 | RELAY2 | RELAY1 | RELAY0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 7: Bits of the RelayOut bytes

Bit 0 - 3

The four relay outputs are organized in one byte. The least significant bit (LSB) contains the status of RELAY0. Bit 7 (MSB) to bit 4 are not assigned. The start value after the power-up or reset of the micro-controller is “0” for the entire RelayOut byte. The relays are thus deactivated at the Start and in SafeState mode.

By writing a “1” to one of the bits (0 - 3), the corresponding relay is activated. The corresponding LED on the PiXtend board lights up and the relay switches can be heard.

To test the function of the relays, the value 255 (decimal) or 0xFF (hex) can be written on the RelayOut byte. All four relays are activated. The fact that bits 4 to 7 are also set here has no effect.

12.1.1.4.3 GPIOOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|-------|-------|-------|-------|
| Name | - | - | - | - | GPIO3 | GPIO2 | GPIO1 | GPIO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 8: Bits of the GPIOOut bytes

Bit 0 - 3

The GPIO outputs are organized in one byte. The least significant bit (LSB) contains the status of GPIO0. Bit 7 (MSB) to bit 4 are not assigned. The start value after the power-up or reset of the micro-controller is "0" for the entire GPIOOut byte. The GPIOs are thus deactivated at the Start and in SafeState mode.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information can be found in the chapter Control and Status Bytes. By writing a "1" into one of the bits (0 - 3), the corresponding GPIO is activated (if configured as output). If the GPIO(s) are configured as input or for the mentioned sensors, changing the values in GPIOOut has no effect*.

To test the function of the GPIOs, they can be configured as outputs. Then the value 255 (decimal) or 0xFF (hex) can be written for the GPIOOut byte. All four GPIO outputs are activated. The fact that bits 4 to 7 are also set here has no effect.

* If a GPIO is configured as an input with pull-ups and the corresponding bit is set to "1" in GPIOOut, a pull-up resistor is activated at this GPIO. This gives the GPIO input a high-impedance (~35 kOhm) reference to 5V and it no longer floats. Without external circuitry, the input now remains at a high level. By default, the GPIO inputs are floating inputs without a defined level (without external circuitry) and the pull-up resistors should generally be activated with the bit "GPiOPullUpEnable".

12.1.1.4.4 PWM0X (L/H) – 16-bit resolution

The PWM units of the PiXtend can be operated in four different modes. In each mode, the PWM process data has a different effect. For this reason, we will discuss the four modes "Servo Mode", "Duty Cycle Mode", "Universal Mode" and "Frequency Mode" separately in this section.

12.1.1.4.5 PWM0Ctrl – for 16-bit PWMs

The PWM0X bytes are the process data for the two 16-bit PWM channels that have the full designation PWM0A and PWM0B on the building block.

12.1.1.4.5.1. Frequency mode

In frequency mode, the bytes PWM0XL and PWM0XH contain a related 16-bit data word. The data word is used to set the frequency of the respective channel. The duty cycle for channel A and B is always 50% and cannot be changed.

This mode is ideal when many different frequencies are required but the duty cycle is not important. This is, for example, the case when the PWM channels are used to control stepper motor drivers that only react to signal edges and do not require a variable duty cycle. Thus, PiXtend can control the speed of up to four stepper motors (2x 16 bit PWM0X, 2x 8 bit PWM1X).

PWM0AL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 15: Bits of the PWM0AL bytes

PWM0AH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 16: Bits of the PWM0AH bytes

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWM0Ctrl0, the fine adjustment via PWM0XL/H. The maximum frequency in this mode is 20 kHz. When setting higher frequencies, the output signal is limited to 20 kHz.

The frequency is calculated as follows:

$$\text{Frequency channel X} = 16 \text{ MHz} / 2 / \text{Prescaler} / \text{PWM0X}$$

An example

A frequency of 500 Hz should be output on channel A and 250 Hz on channel B.

- Activate channels A and B, select frequency mode, configure prescaler to 64

→ PWM0Ctrl0 = 123 (decimal) or 01111011b (binary)

- PWM0A (16-bit data word) = 250 (decimal)

→ Frequency channel A = $16 \text{ MHz} / 2 / 64 / 250 = 500 \text{ Hz}$

- PWM0B (16-bit data word) = 500 (decimal)

→ Frequency channel B = $16 \text{ MHz} / 2 / 64 / 500 = 250 \text{ Hz}$

If the PWM signals should not be continuously present at the outputs, the channel(s) may be activated/deactivated in every cycle if required.

NOTICE

A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.1.1.4.5.2. Servo mode

The servo mode is specially designed for the requirements of model construction servos. The period duration is fixed at 50 Hz (20 ms). At the beginning of each period, the signal is at least 1 ms (minimum position) or maximum 2 ms (maximum position) at the high level. The remaining time is the signal at the low level.

The PWM0X bytes L and H contain a related 16-bit data word. PWM0XL is the low byte and PWM0XH the high byte. The “X” is replaced by the letter of the respective PWM channel (“A” or “B”).

PWM0XL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 9: PWM0XL byte

PWM0XH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Table 10: PWM0XH byte

Adjustable value range

Smallest value: 0 (decimal) → 1 ms, minimum position

Maximum value: 16000 (decimal) → 2 ms, maximum position

Values greater than 16000 are limited by the controller and act as 16000. In the range between 0 and 16000, the movement of the servo is linear. The start value of the PWM0X byte, after a power-up/reset, is “0”.

12.1.1.4.5.3. Duty cycle mode

In duty cycle mode, the bytes PWM0XL and PWM0XH contain a related 16-bit data word. The data word is used to set the duty cycle. The “X” is replaced by the letter of the respective PWM channel (“A” or “B”). A separate duty cycle can be assigned to each channel. The frequency and period duration are set together for both channels.

This mode is ideal for controlling two drives or fans from 0% to 100%, independent of their speed.

PWM0XL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 11: PWM0XL byte

PWM0XH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 12: PWM0XH byte

Which duty cycle results from a certain value depends not only on the PWM0XL/H values, but additionally on the control bytes PWM0Ctrl1L and PWM0Ctrl1H. The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWM0Ctrl0, the fine adjustment via PWM0Ctrl1L/H.

More information about the confirmation of the PWM0 channels may be found in the section: 14.1.2.3.7 PWM0Ctrl.-for 16-bit PWMs

Calculation formula - PWM0 - 16 Bit

Frequency: $f = 16 \text{ MHz} / 2 / \text{Prescaler} / \text{PWM0Ctrl1}$

Duty cycle: $\%on = 100 \% * \text{PWM0X} / \text{PWM0Ctrl1}$

An example

Activate Duty-Cycle Mode and channels A and B, prescaler to 1024:

PWM0Ctrl0 = 249 (decimal) or 11111001b (binary)

In PWM0Ctrl1L/H, the following 16-bit value is written: 5000 (decimal).

The 16-bit value 5000 (decimal) is written in PWM0XL/H. The duty cycle is thus 50%.

If a higher value is written in PWM0XL/H than in PWM0Ctrl1L/H, the PWM channel remains continuously at logical “1”. Continuous logic “0” is reached by the value “0” in PWM0XL/H.

In our example, the frequency is 1.56 Hz.

We recommend that you first test the implementation of the PWM functions in your own programs without connected devices/actuators and check them with an oscilloscope if possible.

Set the PWM values with care!



Some actuators can be damaged or destroyed because of incorrect frequencies or duty cycles. Always check your programs with an oscilloscope before connecting devices to the PWM channels.

12.1.1.4.5.4. Universal mode

In universal mode, the bytes PWM0AL and PWM0AH contain a related 16-bit data word. The data word is used to set the duty cycle of channel A.

The B channel is not adjustable in this mode and outputs a PWM with 50% duty cycle and half the frequency of the A channel.

This mode is suitable for applications with different frequencies and when one channel of the duty cycle is to be configured.

PWM0AL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 13: PWM0AL byte

PWM0AH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 14: PWM0AH byte

The configuration of the A channel is handled in the same way as in duty cycle mode.

A change of the bytes PWM0BL and PWM0BH has no effect. The B channel depends on the frequency configuration of the A channel.

12.1.1.4.6 PWM1X (L/H) – 8-bit resolution

The PWM units of the PiXtend can be operated in four different modes. In each mode, the PWM process data has a different effect. For this reason, we will discuss the four modes “Servo Mode”, “Duty Cycle Mode”, “Universal Mode” and “Frequency Mode” separately in this section.

More information about configuration and modes of the PWM1 outputs may be found in the section: 14.1.2.3.8 PWM1Ctrl. - for 8-bit PWMs.

The PWM1X bytes are the process data for the two 8-bit PWM channels that have the full designation PWM1A and PWM1B on the building block.

12.1.1.4.6.1. Servo mode

The servo mode is specially designed for the requirements of model construction servos. The period duration is fixed at 50 Hz (20 ms). At the beginning of each period, the signal is at least 1 ms (minimum position) or maximum 2 ms (maximum position) at the high level. The remaining time is the signal at the low level.

The PWM1 channels have two PWM1X bytes (L and H) but only the low byte (L) is used. CODESYS only uses the low byte. The “X” is replaced by the letter of the respective PWM channel (“A” or “B”).

PWM1XL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 17: Bits of the PWM1XL bytes

PWM1XH – not used

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 18: Bits of the PWM1XH byte

Adjustable value range

Smallest value: 0 (decimal) → 1 ms, minimum position

Maximum value 125 (decimal) → 2 ms, maximum position

Values greater than 125 result in maximum position (two milliseconds). In the range between 0 and 125, the movement of the servo is linear.

The start value of the PWM1XL byte, after a power-up/reset, is “0”.

12.1.1.4.6.2. Duty cycle mode

The PWM1 channels have two PWM1X bytes (L and H), but only the low byte (L) is used. CODESYS offers only the low byte to be used. The PWM1XL byte is used to set the duty cycle. The "X" is replaced by the letter of the respective PWM channel ("A" or "B"). A separate duty cycle can be assigned to each channel.

The frequency and period duration are set for both channels together. This mode is ideal for controlling two drives or fans from 0% to 100%, independent of their speed.

PWM1XL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 19: Bits of the PWM1XL bytes

PWM1XH – not used

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 20: Bits of the PWM1XH bytes

Which duty cycle results from a certain value depends exclusively on the PWM1XL values. The frequency cannot be set as precisely for the PWM1 channels as with the 16-bit PWM0 channels.

Two factors influence the frequency:

1. Set prescaler in PWM1Ctrl0
2. If a "1" is written to PWM1Ctrl1, the frequency set by the prescaler is doubled

Calculation of the frequency: $f = 16 \text{ MHz} / 2 / \text{prescaler} / 255$

Dividing by 2 is not necessary if a "1" is written to PWM1Ctrl1.

For further information on the configuration, refer to the section: 14.1.2.3.8 PWM1Ctrl. - for 8-bit PWMs PWM1Ctrl. - for 8-bit PWMs

We recommend that you first test the implementation of the PWM functions in your own programs without connected devices/actuators and check them with an oscilloscope if possible.

Set the PWM values with care!

⚠ CAUTION

Some actuators can be damaged or destroyed because of incorrect frequencies or duty cycles. Always check your programs with an oscilloscope before connecting devices to the PWM channels.

12.1.1.4.6.3. Universal mode

The PWM1 channels have two PWM1X bytes (L and H) but only the low byte (L) is used. CODESYS only uses the low byte. The PWM1AL byte is used to set the duty cycle of channel A. The B channel is not adjustable in this mode and outputs a PWM signal with 50% duty cycle and half the frequency of the A channel.

This mode is suitable for applications with different frequencies and when one channel of the duty cycle is to be configured.

PWM1AL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 21: Bits of the PWM1AL bytes

PWM1AH – not used

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 22: Bits of the PWM1AH bytes

Calculation formula

Frequency: $f = 16 \text{ MHz} / 2 / \text{Prescaler} / \text{PWM1Ctrl1}$

Duty cycle: $\%on = 100 \% * \text{PWM1AL} / \text{PWM1Ctrl1}$

A change of the byte PWM1BL has no effect. The B channel only depends on the frequency configuration of the A channel.

12.1.1.4.6.4. Frequency mode

The PWM1 channels have two PWM1X bytes (L and H) but only the low byte (L) is used. CODESYS offers only the low byte to be used. The PWM1AL byte is used to set the frequency of the respective channel. The duty cycle for channel A and B is always 50% and cannot be changed.

This mode is ideal when many different frequencies are required but the duty cycle is not important. This is the case when the PWM channels are used to control stepper motor drivers that only react to signal edges and do not require a variable duty cycle. Thus, up to four stepper motors can be controlled with PiXtend at this speed (2x 16 bit PWM0X, 2x 8 bit PWM1X).

PWM1AL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 23: Bits of the PWM1AL bytes

PWM1AH – not used

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 24: Bits of the PWM1AH bytes

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWM1Ctrl0, the fine adjustment via PWM1XL. The maximum frequency in this mode is 20 kHz. When setting higher frequencies, the output signal is limited to 20 kHz.

The frequency is calculated as follows:

$$\text{Frequency: } f = 16 \text{ MHz} / 2 / \text{Prescaler} / \text{PWM1XL}$$

An example

A frequency of 500 Hz should be output on channel A and 250 Hz on channel B.

- Activate channels A and B, select frequency mode, configure prescaler to 256

→ PWM1Ctrl0 = 219 (decimal) / 11011011 (binary)

- PWM1AL = 63 (decimal)

→ Frequency channel A = $16 \text{ MHz} / 2 / 256 / 63 = 496 \text{ Hz}$

- PWM1BL = 250 (decimal)

→ Frequency channel B = $16 \text{ MHz} / 2 / 256 / 250 = 125 \text{ Hz}$

If required, the channel(s) may be activated/deactivated in every cycle, then the PWM signals are not continuously present at the outputs. In the example, the frequency of 500 Hz at the A channel is not perfectly adjusted.

It would have to be set to 62.5, which is not possible with an integer data type. Check the different prescaler settings for the desired frequency. Different prescaler settings can be used to set the same or similar frequency.

NOTICE

A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.1.1.4.7 RetainDataOutX

The Retain data consists of 32 bytes per direction (RetainDataOut and RetainDataIn). RetainDataOut is the data that is transferred from the Raspberry Pi to the PiXtend micro-controller for the actual backup. The storage location of the Retain data is the micro-controller.

For the 32 bytes, there is no default for the data format. In the application software, you can decide how this data is to be described and used. Under CODESYS the 32 bytes are offered as a byte array.

The RetainDataOutX are transferred in each cycle. If the Retain memory is activated, the data is stored in the micro-controller in case of a power failure or voltage interruption and is available again in the RetainDataIn bytes at the next start.

12.1.1.5 Output data – DAC

There is a Digital to Analog Converter (DAC) on the PiXtend V2 -S-, which is directly controlled by the Raspberry Pi and is responsible for the two analog outputs. The DAC is not part of the micro-controller and the data from the DAC is therefore not in the process image that is exchanged between the Raspberry Pi and the micro-controller.

The DAC has a data format defined by the chip manufacturer. For further information, please refer to the corresponding data sheet of the chip⁸. A 10-bit version of the chip is installed on the PiXtend.

12.1.1.5.1 AnalogOutX (L/H)

The DAC receives one 16-bit data word per channel. The DAC does not provide an “reply” - there is only one data direction. The X in AnalogOutX stands for channel A or B of the chip.

The chip is connected to the Raspberry Pi via the SPI bus (ChipSelect - CS1). First the high byte (AnalogOutXH) then the low byte (AnalogOutXL) is transmitted.

On the PiXtend V2 -S-, channel A is responsible for AO0 and channel B for AO1.

AnalogOutXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|----|----|----|----|----|----|---|---|
| Name | D5 | D4 | D3 | D2 | D1 | D0 | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 25: Bits of the AnalogOutL bytes

The 10-bit value for the output value is in AnalogOutXL and in AnalogOutXH - D0 (LSB) to D9 (MSB). The low byte AnalogOutXL contains the lower 6 bits. The bits “0” and “1” of AnalogOutXL are not evaluated.

AnalogOutXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|------|---|-----|-------|----|----|----|----|
| Name | /A-B | - | /GA | /SHDN | D9 | D8 | D7 | D6 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 26: Bits of the AnalogOutH bytes

The upper four bits of the 10-bit output value are located in AnalogOutXH. Three configuration bits are also located in the AnalogOutXH byte:

⁸Microchip MCP4812

Bit 4 – /SHDN

The abbreviation SHDN means “Output Shutdown Control”. The channel set by bit 7 (/A-B) can be activated or deactivated via the /SHDN bit. If a “1” is written in /SHDN, the channel is active.

Normally, as well as after a power up, the channel is on the value “0” and therefore deactivated (start value). When deactivated, the analog outputs on the PiXtend V2 -S- are set to a voltage of less than 50 mV. This is independent of the value to which the outputs are set.

Bit 5 – /GA

Setting the “Output Gain Selection”. For the intended operating case of the PiXtend V2 -S- (0 - 10V output voltage), the /GA bit must always be set to “0” (start value).

Bit 7 – /A-B

Setting the channel - A or B. If the bit contains a “1”, the 16-bit data word consisting of AnalogOutXL and AnalogOutXH is used for channel B. If bit 7 is set to “0” (start value), it is addressed for channel A.

All further information can be found in the data sheet of the DAC chip - microchip MCP4812.

12.1.1.6 Input data

Input data is the data that the PiXtend micro-controller transmits to the Raspberry Pi, for example the digital and analog inputs.

12.1.1.6.1 DigitalIn

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | DI7 | DI6 | DI5 | DI4 | DI3 | DI2 | DI1 | DI0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 27: Bits of the DigitalIn bytes

The eight digital inputs (DigitalIn) are organized in one byte. The least significant bit (LSB) contains the state of DI0, the most significant (MSB) contains the state of DI7.

The inputs always have the value “0” in the idle state. In the active state (voltage applied), the value changes to “1”. Refer to the PiXtend V2 -S- Hardware Manual for which level results in a “1” or “0”.

It is possible to debounce the digital inputs, which is useful for many applications. Further information is available in section 14.1.2.3.4 DigitalInDebounce.

12.1.1.6.2 AnalogInX (L/H)

AnalogInXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 28: AnalogInXL

AnalogInXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|-----|---|
| Name | - | - | - | - | - | - | MSB | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 29: Bits of the AnalogInXH bytes

The analog/digital converter integrated in the PiXtend controller sends 10-bit values. The values are organized in the process data in a 16-bit data word consisting of two bytes. The most significant two bits of the 10-bit value are in the byte AnalogInXH.

The “X” is replaced by the letter of the respective analog channel (“0” or “1”).

The analog inputs are already digitally filtered in the micro-controller. Ten values are recorded at a time, the average value is calculated and the result is stored in AnalogInX.

The value range is between “0” and “1023”. The analog inputs are marked AI0 and AI1 on the module and on the stainless-steel cover.

To calculate voltages from these values, the following formula is used:

Convert analog value to voltage

$\text{AnalogInX} * 10 / 1024 = \text{voltage at channel X [V]}$

(voltages are measured at AI0 and AI1)

Note the jumper for the voltage inputs. The “10” in the formula above is used if no jumper is set (0 - 10V range).

If the jumper is inserted for one channel (0 - 5V range), a “5” instead of a “10” must be used in the formula.

12.1.1.6.3 GPIOIn

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|-------|-------|-------|-------|
| Name | - | - | - | - | GPIO3 | GPIO2 | GPIO1 | GPIO0 |
| Start value | 0 | 0 | 0 | 0 | 0/1 | 0/1 | 0/1 | 0/1 |

Table 30: Bits of the GPIOIn bytes

The four GPIO inputs (GPIO_IN) are organized in one byte. The least significant bit (LSB) contains the status of GPIO0. Bit 7 (MSB) to bit 4 are not assigned.

The GPIO inputs have no defined idle state* (floating input). Only when a voltage is applied externally does a stable state result:

0V → value "0"

5V → value "1"

Since the inputs are floating, they can be at "0" or "1" without an external connection and can oscillate between the values.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information can be found in chapter Control and Status Bytes.

After a reset or power-up of Pixtend, the GPIOs are configured as inputs.

* It is possible to configure the GPIOs as inputs and to activate pull up resistors via the GPIOOut. This creates a "1" at the input in idle state (without this, they are floating inputs). Further information is available in section GPIOOut

12.1.1.6.4 TempX (L/H)

TempXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 31: TempXL byte

TempXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 32: TempXH byte

In the 1-Wire/DHT mode of the GPIOs, the bytes TempXL and TempXH contain a related 16-bit data word. This data word contains the temperature information of a sensor (if a sensor is connected and configured). The “X” stands for the number of the GPIO, i.e. “0” to “3”.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl.

The contained value can easily be converted into a temperature value (floating point value), but a distinction must be made between DHT11 and DHT22 (TempX represents the 16-bit value):

DHT11: $\text{TempX} / 256 = \text{floating-point value } [^{\circ}\text{C}]$

Example: $5632 / 256 = 22.0^{\circ}\text{C}$ - DHT11 sensors do not provide decimal places!

DHT22: $\text{TempX} / 10 = \text{floating-point value } [^{\circ}\text{C}]$

Example: $224 / 10 = 22.4^{\circ}\text{C}$

Depending on the programming language and data type used, a “typecast” must be carried out; otherwise, after dividing by 10 (DHT22) no floating-point value is obtained, but the integer value (in the example 22 instead of 22.4).

Dividing by 256 for the DHT11 sensors can be achieved with a “right shift” by 8 digits.

With DHT22 sensors, negative temperatures can also be recorded. To detect this, the MSB must be evaluated in TempXH. If the MSB is a “1”, then it is a negative temperature.

12.1.1.6.5 HumidX (L/H)

HumidXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 33: HumidXL byte

HumidXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 34: HumidXH byte

In the 1-Wire/DHT mode of the GPIOs, the bytes HumidXL and HumidXH contain a related 16-bit data word. This data word contains the humidity information of a sensor (if a sensor is connected and configured). The “X” stands for the number of the GPIO, i.e. “0” to “3”.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl.

The contained value can easily be converted into a humidity value (floating point value). However, a distinction must be made between DHT11 and DHT22 (HumidX represents the 16-bit value):

DHT11: $\text{HumidX} / 256 = \text{floating point value [\%]} - \text{relative humidity}$

Example: $10496 / 256 = 41.0\%$ - DHT11 sensors do not provide decimal places!

DHT22: $\text{HumidX} / 10 = \text{floating point value [\%]} - \text{relative humidity}$

Example: $417 / 10 = 41.7 \%$

Depending on the programming language and data type used, a “typecast” must be performed. If not, then after dividing by 10, no floating-point value results, but the integer value (in the example 41 instead of 41.7).

Dividing by 256 for the DHT11 sensors can be achieved with a “right shift” by 8 digits.

12.1.1.6.6 RetainDataInX

The Retain data consists of 32 bytes per direction (RetainDataOut and RetainDataIn). RetainDataIn is the previously backed up data that is transferred again from the PiXtend micro-controller to the Raspberry Pi after a power failure/power interruption. The data is available in RetainDataIn after the PiXtend system is restarted, provided that the Retain memory has been activated beforehand.

The data is retained until new retain data is stored. The data can be retained over any number of power cycles of the system if the Retain functionality is not activated again.

If the Retain memory is activated again and the power supply is interrupted, the data is overwritten by the current data in RetainDataOut.

12.1.2. Control and status bytes

PiXtend or the PiXtend micro-controller can be configured via control bytes. Furthermore, the micro-controller informs the Raspberry Pi about status bytes, about the status and about errors and warnings.

The following pages give an overview of the control and status bytes, then each byte is described in detail. We show different possibilities and perform example calculations.

If you still have questions, please contact us by e-mail (support@pixtend.de) and you will receive an answer and further information as soon as possible.

12.1.2.1 Overview of the control bytes

Control bytes are transferred from the Raspberry Pi to the PiXtend micro-controller. It can be used to control the behavior of the micro-controller.

The following control bytes are available:

- ModelOut
This indicates which hardware version (model) that the RPi software (e.g. CODESYS or pxdev) expects
- UCMODE
This specifies which transfer mode is to be used
- UCCTRL
This is basic settings of the micro-controller (Retain, Watchdog, Safe State...)
- DigitalInDebounce
This indicates whether and how the digital inputs are to be debounced
- GPIOCTRL
This configures the PiXtend GPIOs (input, output, DHT mode)
- GPIODEBOUNCE
This indicates whether and how the GPIO inputs should be debounced
- PWM0CTRL and PWM1CTRL
This configures the PWM channels (mode, enable, frequencies)

12.1.2.2 Overview of the status bytes

Status bytes are transferred from the PiXtend micro-controller to the Raspberry Pi. They contain important information about the status of PiXtend and the micro-controller.

The following status bytes are available:

- Firmware
This informs about the micro-controller firmware version
- Hardware
This informs about the PiXtend hardware version
- ModelIn
This informs about the PiXtend model
- UCState
This contains the operating state and error codes of the micro-controller
- UCWarnings
This contains the warnings of the micro-controller

12.1.2.3 Description of the control bytes

12.1.2.3.1 ModelOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | x | x | x | x | x | x | x | x |

Table 35: ModelOut byte

By means of the byte ModelOut, the application software running on the Raspberry Pi informs the micro-controller which PiXtend model it expects. The micro-controller decides if it is the expected model and if not, it can return an error to the Raspberry Pi. The content of ModelOut is an ASCII character. With the PiXtend V2 -S-, the byte contains the following value: ASCII character **S** (capital letter S - 83 decimal / 53 hexadecimal).

There is usually no reason for the user to change this value. As an example, the CODESYS package “PiXtend V2 Professional for CODESYS” already includes drivers for which the model is permanently programmed and cannot be changed.

12.1.2.3.2 UCMODE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 36: Bits of the UCMODE bytes

Currently (as of October 2017), there is only one transmission mode, “Auto Mode”. The byte is reserved for future use.

For the operation of PiXtend, it is not necessary to enter a value here. If a value is entered, this has no effect.

12.1.2.3.3 UC Ctrl

12.1.2.3.3.1 UC Ctrl0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|------|------|------|------|
| Name | - | - | - | - | WDE3 | WDE2 | WDE1 | WDE0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 37: Bits of the UC Ctrl0 bytes

Bit 0 - 3 – WDE0 - 3 (WatchdogEnable0 - 3)

With the WDE0 - 3 bits, the watchdog timer of the PiXtend micro-controller can be activated and set. All bits are “0” in the standard setting. The activation and setting of the watchdog is explained in the following table:

| WDE3 | WDE2 | WDE1 | WDE0 | Status Watchdog | Configured time |
|------|------|------|------|-----------------|-----------------|
| 0 | 0 | 0 | 0 | deactivated | deactivated |
| 0 | 0 | 0 | 1 | active | 16 ms |
| 0 | 0 | 1 | 0 | active | 32 ms |
| 0 | 0 | 1 | 1 | active | 64 ms |
| 0 | 1 | 0 | 0 | active | 0.125 s |
| 0 | 1 | 0 | 1 | active | 0.25 s |
| 0 | 1 | 1 | 0 | active | 0.5 s |
| 0 | 1 | 1 | 1 | active | 1 s |
| 1 | 0 | 0 | 0 | active | 2 s |
| 1 | 0 | 0 | 1 | active | 4 s |
| 1 | 0 | 1 | 0 | active | 8 s |

Table 38: Watchdog setting

If the watchdog is activated, the communication between the Raspberry Pi and PiXtend is monitored. If there is a pause between two valid cycles which is longer than the set time, the watchdog becomes active and puts the micro-controller into a safe state*. An invalid cycle (for example due to a CRC error) is evaluated by the watchdog as if no cycle had been performed.

The use of the watchdog is useful if the PiXtend controller is to be set to a defined state in the event of an error. An error can occur if the application program on the Raspberry Pi does not react and no data is transferred to the micro-controller.

Recommendation

Do not activate the watchdog during the development of your application software. If a new program is transferred via CODESYS or the user software is debugged, no transfer takes place for a certain time. The watchdog would become active, and a restart of the system would be necessary. This leads to an unnecessary delay of your development process.

* Definition "Safe State":

- All digital outputs and relays are switched off/ put into the idle state
- PWM outputs are switched to high impedance (tri-state)
- Retain data is stored when Retain option has been activated
- The status LED "L1" flashes depending on the cause of the error (Error LED "L1" Signals) when the LED is activated
- The micro-controller or the PiXtend device has to be restarted (power cycle)

If switching off the digital outputs does not correspond to the safe state of your application, external measures must be taken to prevent any critical or dangerous situations.

12.1.2.3.3.2. UCCTRL1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|------|------|----|----|------|
| Name | - | - | - | GPUE | SLED | RE | RC | SAFE |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 39: Bits of the UCCTRL1 bytes

Bit 0 – SAFE (SafeState)

The SAFE bit, when activated by the value “1”, can immediately put the micro-controller into the safe state*.

The bit has the value “0” by default.

* Definition “Safe State”:

- All digital outputs and relays are switched off/ put into the idle state
- PWM outputs are switched to high impedance (tri-state)
- Retain data is stored when Retain option has been activated
- The status LED “L1” flashes depending on the cause of the error when the LED is activated
- The micro-controller or the PiXtend device has to be restarted (power cycle)

If switching off the digital outputs does not correspond to the safe state of your application, external measures must be taken to prevent any critical or dangerous situations.

Bit 1 – RC (RetainCopy)

The RC bit can be used to configure which data is visible in the Retain Input area (RetainDataIn0 - 31).

At the start value “0”, the last stored data is transferred from the micro-controller to the Raspberry Pi (normal Retain operation).

If the value “1” is set for the RC bit, the last data sent by the Raspberry Pi is returned. In the Retain area RetainDataIn0 - 31 (RetainDataOut0 - 31) is mirrored, but with a cycle delay (Retain Copy mode). The content of the Retain data is not lost, it is just not displayed if the RC bit is “1”.

Bit 2 – RE (RetainEnable)

With the RE bit, the retain function of the PiXtend can be activated. For this purpose, a “1” is written into this bit. By default, RE is “0” (after a reset or power-up) and therefore Retain is deactivated.

Recommendation

Only activate Retain if the functionality is really needed. The number of retain memory can only be guaranteed up to 10,000 cycles.

For more information on the PiXtend V2 -S- Retain function refer to chapter: 6 Basic knowledge.

Bit 3 – SLED (StatusLED)

The status LED “L1” can be deactivated via the SLED bit, if it is not needed. The status LED is active by default and can be deactivated by the value “1” in SLED.

Bit 4 – GPUE (GPIOPullUpEnable)

The GPUE bit can be used to enable the pull-up resistors of the PiXtend GPIOs (these are not yet activated by this process). Enabling it has an effect if the GPIOs are configured as inputs and a “1” is written in the byte GPIOOut for the respective GPIO.

A “1” in GPUE activates this functionality.

12.1.2.3.4 DigitalInDebounce

DigitalInDebounce01

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 40: Bits of the DigitalInDebounce01 bytes

DigitalInDebounce23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 41: Bits of the DigitalInDebounce23 bytes

DigitalInDebounce45

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 42: Bits of the DigitalInDebounce45 bytes

DigitalInDebounce67

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 43: Bits of the DigitalInDebounce67 bytes

With the help of the DigitalInDebounce bytes, the debouncing of the eight digital inputs of PiXtend can be configured. By default, the debouncing of the inputs is not active.

Debouncing is activated/deactivated for two channels together. There are no interactions between the two channels, they only receive the same debounce setting.

The number in a DigitalInDebounce byte corresponds to the number of cycles in which an electrical signal applied to the digital input must remain constant in order to pass on the level change to the application software (Raspberry Pi). This applies to a change of the signal level from "0" (low) to "1" (high) and vice versa. If the bytes are not changed or a "0" is written as a value, no debouncing takes place and the application software receives a new value in each cycle.

An example

The first two digital inputs (DI0 and DI1) should be debounced. Only changes to the digital inputs that are constant for longer than 100 ms should be visible. The cycle time (data exchange between PiXtend and Raspberry Pi) is 10 ms.

- The byte DigitalInDebounce01 is used
- Calculation of the value for DigitalInDebounce01: $100 \text{ ms} / 10 \text{ ms} = 10$

The byte is written with a 10 (decimal) to get the desired effect. If the cycle time is changed during software development, the value for debouncing must be adjusted.

12.1.2.3.5 GPIOCtrl

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|-------|-------|-------|-----|-----|-----|-----|
| Name | SENS3 | SENS2 | SENS1 | SENS0 | IO3 | IO2 | IO1 | IO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 44: Bits of the GPIOCtrl bytes

Bits 0 - 3 – IO0 - 3

With the IO bits, the four PiXtend GPIOs can be configured as digital input or output. All GPIOs are configured as inputs with the start value "0". If the IO3 bit is set to "1", GPIO3 becomes an output. The value is set via the data word GPIOOut.

Bits 4 - 7 – SENS0 - 3 (Sensor0 - 3)

The GPIOs can be used as input/output and for 1-Wire sensors (DHT11, DHT22, AM2302). The upper four bits of the GPIOCtrl data word are available for this purpose. If the SENS3 bit is set to "1" here, a sensor can be connected to GPIO3 and evaluated. The setting of the IO bit is then no longer important. The SENSX bits prevail over the IOX bits.

The results/measured values of the sensors are in TempXL/TempXH and HumidXL/HumidXH. The "X" corresponds to the number of the GPIO to which a sensor is connected.

NOTICE

Communication with the sensors requires time (ca. 25 ms), using at least one sensor results in a minimum cycle time of 30 ms. It does not matter whether one or four sensors are queried.

12.1.2.3.6 GPIODebounce

GPIODebounce01

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 45: Bits of the GPIODebounce01 bytes

GPIODebounce23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 46: Bits of the GPIODebounce23 bytes

Debouncing the GPIOs is only possible if they are configured as inputs. The GPIODebounce bytes are used like the DigitalInDebounce bytes. More information and examples are in the section:

DigitalInDebounce

12.1.2.3.7 PWM0Ctrl – for 16-bit PWMs

The following descriptions refer to the PWM0 channels (PWM0A and PWM0B) with 16-bit resolution. The information about the 8-bit PWM channels may be found in the chapter PWM1Ctrl - for 8-bit PWMs.

12.1.2.3.7.1. PWM0Ctrl0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|---------|---------|---|-------|-------|
| Name | PS2 | PS1 | PS0 | EnableB | EnableA | - | MODE1 | MODE0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 47: Bits of the PWM0Ctrl0 bytes

Bits 0 - 1 – MODE0 - 1

The PWM outputs are operated in four different modes. The mode is configured via the MODE bits. PiXtend starts in servo mode (MODE0 - 1 = 00) by default.

| MODE1 | MODE0 | Mode | Description |
|-------|-------|-----------------|---|
| 0 | 0 | Servo mode | Both channels (A/B) are used to control model construction servos - special signal form |
| 0 | 1 | Duty cycle mode | Both channels (A/B) have the same frequency, but an independently adjustable duty cycle |
| 1 | 0 | Universal mode | Free adjustable frequency and duty cycle for channel A, half frequency for channel B and 50% duty cycle |
| 1 | 1 | Frequency mode | Freely adjustable frequencies for both channels (A/B), the duty cycle is always 50%. |

Table 48: Bits of the PWM0Ctrl0 byte

The Servo mode and duty cycle mode are sufficient, the universal and frequency mode can be useful and helpful for special applications but are more complex to configure and use.

Bit 3 - 4 – EnableA, EnableB

With the bits EnableA and EnableB, the respective channel is activated. By default, the PWM channels are deactivated. By writing a "1" into the respective bit, the channel is activated and the PWM becomes visible at the output.

The PWM can first be configured for the desired request (mode, frequency/value ...) and only then be activated. The activation is possible in the same cycle.

Bit 5 - 7 – PS0 - PS2 (Prescaler0 - 3)

The prescaler bits (PS bits) enable the approximate setting of the PWM frequency. Depending on the PWM mode (see MODE bits), the PS bits have different effects.

The following table shows the effects of the PS bits:

| PS2 | PS1 | PS0 | Description | Base frequency |
|-----|-----|-----|-------------------------|----------------|
| 0 | 0 | 0 | PWM outputs deactivated | - |
| 0 | 0 | 1 | Prescaler: 1 | 16 MHz |
| 0 | 1 | 0 | Prescaler: 8 | 2 MHz |
| 0 | 1 | 1 | Prescaler: 64 | 250 kHz |
| 1 | 0 | 0 | Prescaler: 256 | 62.5 kHz |
| 1 | 0 | 1 | Prescaler: 1024 | 15.625 kHz |
| 1 | 1 | 0 | Prescaler: 1024 | 15.625 kHz |
| 1 | 1 | 1 | Prescaler: 1024 | 15.625 kHz |

Table 49: Prescaler bits of the PWM0Ctrl0 byte

The prescaler settings have different effects depending on the selected mode:

| Mode | Description |
|-----------------|--|
| Servo mode | No effect |
| Duty cycle mode | Setting of the prescaler or the base frequency |
| Universal mode | Setting of the prescaler or the base frequency |
| Frequency mode | Setting of the prescaler or the base frequency |

Table 50: PWM mode

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWM0Ctrl0, the fine adjustment via PWM0Ctrl1L/H.

See section 16-bit Resolution for the exact frequency/period duration setting.

NOTICE

If the frequency mode is to be used, no DHT sensors must be connected and activated in the GPIOCtrl byte. A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.1.2.3.7.2. PWM0Ctrl1 (L/H)

The PWM0Ctrl1 bytes L and H contain a related 16-bit data word. PWM0Ctrl1L is the low byte and PWM0Ctrl1H the high byte.

With the two bytes, the PWM frequency/period duration can be set (in PWM mode “duty cycle” and “universal”). The frequency is valid for both channels. In the duty cycle mode, the duty cycle can be configured independently for each channel. In the universal mode, the duty cycle can only be set for channel A.

In the frequency mode, the PWM1Ctrl1 bytes have no effect.

PWM0Ctrl1L

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 51: PWM0Ctrl1L byte

PWM0Ctrl1H

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 52: PWM0Ctrl1H byte

Example for the PWM period duration in “duty cycle mode”:

The prescaler (PS0 - 2) is set to the value 64 and thus the basic frequency to 250 kHz. In addition, the duty cycle mode and channel A are activated:

PWM0Ctrl0: 01101001b

The 16-bit data word PWM0Ctrl1 is set to the value 1000:

PWM0Ctrl1L: 11101000b

PWM0Ctrl1H: 00000011b

PWM period duration = micro-controller basic clock / 2 / prescaler / PWM0Ctrl1

PWM period duration = 16 MHz / 2 / 64 / 1000 = 125 Hz

12.1.2.3.8 PWM1Ctrl – for 8-bit PWMs

The following descriptions refer to the PWM1 channels (PWM1A and PWM1B) with 8-bit resolution. The information about the 16-bit PWM channels may be found in the chapter 12.1.2.3.7 PWMOCtrl- for 16-bit PWMs.

12.1.2.3.8.1. PWM1Ctrl0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|---------|---------|---|-------|-------|
| Name | PS2 | PS1 | PS0 | EnableB | EnableA | - | MODE1 | MODE0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bits 0 - 1 – MODE0 - 1

The PWM outputs are operated in four different modes. The mode is configured via the MODE bits, by default PiXtend starts in servo mode (MODE0 - 1 = 00).

| MODE1 | MODE0 | Mode | Description |
|-------|-------|-----------------|---|
| 0 | 0 | Servo mode | Both channels (A/B) are used to control model construction servos - special signal form |
| 0 | 1 | Duty cycle mode | Both channels (A/B) have the same frequency, but an independently adjustable duty cycle |
| 1 | 0 | Universal mode | Free adjustable frequency and duty cycle for channel A, half frequency for channel B and 50% duty cycle |
| 1 | 1 | Frequency mode | Freely adjustable frequencies for both channels (A/B), the duty cycle is always 50%. |

The Servo mode and duty cycle mode are sufficient, the universal and frequency mode can be useful and helpful for special applications but are more complex to configure and use.

Bit 3 - 4 – EnableA, EnableB

With the bits EnableA and EnableB, the respective channel is activated. By default, the PWM channels are deactivated. By writing a "1" into the respective bit, the channel is activated and the PWM becomes visible at the output.

The PWM can first be configured for the desired request (mode, frequency/value ...) and only then be activated. The activation is possible in the same cycle.

Bit 5 - 7 – PS0 - PS2 (Prescaler0 - 3)

The prescaler bits (PS bits) enable the approximate setting of the PWM frequency. Depending on the PWM mode (see MODE bits), the PS bits have different effects.

The following table shows the effects of the PS bits:

| CS2 | CS1 | CS0 | Description | Base frequency |
|-----|-----|-----|-------------------------|----------------|
| 0 | 0 | 0 | PWM outputs deactivated | - |
| 0 | 0 | 1 | Prescaler: 1 | 16 MHz |
| 0 | 1 | 0 | Prescaler: 8 | 2 MHz |
| 0 | 1 | 1 | Prescaler: 32 | 0.5 MHz |
| 1 | 0 | 0 | Prescaler: 64 | 250 kHz |
| 1 | 0 | 1 | Prescaler: 128 | 125 kHz |
| 1 | 1 | 0 | Prescaler: 256 | 62.5 kHz |
| 1 | 1 | 1 | Prescaler: 1024 | 15.625 kHz |

The prescaler settings have different effects depending on the selected mode:

| Mode | Description |
|-----------------|--|
| Servo mode | No effect |
| Duty cycle mode | Setting of the prescaler or the base frequency |
| Universal mode | Setting of the prescaler or the base frequency |
| Frequency mode | Setting of the prescaler or the base frequency |

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWM1Ctrl0, the fine adjustment via PWM1Ctrl1L/H.

Refer to the section 12.1.1.4.5 PWM1X (L/H) for 8-bit resolution for the exact frequency/period duration setting.

NOTICE

A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.1.2.3.8.2. PWM1Ctrl1 (L/H)

The PWM1 channels (8-bit PWM) have two PWM1Ctrl bytes (L and H) but only the low byte (L) is used.

With PWM1Ctrl1L, the PWM frequency/period duration can be set (in PWM mode “duty cycle” and “universal”). The frequency is valid for both PWM1 channels (A and B).

In duty cycle mode, the duty cycle for each channel can be configured independently. In the universal mode, the duty cycle can only be set for channel A. In the frequency mode, the PWM1Ctrl1 bytes have no effect.

PWM1Ctrl1L

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 53: Bits of the PWM1Ctrl1L bytes

PWM1Ctrl1H – not used

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 54: Bits of the PWM1Ctrl1H bytes

Example for the PWM period duration in the “universal mode”:

The prescaler (PS0 - 2) is set to the value 64 and thus the basic frequency to 250 kHz, in addition the universal mode and channel A are activated:

PWM1Ctrl0: 10001010b

The 8-bit data byte PWM1Ctrl1L is set to the value 150:

PWM1Ctrl1L: 10010110b

PWM1Ctrl1H: not used

PWM period duration = micro-controller quartz / 2 / prescaler / PWM0Ctrl1

PWM period duration = 16 MHz / 2 / 64 / 150 = 0.833 kHz

Example for the PWM period duration in “duty cycle mode”:

The prescaler (PS0 - 2) is set to the value 64 and thus the basic frequency to 250 kHz, in addition the duty cycle mode and channel A are activated:

PWM1Ctrl0: 10001001b

Here a special feature of the 8-bit PWMs in duty cycle mode must be considered: there are two different setting options for PWM1Ctrl1L (“0” or “1”)

PWM1Ctrl1L: 00000000b

PWM1Ctrl1H: not used

PWM period duration = micro-controller basic clock / Prescaler / 255 / 2

PWM period duration = 16 MHz / 64 / 255 / 2 = 490 Hz

PWM1Ctrl1L: 00000001b

PWM1Ctrl1H: not used

PWM period duration = micro-controller basic clock / prescaler / 255

PWM period duration = 16 MHz / 64 / 255 = 980 Hz

12.1.2.4 Description of the status bytes

The status bytes inform the user and the application program running on the Raspberry Pi about the status of the micro-controller.

The status bytes can be queried via CODESYS, PiXtend-C-Library or PiXtend-Python-Library (PPL).

12.1.2.4.1 Firmware

The firmware byte contains a number that stands for the firmware version. The number can be between 0 and 255. If you request support for your PiXtend V2 device, please always tell us the firmware version.

12.1.2.4.2 Hardware

The hardware byte contains the version number of your PiXtend board. With the version number, the user software can check if software and drivers fit to the existing hardware.

Example:

Hardware byte contains the number 21

→ PiXtend hardware version 2.1

12.1.2.4.3 ModelIn

In the ModelIn byten the micro-controller informs the user software about the PiXtend V2 model. With the model identifier, the user software can check whether the software and drivers match the existing hardware.

The content of ModelIn is an ASCII character. With the PiXtend V2 -S-, the byte contains the following value: ASCII character **S** (capital letter S - 83 decimal / 53 hexadecimal).

12.1.2.4.4 UCState

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|------|------|------|------|---|---|---|-----|
| Name | ERR3 | ERR2 | ERR1 | ERR0 | - | - | - | RUN |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 55: Bits of the UCState bytes

Bit 0 – RUN

By querying this bit, the user can check whether the PiXtend is ready for operation ("1") or whether a successful data transmission has been performed. This can be useful after starting the PiXtend system.

If the controller is set to the safe state (for example by the watchdog), the bit contains a "0". This can no longer be queried since no communication is possible until a restart has been performed.

Bits 4 - 7 – ERROR0 - 3

The ERR bits provide information about problems that the micro-controller detects and passes on to the user program.

The following table describes the error codes and their causes:

| ERR3 | ERR2 | ERR1 | ERR0 | Error description |
|------|------|------|------|--|
| 0 | 0 | 0 | 0 | No error |
| 0 | 0 | 0 | 1 | - |
| 0 | 0 | 1 | 0 | CRC error - data the payload data received from the Raspberry Pi is incorrect |
| 0 | 0 | 1 | 1 | Data block too short Driver faulty/incorrect model defined |
| 0 | 1 | 0 | 0 | PiXtend model does not match the expected and the actual model do not match (ModelIn and ModelOut are unequal) |
| 0 | 1 | 0 | 1 | CRC error - Header the header data received from the Raspberry Pi is incorrect |
| 0 | 1 | 1 | 0 | SPI frequency too high the set SPI frequency is too high, transmission errors occur |

Table 56: Breakdown of the error bits in the UCState byte

12.1.2.4.5 UCWarnings

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|----|----|---|
| Name | - | - | - | - | - | W1 | W0 | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 57: Bits of the UCWarnings bytes

Bits 1 - 2 – W0 - 1 (RetainCRCErrror. RetainVoltageError)

The WX bits inform the user about warnings regarding the Retain functionality of the PiXtend V2. These bits can be checked in the user program. Thus, it can be recognized whether the Retain memory is working correctly or not.

W0 – RetainCRCErrror

This is “1” if an error was detected when checking the retain memory.

W1 – RetainVoltageError

This is “1” if the current supply voltage of the PiXtend is too low (less than 19V). Retain memory cannot be activated.

12.2. PiXtend V2 -L-

12.2.1. Process data

12.2.1.1 Overview of the process data

There are two types of process data:

- Process data transmitted from the Raspberry Pi to the PiXtend
- Process data transferred from the PiXtend to the Raspberry Pi

12.2.1.1.1 Process data transmitted from the Raspberry Pi to the PiXtend

Data for digital outputs, relays and GPIOs (as outputs) PWM data, Retain data

- DigitalOutX
- RelayOut
- GPIOOut
- PWMYX (L/H)
- RetainDataOutX

12.2.1.1.2 Process data transmitted from the Raspberry Pi to the PiXtend DAC

Date for analog outputs

- AnalogOutX (L/H)

12.2.1.1.3 Process data transferred from the PiXtend to the Raspberry Pi

Data of the digital and analog inputs, GPIOs (as inputs)

Temperature and humidity of DHT11/22 or AM2302 sensors, Retain data

- DigitalInX
- AnalogInX (L/H)
- GPIOIn
- TempX (L/H)
- HumidX (L/H)
- RetainDataInX

12.2.1.2 SPI bus protocol overview

| INPUT | | | | OUTPUT | | |
|-------------------------|--------------|-----------|----------|-------------------------|-----------------------|-----------|
| MC (Out) --> RasPi (In) | | | | RasPi (Out) --> MC (In) | | |
| | Name | used Bits | Byte Idx | | Name | used Bits |
| HeaderIn | Firmware | 8 | 0 | HeaderOut | ModelOut | 8 |
| | Hardware | 8 | 1 | | UCMode | 8 |
| | ModelIn | 8 | 2 | | UCCtrl0 | 8 |
| | UCState | 8 | 3 | | UCCtrl1 | 8 |
| | UCWarnings | 8 | 4 | | Reserved | 0 |
| | Reserved | 0 | 5 | | Reserved | 0 |
| | Reserved | 0 | 6 | | Reserved | 0 |
| | CRCHeaderInL | 8 | 7 | | CRCHeaderOutL | 8 |
| | CRCHeaderInH | 8 | 8 | | CRCHeaderOutH | 8 |
| DataIn | DigitalIn0 | 8 | 9 | DataOut | DigitalInDebounce01 | 8 |
| | DigitalIn1 | 8 | 10 | | DigitalInDebounce23 | 8 |
| | AnalogIn0L | 8 | 11 | | DigitalInDebounce45 | 8 |
| | AnalogIn0H | 2 | 12 | | DigitalInDebounce67 | 8 |
| | AnalogIn1L | 8 | 13 | | DigitalInDebounce89 | 8 |
| | AnalogIn1H | 2 | 14 | | DigitalInDebounce1011 | 8 |
| | AnalogIn2L | 8 | 15 | | DigitalInDebounce1213 | 8 |
| | AnalogIn2H | 2 | 16 | | DigitalInDebounce1415 | 8 |
| | AnalogIn3L | 8 | 17 | | DigitalOut0 | 8 |
| | AnalogIn3H | 2 | 18 | | DigitalOut1 | 4 |
| | AnalogIn4L | 8 | 19 | | RelayOut | 4 |
| | AnalogIn4H | 2 | 20 | | GPIOCtrl | 8 |
| | AnalogIn5L | 8 | 21 | | GPIOOut | 4 |
| | AnalogIn5H | 2 | 22 | | GPIODebounce01 | 8 |
| | GPIOIn | 4 | 23 | | GPIODebounce23 | 8 |
| | Temp0L | 8 | 24 | | PWM0Ctrl0 | 8 |
| | Temp0H | 8 | 25 | | PWM0Ctrl1L | 8 |
| | Humid0L | 8 | 26 | | PWM0Ctrl1H | 8 |
| | Humid0H | 8 | 27 | | PWM0AL | 8 |
| | Temp1L | 8 | 28 | | PWM0AH | 8 |
| | Temp1H | 8 | 29 | | PWM0BL | 8 |
| | Humid1L | 8 | 30 | | PWM0BH | 8 |
| | Humid1H | 8 | 31 | | PWM1Ctrl0 | 8 |
| | Temp2L | 8 | 32 | | PWM1Ctrl1L | 8 |
| | Temp2H | 8 | 33 | | PWM1Ctrl1H | 8 |

| | | | | |
|----------------------|-----|-----|-----------------------|-----|
| Humid2L | 8 | 34 | PWM1AL | 8 |
| Humid2H | 8 | 35 | PWM1AH | 8 |
| Temp3L | 8 | 36 | PWM1BL | 8 |
| Temp3H | 8 | 37 | PWM1BH | 8 |
| Humid3L | 8 | 38 | PWM2Ctrl0 | 8 |
| Humid3H | 8 | 39 | PWM2Ctrl1L | 8 |
| Reserved | 0 | 40 | PWM2Ctrl1H | 8 |
| Reserved | 0 | 41 | PWM2AL | 8 |
| Reserved | 0 | 42 | PWM2AH | 8 |
| Reserved | 0 | 43 | PWM2BL | 8 |
| Reserved | 0 | 44 | PWM2BH | 8 |
| RetainDataIn0 ... 63 | all | 45 | RetainDataOut0 ... 63 | all |
| CRCDataInL | 8 | 109 | CRCDataOutL | 8 |
| CRCDataInH | 8 | 110 | CRCDataOutH | 8 |

Table 58: PiXtend V2 -L- SPI protocol overview

Every byte of the SPI protocol is described in this chapter, partly with application notes and calculation examples, see, for example, PWMs.

The SPI protocol for the PiXtend V2 -L- includes a total of 111 bytes, table 60 shows a shortened form, here the 64 bytes of the Retain memory were summarized. If an entry is a single byte, the SPI protocol only contains the name or designation of the byte.

With 16-bit values (2 bytes), for example temperature, the value is divided into two individual bytes. This can be recognized by the addition's "L" for the low byte and "H" for the high byte. If you want to implement the SPI protocol, make sure that the bytes for High and Low are assigned accordingly.

The CRC calculation for the header and data is not explicitly listed. Please take this information from the source code for our Linux tools (pxdev), to be found on SourceForge, our website in the download section or our Python modules. There is a corresponding implementation that you can use it as a template for your programming language.

12.2.1.3 SPI bus configuration

To address PiXtend V2 SPI devices from your own applications, the following SPI bus settings have to be considered:

- SPI-Mode 0 (CPOL = 0, CPHA = 0)
- SPI speed: 700 kHz
- SPI ChipSelect 0 (CS0): PiXtend V2 Micro-controller
- SPI ChipSelect 1 (CS1): PiXtend DAC or CAN interface (for V2 -L-)
- GPIO24, wiringPi BCM numbering (wiringPi's own numbering GPIO 5) of the Raspberry Pi must be set to "high" / logical "1" → SPI Enable.

Cycle time limit:

- Minimum cycle time V2 -S-: 2.5 ms
- Minimum cycle time V2 -L-: 5 ms

This is the minimum time interval between two communication processes with the micro-controller. Longer cycle times are better.

We recommend reading over the source code of the PiXtend library and the Python modules to get a better feeling for using the SPI bus.

12.2.1.4 Output data

Output data is the data that the Raspberry Pi transfers to the PiXtend micro-controller, for example, digital outputs or relays.

12.2.1.4.1 DigitalOutX

The digital outputs are organized in two bytes. The byte DigitalOut0 contains the first eight outputs (DO0 - DO7). The byte DigitalOut1 contains the remaining four outputs (DO8 - DO11).

DigitalOut0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | DO7 | DO6 | DO5 | DO4 | DO3 | DO2 | DO1 | DO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 59: Bits of the DigitalOut0 bytes

DigitalOut0 - Bit 0 - 7

The least significant bit (LSB) contains the status of DO0. The most significant bit (MSB) contains the status of DO7.

DigitalOut1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|------|------|-----|-----|
| Name | - | - | - | - | DO11 | DO10 | DO9 | DO8 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 60: Bits of the DigitalOut1 bytes

DigitalOut1 - Bit 0 - 3

The least significant bit (LSB) contains the status of DO8. Bit 7 (MSB), bit 6, bit 5 and bit 4 are not assigned.

The start value after the power-up or reset of the micro-controller is “0” for the DigitalOutX byte. The outputs are deactivated at the Start and in SafeState mode. By writing a “1” to one of the DO bits, the corresponding output is activated. The corresponding LED on the PiXtend board lights up.

The digital outputs on the PiXtend V2 have a separate power supply for each of the four outputs:

DO0 – DO3: VCC-DO0-3
DO4 – DO7: VCC-DO4-7
DO8 – DO11: VCC-DO8-11

The power supply connections are located on the terminal block, which also houses the four outputs. The LED “VCC-DOx-x” indicates whether a supply voltage is connected to the power supply.

The LEDs of the digital outputs also light up if the VCC-DO supply has not been connected. A voltage at the DO pins “does not become visible”. For more information, refer to the hardware manual of the PiXtend V2 -L-.

To test the function of the outputs, the value 255 (decimal) or 0xFF (hex) can be written into the DigitalOutX bytes. All 12 outputs are activated. The fact that bits 4 to 7 of DigitalOut1 are also set here has no effect.

12.2.1.4.2 RelayOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|--------|--------|--------|--------|
| Name | - | - | - | - | RELAY3 | RELAY2 | RELAY1 | RELAY0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 61: Bits of the RelayOut bytes

Bit 0 - 3

The four relay outputs are organized in one byte. The least significant bit (LSB) contains the status of RELAY0. Bit 7 (MSB) to bit 4 are not assigned. The start value after the power-up or reset of the micro-controller is “0” for the entire RelayOut byte. The relays are thus deactivated at the Start and in SafeState mode.

By writing a “1” to one of the bits (0 - 3), the corresponding relay is activated. The corresponding LED on the PiXtend board lights up and the relay switches can be heard.

To test the function of the relays, the value 255 (decimal) or 0xFF (hex) can be written on the RelayOut byte. All four relays are activated. The fact that bits 4 to 7 are also set here has no effect.

12.2.1.4.3 GPIOOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|-------|-------|-------|-------|
| Name | - | - | - | - | GPIO3 | GPIO2 | GPIO1 | GPIO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 62: Bits of the GPIOOut bytes

Bit 0 - 3

The GPIO outputs are organized in one byte. The least significant bit (LSB) contains the status of GPIO0. Bit 7 (MSB) to bit 4 are not assigned. The start value after the power-up or reset of the micro-controller is "0" for the entire GPIOOut byte. The GPIOs are thus deactivated at the Start and in SafeState mode.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information is available in the chapter 12.2.2 Control and Status Bytes.

By writing a "1" into one of the bits (0 - 3), the corresponding GPIO is activated (if configured as output). If the GPIO(s) are configured as input or for the mentioned sensors, changing the values in GPIOOut has no effect*.

To test the function of the GPIOs, they can be configured as outputs. Then the value 255 (decimal) or 0xFF (hex) can be written for the GPIOOut byte. All four GPIO outputs are activated. The fact that bits 4 to 7 are also set here has no effect.

* If a GPIO is configured as an input with pull-ups and the corresponding bit is set to "1" in GPIOOut, a pull-up resistor is activated at this GPIO. This gives the GPIO input a high-impedance (~35 kOhm) reference to 5V and it no longer floats. Without external circuitry, the input remains at a high level. By default, the GPIO inputs are floating inputs without a defined level (without external circuitry). The pull up resistors are generally activated by the bit "GIOPullUpEnable".

12.2.1.4.4 PWMYX – 16-bit resolution

The PWM units of the PiXtend can be operated in four different modes. In each mode, the PWM process data has a different effect. For this reason, we will discuss the four modes "Servo Mode", "Duty Cycle Mode", "Universal Mode" and "Frequency Mode" separately in this section.

More information about configuration and modes of the PWM outputs may be found in the section: 12.2.2.3.7 PWMYCtrl – for 16-bit PWMs.

The PWM bytes described below are the process data of the six 16-bit PWM channels that have the full designation PWM0A (P0A), PWM0B (P0B), PWM1A (P1A), PWM1B (P1B), PWM2A (P2A) and PWM2B (P2B) on the module.

We speak of a group when two channels have the same number, for example, PWM0A and PWM0B form the first PWM group. If a group is put into a mode, both channels A and B are in that mode. A single channel of a group cannot be set to another mode, always only the whole group.

12.2.1.4.4.1. Servo mode

The servo mode is specially designed for the requirements of model construction servos. The period duration is fixed at 50 Hz (20 ms). At the beginning, the signal is at least 1 ms (minimum position) or maximum 2 ms (maximum position) at the high level. The remaining time is always the signal at the low level.

The PWMYX bytes L and H contain a related 16-bit data word. PWMYXL is the low byte and PWMYXH the high byte. The “Y” is replaced by the number of the group (“0”, “1” or “2”). The “X” is replaced by the letter of the respective PWM channel (“A” or “B”). Thus all six PWM outputs can be mapped according to the same principle.

PWMYXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 63: PWMYXL byte

PWMYXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 64: PWMYXH byte

Adjustable value range

Smallest value: 0 (decimal) → 1 ms, minimum position

Maximum value: 16000 (decimal) → 2 ms, maximum position

Values greater than 16000 are limited by the controller and act as 16000. In the range between 0 and 16000, the movement of the servo is linear. The start value of the PWMYX byte, after a power-up/reset, is “0”.

12.2.1.4.4.2. Duty cycle mode

In duty cycle mode, the bytes PWM0XL and PWM0XH contain a related 16-bit data word. The data word is used to set the duty cycle.

The “Y” is replaced by the number of the group (“0”, “1” or “2”). The “X” is replaced by the letter of the respective PWM channel (“A” or “B”). A separate duty cycle can be assigned to each channel. The frequency and period duration are set together for both channels of a group.

This mode is ideal for controlling several drives or fans from 0% to 100% with the same frequency, independent of their speed.

PWMYXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 65: PWMYXL byte

PWMYXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 66: PWMYXH byte

Which duty cycle results from a certain value depends on the PWMYXL/H values and on the control bytes PWMYCtrl1L and PWMYCtrl1H. The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWMYCtrl0, the fine adjustment via PWMYCtrl1L/H.

More information about the confirmation of the PWMYX channels may be found in the section: 12.2.2.3.7 PWMYCtrl – for 16-bit PWMs.

Calculation formula - PWM –16 bit

Frequency: $f = 16 \text{ MHz} / 2 / \text{Prescaler} / \text{PWMYCtrl1}$

Duty cycle: $\%on = 100 \% * \text{PWMYX} / \text{PWMYCtrl1}$

An example

Activate duty cycle mode and channels 0A and 0B, prescaler to 1024:

PWM0Ctrl0 = 249 (decimal) or 11111001b (binary)

In PWM0Ctrl1L/H, the following 16-bit value is written: 5000 (decimal).

The 16-bit value 5000 (decimal) is written in PWM0XL/H. The duty cycle is thus 50%.

If a higher value is written in PWM0XL/H than in PWM0Ctrl1L/H, the PWM channel remains continuously at logical "1". Continuous logic "0" is reached by the value "0" in PWM0XL/H.

In this example, the frequency is 1.56 Hz.

We recommend to first test the implementation of the PWM functions in your programs without connected devices and actuators and to check it with an oscilloscope.

Set the PWM values with care!

CAUTION

Actuators can be damaged or destroyed because of incorrect frequencies or duty cycles. Check your programs initially with an oscilloscope before connecting devices to the PWM channels.

12.2.1.4.4.3. Universal mode

In universal mode, the bytes PWMYAL and PWMYAH contain a related 16-bit data word. The data word is used to set the duty cycle of channel A.

The B channel is not adjustable in this mode. It outputs a PWM with a 50% duty cycle and half the frequency of the A channel. The "Y" is replaced by the number of the group ("0", "1" or "2").

This mode is suitable for applications with different frequencies and when one channel of the duty cycle is to be configured.

PWMYAL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 67: PWMYAL byte

PWMYAH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 69: PWMYAH byte

The configuration of the A channel is handled in the same way as in duty cycle mode.

A change of the bytes PWMYBL and PWMYBH has no effect. The B channel depends on the frequency configuration of the A channel.

12.2.1.4.4.4. Frequency mode

In the frequency mode, the bytes PWMYXL and PWMYXH contain a related 16-bit data word. The data word is used to set the frequency of the respective channel. The “Y” is replaced by the number of the group (“0”, “1” or “2”). The “X” is replaced by the letter of the respective PWM channel (“A” or “B”).

On all channels configured in frequency mode, the duty cycle is set to 50% and cannot be changed.

This mode is ideal when many different frequencies are required but the duty cycle is not important. This is, for example, the case when the PWM channels are used to control stepper motor drivers that only react to signal edges and do not require a variable duty cycle. The speed of up to six stepper motors can be controlled with the PiXtend V2 -L-.

PWMYXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 70: Bits of the PWMYXL bytes

PWMYXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 70: Bits of the PWMYXH bytes

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWMYCtrl0, the fine adjustment via PWMYXL/H.

The maximum frequency in this mode is 20 kHz. When setting higher frequencies, the output signal is limited to 20 kHz.

The frequency is calculated as follows:

Frequency channel X = 16 MHz / 2 / Prescaler / PWMYX

An example

A frequency of 500 Hz should be output on channel 0A and 250 Hz on channel 0B (for PWM group 0).

- Activate channels A and B, select frequency mode, configure prescaler to 64

→ PWM0Ctrl0 = 123 (decimal) or 01111011b (binary)

- PWM0A (16-bit data word) = 250 (decimal)

→ Frequency channel A = $16 \text{ MHz} / 2 / 64 / 250 = 500 \text{ Hz}$

- PWM0B (16-bit data word) = 500 (decimal)

→ Frequency channel B = $16 \text{ MHz} / 2 / 64 / 500 = 250 \text{ Hz}$

If required, the channel(s) may be activated/deactivated in every cycle, then the PWM signals are not continuously present at the outputs.

NOTICE

A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.2.1.4.5 RetainDataOutX

The Retain data consists of 64 bytes per direction (RetainDataOut and RetainDataIn). RetainDataOut is the data that is transferred from the Raspberry Pi to the PiXtend micro-controller for the actual backup. The storage location of the Retain data is the micro-controller.

There is no specification for the data format of the 64 bytes. In the application software, you decide how the data is described and used. Under CODESYS the 64 bytes are offered as a byte array.

The RetainDataOutX are transferred in each cycle. If the Retain memory is activated, the data is stored in the micro-controller in case of a power failure or voltage interruption. They are available again in the RetainDataIn bytes at the next start.

12.2.1.5 Output data – DAC

There is a Digital to Analog Converter (DAC) on the PiXtend V2 -L-, which is directly controlled by the Raspberry Pi and is responsible for the two analog outputs. The DAC is not part of the micro-controller and the data from the DAC is not in the process image that is exchanged between the Raspberry Pi and the micro-controller.

The DAC has a data format defined by the chip manufacturer, which is explained below. For further information, refer the data sheet of the chip⁹. The 10-bit version of the chip is on the PiXtend.

12.2.1.5.1 AnalogOutX (L/H)

The DAC receives one 16-bit data word per channel. The DAC does not provide an “reply” - there is only one data direction. The **X** in AnalogOut**X** for channel A or B of the chip.

The chip is connected to the Raspberry Pi via the SPI bus (ChipSelect - CS1). First the high byte (AnalogOut**XH**) then the low byte (AnalogOut**XL**) is transmitted.

On the PiXtend V2 -L-, channel A is responsible for AO0 and channel B for AO1.

AnalogOutXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|----|----|----|----|----|----|---|---|
| Name | D5 | D4 | D3 | D2 | D1 | D0 | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 71: Bits of the AnalogOutXL bytes

The 10-bit value for the output value is in AnalogOut**XL** and in AnalogOut**XL** - D0 (LSB) to D9 (MSB).
The low byteAnalogOut**XL** contains the lower 6 bits. The bits “0” and “1” of AnalogOut**XL** are not evaluated.

⁹Microchip MCP4812

AnalogOutXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|------|---|-----|-------|----|----|----|----|
| Name | /A-B | - | /GA | /SHDN | D9 | D8 | D7 | D6 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 72: Bits of the AnalogOutXH bytes

The upper four bits of the 10-bit output value are located in AnalogOutXH.

The three configuration bits are in the AnalogOutXH byte:

Bit 4 – /SHDN

The abbreviation SHDN means “Output Shutdown Control”. The channel set by bit 7 (/A-B) can be activated or deactivated via the /SHDN bit. If a “1” is written in /SHDN, the channel is active.

After a power-up, the channel is on the value “0” and therefore deactivated (start value). In the deactivated state, a voltage of less than 50 mV is output at the analog outputs of the PiXtend V2 -L-, regardless of the value to which the outputs are set.

Bit 5 – /GA

Setting the “Output Gain Selection”. For the intended operating case of the PiXtend V2 -L- (0 - 10V output voltage), the /GA bit must always be set to “0” (start value).

Bit 7 – /A-B

Setting the channel - A or B. If the bit contains a “1”, the 16-bit data word consisting of AnalogOutXL and AnalogOutXH is used for channel B. If bit 7 is set to “0” (start value), it is addressed for channel A.

All further information can be found in the data sheet of the DAC chip - microchip MCP4812.

12.2.1.6 Input data

Input data is the data that the PiXtend micro-controller transmits to the Raspberry Pi, for example, the digital and analog inputs.

12.2.1.6.1 DigitalInX

DigitalIn0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | DI7 | DI6 | DI5 | DI4 | DI3 | DI2 | DI1 | DI0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 73: Bits of the DigitalIn0 bytes

The first eight digital inputs (DigitalIn0) are organized in one byte. The least significant bit (LSB) contains the state of DI0, the most significant (MSB) contains the state of DI7.

DigitalIn1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|------|------|------|------|------|------|-----|-----|
| Name | DI15 | DI14 | DI13 | DI12 | DI11 | DI10 | DI9 | DI8 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 74: Bits of the DigitalIn1 bytes

The next eight digital inputs (DigitalIn1) are organized in one byte. The least significant bit (LSB) contains the state of DI8, the most significant (MSB) contains the state of DI15.

The inputs have the value “0” in the idle state. In the active state (voltage applied), the value changes to “1”. Refer to the PiXtend V2 -L- Hardware Manual for which level results in a “1” or “0”.

It is possible to debounce the digital inputs, which is useful for many applications. Further information is available in section 12.2.2.3.4 DigitalInBounce.

12.2.1.6.2 AnalogInX (L/H)

AnalogInXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 75: AnalogInXL

AnalogInXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|-----|---|
| Name | - | - | - | - | - | - | MSB | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 76: Bits of the AnalogInXH bytes

The analog/digital converter integrated in the PiXtend controller sends 10-bit values. The values are organized in the process data in a 16-bit data word consisting of two bytes. The most significant two bits of the 10-bit value are in the byte AnalogInXH.

The “X” is replaced by the letter of the respective analog channel (“0” or “1”). The analog inputs are already digitally filtered in the micro-controller. Ten values are recorded at a time, the average value is calculated and the result is stored in AnalogInX.

The value range is between “0” and “1023”. The analog inputs are marked AI0, AI1, AI2, AI3, AI4 and AI5 on the module and on the stainless-steel cover. AI0 - AI3 are voltage inputs (0 - 5V or 0 - 10V), the inputs AI4 and AI5 are current inputs (0 - 20 mA / 4 - 20 mA).

To calculate the AI0 - AI3 voltages from these values, the following formula is used:

Convert analog value to voltage:

$\text{AnalogInX} \cdot 10 / 1024 = \text{voltage at channel X [V]}$

(voltages are measured at AI0 and AI3)

Note the jumper for the voltage inputs. The “10” in the formula above is used if no jumper is set (0 - 10V range). If the jumper is inserted for one channel (0 - 5V range), a “5” instead of a “10” must be used in the formula.

To calculate the AI4 - AI5 current from these values, the following formula is used:

Convert analog value to current

$\text{AnalogInX} \cdot 0.020158400229358 = \text{current at channel X [V]}$

(currents are measured at AI4 and AI5)

12.2.1.6.3 GPIOIn

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|-------|-------|-------|-------|
| Name | - | - | - | - | GPIO3 | GPIO2 | GPIO1 | GPIO0 |
| Start value | 0 | 0 | 0 | 0 | 0/1 | 0/1 | 0/1 | 0/1 |

Table 77: Bits of the GPIOIn bytes

The four GPIO inputs (GPIO_IN) are organized in one byte. The least significant bit (LSB) contains the status of GPIO0. Bit 7 (MSB) to bit 4 are not assigned.

The GPIO inputs have no defined idle state* (floating input). Only when a voltage is applied externally does a stable state result:

0V → value "0"

5V → value "1"

Since the inputs are floating, they can be at "0" or "1" without an external connection and can oscillate between the values.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information can be found in chapter Control and Status - Bytes.

After a reset or power-up of Pixtend, the GPIOs are configured as inputs.

* It is possible to configure the GPIOs as inputs and to activate pull up resistors via the GPIOOut. This creates a "1" at the input in idle state (without this, they are floating inputs). Further information is available in section GPIOOut.

12.2.1.6.4 TempX (L/H)

TempXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 78: TempXL byte

TempXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 79: TempXH byte

In the 1-Wire/DHT mode of the GPIOs, the bytes TempXL and TempXH contain a related 16-bit data word. This data word contains the temperature information of a sensor (if a sensor is connected and configured). The “X” stands for the number of the GPIO, i.e. “0” to “3”.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information can be found in chapter Control and Status - Bytes.

The contained value can be converted into a temperature value (floating point value). However, a distinction must be made between DHT11 and DHT22 (TempX represents the 16-bit value):

DHT11: $\text{TempX} / 256 = \text{floating-point value } [^{\circ}\text{C}]$

Example: $5632 / 256 = 22.0\text{ }^{\circ}\text{C}$ - DHT11 sensors do not provide decimal places!

DHT22: $\text{TempX} / 10 = \text{floating-point value } [^{\circ}\text{C}]$

Example: $224 / 10 = 22.4\text{ }^{\circ}\text{C}$

Depending on the programming language and data type used, a “typecast” must be performed. If not, then after dividing by 10 (DHT22), no floating-point value results, but the integer value (in the example 22 instead of 22.4).

Dividing by 256 for the DHT11 sensors can be achieved with a “right shift” by 8 digits. DHT22 sensors can also detect negative temperatures, to detect this the MSB must be evaluated in TempXH. If the MSB is a “1”, then it is a negative temperature.

12.2.1.6.5 HumidX (L/H)

HumidXL

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 80: HumidXL byte

HumidXH

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 81: HumidXH byte

In the 1-Wire/DHT mode of the GPIOs, the bytes HumidXL and HumidXH contain a related 16-bit data word. This data word contains the humidity information of a sensor (if a sensor is connected and configured). The “X” stands for the number of the GPIO, i.e. “0” to “3”.

The GPIOs can perform three different tasks: control input, output or temperature/humidity sensors DHT11/22, AM2302. The configuration is done via the control byte GPIOCtrl. Further information can be found in chapter Control and Status - Bytes.

The contained value can be converted into a humidity value (floating point value). However, a distinction must be made between DHT11 and DHT22 (HumidX represents the 16-bit value):

DHT11: $\text{HumidX} / 256 = \text{floating point value [\%]} - \text{relative humidity}$

Example: $10496 / 256 = 41.0\%$ - DHT11 sensors do not provide decimal places!

DHT22: $\text{HumidX} / 10 = \text{floating point value [\%]} - \text{relative humidity}$

Example: $417 / 10 = 41.7 \%$

Depending on the programming language and data type used, a “typecast” must be performed. If not, then after dividing by 10, no floating-point value results, but the integer value (in the example 41 instead of 41.7). Dividing by 256 for the DHT11 sensors can be achieved with a “right shift” by 8 digits.

12.2.1.6.6 RetainDataInX

The Retain data consists of 64 bytes per direction (RetainDataOut and RetainDataIn). RetainDataIn is the backed-up data that is transferred from the PiXtend micro-controller to the Raspberry Pi after a power failure/power interruption. The data is available in RetainDataIn after the PiXtend system is restarted, provided that the Retain memory has been activated beforehand.

The data is retained until new retain data is stored. The data can be retained over any number of power cycles of the system if the Retain functionality is not activated again. If the Retain memory is activated again and the power supply is interrupted, the data is overwritten by the current data in RetainDataOutX.

12.2.2. Control and status bytes

PiXtend or the PiXtend micro-controller can be configured via control bytes. Furthermore, the micro-controller informs the Raspberry Pi about status bytes, about the status and about errors and warnings.

The following pages give an overview of the control and status bytes, then each byte is described in detail. Different possibilities are shown, and example calculations are performed.

If any questions remain open, please inform via e-mail (support@pixtend.de). You will receive an answer and further information as soon as possible.

12.2.2.1 Overview of the control bytes

Control bytes are transferred from the Raspberry Pi to the PiXtend micro-controller and can control the behavior of the micro-controller.

The following control bytes are available:

- ModelOut
This indicates which hardware version (model) that the RPi software (e.g. CODESYS or pxdev) expects
- UCMODE
This specifies which transfer mode is to be used
- UCCTRL
This is basic settings of the micro-controller (Retain, Watchdog, Safe State...)
- DigitalInDebounce
This indicates whether and how the digital inputs can be debounced
- GPIOCTRL
This configures the PiXtend GPIOs (input, output, DHT mode)
- GPIODEBOUNCE
This indicates whether and how the GPIO inputs can be debounced
- PWMYCTRL
This configures the PWM channels (mode, enable, frequencies)

12.2.2.2 Overview of the status bytes

Status bytes are transferred from the PiXtend micro-controller to the Raspberry Pi and contain important information about the status of PiXtend or the micro-controller.

The following status bytes are available:

- Firmware
This informs about the micro-controller firmware version
- **Hardware**
This informs about the PiXtend hardware version
- ModelIn
This informs about the PiXtend model
- UCSTATE
This contains the operating state and error codes of the micro-controller
- UCWARNINGS
This contains the warnings of the micro-controller

12.2.2.3 Description of the control bytes

12.2.2.3.1 ModelOut

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | x | x | x | x | x | x | x | x |

Table 82: ModelOut byte

By means of the byte ModelOut, the application software running on the Raspberry Pi informs the micro-controller which PiXtend model it expects. The micro-controller decides if it is the expected model and if not, it can return an error to the Raspberry Pi. The content of ModelOut is an ASCII character. With the PiXtend V2 -L-, the byte contains the following value: ASCII character **L** (capital letter L - 76 decimal / 4C hexadecimal).

There is no reason for the user to change this value. As an example, the CODESYS package “PiXtend V2 Professional for CODESYS” already includes drivers for which the model is permanently programmed and cannot be changed.

12.2.2.3.2 UCMoDe

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|---|
| Name | - | - | - | - | - | - | - | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 83: Bits of the UCMoDe bytes

Currently (as of June 2018), there is only one transmission mode, “Auto Mode”. The byte is reserved for future use.

For the operation of PiXtend, it is not necessary to enter a value here. If a value is entered, this has no effect.

12.2.2.3.2.1. UCContrl

12.2.2.3.2.2. UCContrl0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|------|------|------|------|
| Name | - | - | - | - | WDE3 | WDE3 | WDE1 | WDE0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 84: Bits of the UCContrl0 bytes

Bit 0 - 3 – WDE0 - 3 (WatchdogEnable0 - 3)

With the WDE0 - 3 bits, the watchdog timer of the PiXtend micro-controller can be activated and set.

All bits are "0" in the standard setting. The activation and setting of the watchdog are explained in the following table:

| WDE3 | WDE2 | WDE1 | WDE0 | Status Watchdog | Configured time |
|------|------|------|------|-----------------|-----------------|
| 0 | 0 | 0 | 0 | deactivated | deactivated |
| 0 | 0 | 0 | 1 | active | 16 ms |
| 0 | 0 | 1 | 0 | active | 32 ms |
| 0 | 0 | 1 | 1 | active | 64 ms |
| 0 | 1 | 0 | 0 | active | 0.125 s |
| 0 | 1 | 0 | 1 | active | 0.25 s |
| 0 | 1 | 1 | 0 | active | 0.5 s |
| 0 | 1 | 1 | 1 | active | 1 s |
| 1 | 0 | 0 | 0 | active | 2 s |
| 1 | 0 | 0 | 1 | active | 4 s |
| 1 | 0 | 1 | 0 | active | 8 s |

Table 85: Watchdog setting

If the watchdog is activated, the communication between the Raspberry Pi and PiXtend is monitored. If there is a pause between two valid cycles which is longer than the set time, the watchdog becomes active and puts the micro-controller into a safe state*. An invalid cycle (for example due to a CRC error) is evaluated by the watchdog as if no cycle had been performed.

The use of the watchdog is useful if the PiXtend controller is to be set to a defined state in the event of an error. An error can occur if the application program on the Raspberry Pi does not react and no data is transferred to the micro-controller.

Recommendation

Do not activate the watchdog during the development of your application software. If a new program is transferred via CODESYS or the user software is debugged, no transfer takes place for a certain time. The watchdog will become active and a restart of the system is necessary. This leads to an unnecessary delay of your development process.

* Definition "Safe State":

- All digital outputs and relays are switched off/ put into the idle state
- PWM outputs are switched to high impedance (tri-state)
- Retain data is stored when Retain option has been activated
- The status LED "L1" flashes depending on the cause of the error (Error LED "L1" Signals) when the LED is activated
- The micro-controller or the PiXtend device has to be restarted (power cycle)

If switching off the digital outputs does not correspond to the safe state of your application, external measures must be taken to prevent any critical or dangerous situations.

12.2.2.3.2.3. UC Ctrl1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|------|------|----|----|------|
| Name | - | - | - | GPUE | SLED | RE | RC | SAFE |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 86: Bits of the UC Ctrl1 bytes

Bit 0 – SAFE (SafeState)

The SAFE bit, when activated by the value “1”, can immediately put the micro-controller into the safe state*. The bit has the value “0” by default.

* Definition “Safe State”:

- All digital outputs and relays are switched off/ put into the idle state
- PWM outputs are switched to high impedance (tri-state)
- Retain data is stored when Retain option has been activated
- The status LED “L1” flashes depending on the cause of the error (Error LED “1” – Signaling) when the LED is activated
- The micro-controller or the PiXtend device has to be restarted (power cycle)

If switching off the digital outputs does not correspond to the safe state of your application, external measures must be taken to prevent any critical or dangerous situations.

Bit 1 – RC (RetainCopy)

The RC bit can be used to configure which data is visible in the Retain Input area (RetainDataIn0 - 63). At the start value “0”, the last saved data is transferred from the micro-controller to the Raspberry Pi, normal Retain operation. If the value “1” is set for the RC bit, the last data sent by the Raspberry Pi is returned. In the Retain area RetainDataIn0 - 63 (RetainDataOut0 -.63) is mirrored, but with a cycle delay (Retain Copy mode). The content of the Retain data is not lost, it is just not displayed as long as the RC bit is “1”.

For more information on the PiXtend Retain function, see 6.4 Retain memory.

Bit 2 – RE (RetainEnable)

With the RE bit, the retain function of the PiXtend can be activated. For this purpose, a “1” is written into this bit. By default, RE is “0”, after a reset or power-up and therefore Retain is deactivated.

Recommendation

Only activate Retain function if the functionality is really needed. The number of retain memory can only be guaranteed up to 10,000 cycles. For more information on the PiXtend V2 Retain function refer to chapter: 6 Basic knowledge.

Bit 3 – SLED (StatusLED)

The status LED “L1” can be deactivated via the SLED bit, if it is not needed. The status LED is active by default and can be deactivated by the value “1” in SLED.

Bit 4 – GPUE (GPIOPullUpEnable)

The GPUE bit can be used to enable the pull-up resistors of the PiXtend GPIOs, but they are not yet activated by this process. Enabling it has an effect if the GPIOs are configured as inputs and a “1” is written in the byte GPIOOut for the respective GPIO. A “1” in GPUE activates this functionality. For further information on PiXtend GPIOs, refer to the section: GPIOOut.

12.2.2.3.3 DigitalInDebounce

DigitalInDebounce01

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 87: Bits of the DigitalInDebounce01 bytes

DigitalInDebounce23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 88: Bits of the DigitalInDebounce23 bytes

DigitalInDebounce45

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 89: Bits of the DigitalInDebounce45 bytes

DigitalInDebounce67

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 90: Bits of the DigitalInDebounce67 bytes

DigitalInDebounce89

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 91: Bits of the DigitalInDebounce89 bytes

DigitalInDebounce1011

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 92: Bits of the DigitalInDebounce1011 bytes

DigitalInDebounce1213

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 93: Bits of the DigitalInDebounce1213 bytes

DigitalInDebounce1415

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 94: Bits of the DigitalInDebounce1415 bytes

With the help of the DigitalInDebounce bytes, the debouncing of the 16 digital inputs of PiXtend V2 -L- can be configured. By default, the debouncing of the inputs is not active. Debouncing is activated/deactivated for two channels together. There are no interactions between the two channels, they only receive the same debounce setting.

The number in a DigitalInDebounce byte corresponds to the number of cycles in which an electrical signal applied to the digital input must remain constant in order to pass on the level change to the application software (Raspberry Pi). This applies to a change of the signal level from "0" (low) to "1" (high) and vice versa. If the bytes are not changed or a "0" is written as a value, no debouncing takes place and the application software receives a new value in each cycle.

An example:

The first two digital inputs (DI0 and DI1) are to be debounced. Only changes to the digital inputs that are constant for longer than 100 ms are visible. The cycle time (data exchange between PiXtend and Raspberry Pi) is 10 ms.

- The byte DigitalInDebounce01 is used
- Calculation of the value for DigitalInDebounce01: $100 \text{ ms} / 10 \text{ ms} = 10$

The byte is written with a 10 (decimal) to get the desired effect. If the cycle time is changed during software development, the value for debouncing must be adjusted.

12.2.2.3.4 GPIOCtrl

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-------|-------|-------|-------|-----|-----|-----|-----|
| Name | SENS3 | SENS2 | SENS1 | SENS0 | IO3 | IO2 | IO1 | IO0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 95: Bits of the GPIOCtrl bytes

Bits 0 - 3 – IO0 - 3

With the IO bits, the four PiXtend GPIOs can be configured as digital input or output. All GPIOs are configured as inputs with the start value “0”. If the IO3 bit is set to “1”, GPIO3 becomes an output. The value is set via the data word GPIOOut.

Bits 4 - 7 – SENS0 - 3 (Sensor0 - 3)

The GPIOs can be used in addition to the input/output for 1-Wire sensors (DHT11, DHT22, AM2302), for which the upper four bits of the GPIOCtrl data word are available. If the SENS3 bit is set to “1”, a sensor can be connected to GPIO3 and evaluated. The setting of the IO bit is not important in this case, the SENSX bits prevail over the IOX bits.

The results/measured values of the sensors are in TempXL/TempXH and HumidXL/HumidXH. The “X” corresponds to the number of the GPIO to which a sensor is connected.

NOTICE

Communication with the sensors requires time (ca. 25 ms), using at least one sensor results in a minimum cycle time of 30 ms. It does not matter whether one or four sensors are queried.

12.2.2.3.5 GPIODebounce

GPIODebounce01

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 96: Bits of the GPIODebounce01 bytes

GPIODebounce23

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|-----|
| Name | MSB | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 97: Bits of the GPIODebounce23 bytes

Debouncing the GPIOs is possible if they are configured as inputs. The GPIODebounce bytes are used just like the DigitalInDebounce bytes. More information and examples are in the section: DigitalInDebounce.

12.2.2.3.6 PWMYCtrl – for 16-bit PWMs

The following descriptions refer to all 16-bit PWMYX channels (PWM0A, PWM0B, PWM1A, PWM1B, PWM2A & PWM2B).

The “Y” is replaced by the number of the group (“0”, “1” or “2”) and the “X” by the letter of the respective PWM channel (“A” or “B”).

12.2.2.3.6.1. PWMYCtrl0

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|-----|-----|---------|---------|---|-------|-------|
| Name | PS2 | PS1 | PS0 | EnableB | EnableA | - | MODE1 | MODE0 |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 98: Bits of the PWMYCtrl0 bytes

Bit 0 - 1 – MODE0 - 1

The PWM outputs can be operated in four different modes. The mode is configured via the MODE bits. PiXtend starts in servo mode (MODE0 - 1 = 00) by default.

| MODE1 | MODE0 | Mode | Description |
|-------|-------|-----------------|---|
| 0 | 0 | Servo mode | Both channels (A/B) are used to control model construction servos - special signal form |
| 0 | 1 | Duty cycle mode | Both channels (A/B) have the same frequency, but an independently adjustable duty cycle |
| 1 | 0 | Universal mode | Free adjustable frequency and duty cycle for channel A, half frequency for channel B and 50% duty cycle |
| 1 | 1 | Frequency mode | Freely adjustable frequencies for both channels (A/B), the duty cycle is always 50%. |

Table 99: Mode bits of the PWMYCtrl0 byte

The Servo mode and duty cycle mode are sufficient, the universal and frequency mode can be useful and helpful for special applications but are more complex to configure and use.

Bit 3 - 4 – EnableA, EnableB

With the bits EnableA and EnableB, the respective channel can be activated. By default, the PWM channels are deactivated. By writing a “1” into the respective bit, the channel is activated and the PWM becomes visible at the output.

The PWM can first be configured for the necessary requirements (mode, frequency/value...) and only activated in the second step. The activation is possible in the same cycle.

Bit 5 - 7 – PS0 - PS2 (Prescaler0 - 3)

The prescaler bits (PS bits) enable the approximate setting of the PWM frequency. Depending on the PWM mode (see MODE bits), the PS bits have different effects.

The following table shows the effects of the PS bits:

| PS2 | PS1 | PS0 | Description | Base frequency |
|-----|-----|-----|-------------------------|----------------|
| 0 | 0 | 0 | PWM outputs deactivated | - |
| 0 | 0 | 1 | Prescaler: 1 | 16 MHz |
| 0 | 1 | 0 | Prescaler: 8 | 2 MHz |
| 0 | 1 | 1 | Prescaler: 64 | 250 kHz |
| 1 | 0 | 0 | Prescaler: 256 | 62.5 kHz |
| 1 | 0 | 1 | Prescaler: 1024 | 15.625 kHz |
| 1 | 1 | 0 | Prescaler: 1024 | 15.625 kHz |
| 1 | 1 | 1 | Prescaler: 1024 | 15.625 kHz |

Table 100: Prescaler bits of the PWMYCtrl0 byte

The prescaler settings have different effects depending on the selected mode:

| Mode | Description |
|-----------------|--|
| Servo mode | No effect |
| Duty cycle mode | Setting of the prescaler or the base frequency |
| Universal mode | Setting of the prescaler or the base frequency |
| Frequency mode | Setting of the prescaler or the base frequency |

Table 101: PWM mode

The approximate setting of the frequency is done via the prescaler bits (PS0 - 2) in PWMYCtrl0, the fine adjustment via PWMYCtrl1L/H.

See section PWMYX – 16-bit Resolution for the exact frequency/period duration setting.

NOTICE

If the frequency mode is to be used, no DHT sensors must be connected and activated in the GPIOCtrl byte. A combination of frequency mode and DHT sensors is not possible. If one or more DHT sensors are activated at the PiXtend GPIOs, the frequency mode cannot be activated. If you try to activate the frequency mode, although DHT sensors are used, the PWM outputs are deactivated.

12.2.2.3.6.2. PWMYCtrl1 (L/H)

The PWMYCtrl1 bytes L and H contain a related 16-bit data word, PWMYCtrl1L is the low byte and PWMYCtrl1H is the high byte.

With the two bytes, the PWM frequency/period duration can be set (in PWM mode “duty cycle” and “universal”). The frequency is valid for both channels.

In the duty cycle mode, the duty cycle for each channel can be configured independently, in universal mode the duty cycle can only be set for channel A.

In the frequency mode, the PWMYCtrl1 bytes have no effect.

PWMYCtrl1L

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|---|---|---|-----|
| Name | | | | | | | | LSB |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 102: PWMYCtrl1L byte

PWMYCtrl1H

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|-----|---|---|---|---|---|---|---|
| Name | MSB | | | | | | | |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 103: PWMYCtrl1H byte

Example for the PWM period duration in “duty cycle mode”:

The prescaler (PS0 - 2) is set to the value 64 and thus the basic frequency to 250 kHz, in addition the duty cycle mode and channel PWM0A are activated:

PWM0Ctrl0: 01101001b

The 16-bit data word PWM0Ctrl1 is set to the value 1000:

PWM0Ctrl1L: 11101000b

PWM0Ctrl1H: 00000011b

PWM period duration = micro-controller basic clock / 2 / prescaler / PWMYCtrl1

PWM period duration = 16 MHz / 2 / 64 / 1000 = 125 Hz

12.2.2.4 Description of the status bytes

The status bytes inform the user and the application program running on the Raspberry Pi about the status of the micro-controller. The status bytes can be queried via CODESYS, PiXtend-C-Library or PiXtend-Python-Library (PPL).

12.2.2.4.1 Firmware

In the firmware byte, there is a number that stands for the firmware version, the number can be between 0 and 255. If you request support for your PiXtend V2 device, please always tell us the firmware version.

12.2.2.4.2 Hardware

The hardware byte contains the version number of your PiXtend board. With the version number, the user software can check whether the software and drivers match the existing hardware.

Example:

Hardware byte contains the number 21
→ PiXtend hardware version 2.1

12.2.2.4.3 ModelIn

In the ModelIn byte the micro-controller informs the user software about the PiXtend V2 model. With the model identifier, the user software can check whether the software and drivers match the existing hardware.

The content of ModelIn is an ASCII character. With the PiXtend V2 -L-, the byte contains the following value: ASCII character **L** (capital letter L - 76 decimal / 4C hexadecimal).

12.2.2.4.4 UCState

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|------|------|------|------|---|---|---|-----|
| Name | ERR3 | ERR2 | ERR1 | ERR0 | - | - | - | RUN |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 104: Bits of the UCState bytes

Bit 0 – RUN

By querying this bit, the user can check whether PiXtend is ready for operation ("1") and whether a successful data transmission has been performed. This can be useful after starting the PiXtend system.

If the controller is set to the safe state, e.g. by the watchdog, the bit contains a "0". Since no communication is possible until a restart, this bit cannot be queried.

Bits 4 - 7 – ERROR0 - 3

The ERR bits provide information about problems that the micro-controller detects and passes on to the user program. The following table describes the error codes and their causes:

| ERR3 | ERR2 | ERR1 | ERR0 | Error description |
|------|------|------|------|--|
| 0 | 0 | 0 | 0 | No error |
| 0 | 0 | 0 | 1 | - |
| 0 | 0 | 1 | 0 | CRC error - data the payload data received from the Raspberry Pi is incorrect |
| 0 | 0 | 1 | 1 | Data block too short Driver faulty/incorrect model defined |
| 0 | 1 | 0 | 0 | PiXtend model does not match the expected and the actual model do not match (ModelIn and ModelOut are unequal) |
| 0 | 1 | 0 | 1 | CRC error - Header the header data received from the Raspberry Pi is incorrect |
| 0 | 1 | 1 | 0 | SPI frequency too high the set SPI frequency is too high, transmission errors occur |

Table 105: Breakdown of the error bits in the UCState byte

12.2.2.4.5 UCWarnings

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|---|---|---|---|----|----|----|---|
| Name | - | - | - | - | W2 | W1 | W0 | - |
| Start value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 106: Bits of the UCWarnings bytes

Bits 1 - 2 – W0 - 1 (RetainCRCErrror. RetainVoltageError)

The W0 - 1 bits inform the user about warnings regarding the Retain functionality of the PiXtend V2. The user program can check for these bits and recognize whether the Retain memory is working correctly.

W0 – RetainCRCErrror

This is “1” if an error was detected when checking the retain memory.

W1 – RetainVoltageError

This is “1” if the current supply voltage of the PiXtend is too low (less than 19V). The Retain memory cannot be activated.

Bit 3 – W2 (I2CErrror)

The PiXtend micro-controller and the PWM controller are connected via the I²C bus.

W2 is “1” if an error has occurred during the check (I²C/TWI) of the PWM data for PWM1A, PWM1B, PWM2A and PWM2B.

12.3. Error LED “L1” Signals

There are error conditions that can occur in the micro-controller that may not be able to be reported back to the Raspberry Pi via the SPI bus. For example, in case of a faulty communication or if the watchdog of the micro-controller has been triggered. For this situation an LED (L1) was placed on the PiXtend V2 board to help the user to determine the source of a problem.

The meaning of the different LED (L1) signals can be found below.

| State (Signal) | Meaning | Possible solutions |
|-----------------------|------------------------------|---|
| Off | Ready, no errors | - |
| On/Blinking very fast | Communication error (SPI) | Check the switch “SPI_EN”, check the software component used for correct operation, only one program at a time is allowed to communicate with the micro-controller. In C programs, the input and output structure variables must be declared globally or locally with the keyword “static”. |
| Blinking fast (0.2 s) | Watchdog has been activated. | Shutdown the RPi and turn off all power supplies (power cycle). The micro-controller needs to restart in order to get it to work again. |
| Fading (3s) | AC/Retain error | The supply voltage is too low and the Retain function is activated, the retain data cannot be saved and is not working. Check if the supply voltage has 24 volts. |
| Pulsing (5s) | Shutdown (safe state) | The micro-controller is entering or is in safe state. The bit 0 within the control byte UCContr1 was set. |

Table 107: LED “L1” Signals

List of Figures

| | |
|---|----|
| Figure 1: Software - Win32DiskImager | 22 |
| Figure 2: Basic knowledge – Determining the IP address of the Raspberry Pi in CODESYS | 25 |
| Figure 3: Basic knowledge – List of all Raspberry Pi devices found in the network | 25 |
| Figure 4: RPi - Turning off Bluetooth® | 30 |
| Figure 5: CODESYSControl_User.cfg - serial connection ttyAMA | 31 |
| Figure 6: CODESYS Development System (source: 3S) | 37 |
| Figure 7: CODESYS - Visualization for the V2 demo project | 38 |
| Figure 8: CODESYS - New project | 42 |
| Figure 9: CODESYS - New project | 43 |
| Figure 10: CODESYS - New project | 43 |
| Figure 11: CODESYS - Add device | 44 |
| Figure 12: CODESYS - Add device - Modbus master | 45 |
| Figure 13: CODESYS - Add device - PiXtend V2 -S- | 46 |
| Figure 14: CODESYS - Adding Global Variable List | 47 |
| Figure 15: CODESYS - GVL – Declarations | 48 |
| Figure 16: CODESYS - PiXtend V2 -S- I/O mapping | 49 |
| Figure 17: CODESYS - Input assistance | 50 |
| Figure 18: CODESYS - PiXtend V2 -S- I/O mapping - State | 51 |
| Figure 19: CODESYS - GPIO parameter - GPIO24 | 51 |
| Figure 20: CODESYS - GPIO I/O mapping - rpi_gpio24 variable | 52 |
| Figure 21: CODESYS - Task Configuration - Main Task | 53 |
| Figure 22: CODESYS - PLC_PRG (CFC) - Input | 54 |
| Figure 23: CODESYS - CFC - Three inputs connected to three inputs | 55 |
| Figure 24: CODESYS - CFC - Configured | 55 |
| Figure 25: CODESYS - Communication - Scan Network | 56 |
| Figure 26: CODESYS - PiXtend V2 -S- I/O mapping online | 57 |
| Figure 27: CODESYS - PLC_PRG (CFC) - Demo Program | 58 |
| Figure 28: CODESYS - Web Visualization - Configuration | 59 |
| Figure 29: CODESYS - Visualization | 60 |
| Figure 30: CODESYS - Visualization - Element properties | 66 |
| Figure 31: CODESYS - Visualization - Configuration of a rectangle | 61 |
| Figure 32: CODESYS - Visualization - Completed | 62 |
| Figure 33: CODESYS - DAC - New project | 63 |
| Figure 34: CODESYS - DAC - Select device | 64 |
| Figure 35: CODESYS - DAC - Project overview | 64 |
| Figure 36: CODESYS - DAC - Adding SPI device | 65 |
| Figure 37: CODESYS - DAC - Adding SPI master | 66 |
| Figure 38: CODESYS - DAC - Configuring SPI master | 67 |
| Figure 39: CODESYS - DAC - Adding PiXtend V2 DAC | 71 |
| Figure 40: CODESYS - DAC - I/O mapping settings | 72 |
| Figure 41: CODESYS - DAC - Adding Global Variable List | 68 |
| Figure 42: CODESYS - DAC - GVL Variables | 68 |
| Figure 43: CODESYS - DAC - Variable mapping | 69 |
| Figure 44: CODESYS - DAC - Using GPIO 24 as an output | 69 |
| Figure 45: CODESYS - DAC - GPIO 24 | 70 |
| Figure 46: CODESYS - DAC - Setting variable rpi_gpio24 | 73 |
| Figure 47: CODESYS - DAC - Scan Network | 74 |
| Figure 48: CODESYS - DAC - WebVisu settings | 75 |
| Figure 49: CODESYS - DAC - WebVisu - Sliders | 76 |
| Figure 50: CODESYS - DAC - Slider - Setting scale end | 76 |
| Figure 51: CODESYS - DAC - WebVisu - Text field settings | 77 |
| Figure 52: CODESYS - DAC - WebVisu - Completed | 78 |
| Figure 53: CODESYS - Information on how to use both PiXtend V2 SPI devices | 79 |
| Figure 54: CODESYS - RS232 wiring diagram | 80 |
| Figure 55: CODESYS - RS485 wiring diagram | 81 |
| Figure 56: CODESYS - HTerm 0.8.1beta - RS232 test program | 84 |
| Figure 57: CODESYS - Visualization - Test program - Serial communication | 85 |
| Figure 58: CODESYS - CAN bus test under Linux | 92 |
| Figure 59: CODESYS - CAN bus - Creating a new project | 95 |
| Figure 60: CODESYS - CAN bus slave - Adding SPS device | 96 |
| Figure 61: CODESYS - CAN bus slave - Adding Raspberry Pi | 96 |
| Figure 62: CODESYS - CAN bus slave - Adding a device to the SPS | 97 |
| Figure 63: CODESYS - CAN bus slave - Adding CAN bus device | 98 |

| | |
|--|-----|
| Figure 64: CODESYS - CAN bus slave - Adding CANopen device | 99 |
| Figure 65: CODESYS - CAN bus slave - Configuring I/O area | 100 |
| Figure 66: CODESYS - CAN bus slave - Setting PDO | 100 |
| Figure 67: CODESYS - CAN bus slave - Installing your own CAN device | 101 |
| Figure 68: CODESYS - CAN bus slave - POU and task configuration | 102 |
| Figure 69: CODESYS - CAN bus slave - Task configuration | 103 |
| Figure 70: CODESYS - CAN bus slave - CANopen I/O mapping | 103 |
| Figure 71: CODESYS - CAN bus slave - Baud rate..... | 104 |
| Figure 72: CODESYS - CAN bus master - Adding Raspberry Pi | 105 |
| Figure 73: CODESYS - CAN bus master - Adding a CANbus | 106 |
| Figure 74: CODESYS - CAN bus master - Adding a CANopen manager | 107 |
| Figure 75: CODESYS - CAN bus master - Adding CAN bus slave device | 108 |
| Figure 76: CODESYS - CAN bus master - CAN slave configuration | 109 |
| Figure 77: CODESYS - CAN bus test - Multiple download | 110 |
| Figure 78: CODESYS - CAN bus test - Master and slave exchange data | 111 |
| Figure 79: CODESYS - Retain demo - Device tree | 112 |
| Figure 80: CODESYS - Retain - Variable declaration | 113 |
| Figure 81: CODESYS - Retain - Example program | 114 |
| Figure 82: CODESYS - Retain – Start value 13 | 115 |
| Figure 83: CODESYS - Retain - arRetainDataOut[0] increased by one to 14 | 115 |
| Figure 84: CODESYS - Retain - After a power cycle | 115 |
| Figure 85: CODESYS - Shutdown - Library Manager | 117 |
| Figure 86: CODESYS - Shutdown - Adding libraries | 118 |
| Figure 87: CODESYS - Shutdown - Variables | 119 |
| Figure 88: CODESYS - Shutdown - Example program | 119 |
| Figure 89: CODESYS PiXtend V2 -S- - SPI devices parameter tab | 120 |
| Figure 90: CODESYS PiXtend V2 -L- - SPI devices parameter tab | 125 |
| Figure 91: CODESYS - GPIO24 as an output | 131 |
| Figure 92: Linux Tools - pixtendtool2s - Switch '-di' | 141 |
| Figure 93: Linux Tools - pixtendtool2s - Switch '-ai' | 141 |
| Figure 94: Linux Tools - pixtendtool2s - Switch '-sww' | 141 |
| Figure 95: Linux Tools - pixtendtool2s - Switch '-sr' | 142 |
| Figure 96: Linux-Tools - pxauto2s - Main menu | 144 |
| Figure 97: Linux-Tools - pxauto2s - menu DOut | 146 |
| Figure 98: Linux-Tools - pxauto2s | 147 |
| Figure 99: Linux-Tools - pixtendtool2s | 147 |
| Figure 100: C program - 100ms cycle time | 162 |
| Figure 101: FHEM - Activation of the sensor functionality | 171 |
| Figure 102: FHEM - Creating a plot via the LogFile | 172 |
| Figure 103: FHEM - Diagram of the measured sensor values | 172 |
| Figure 104: Python - image source: https://www.python.org/ , The Python Brochure | 175 |
| Figure 105: Python - Raspbian PIXEL- Terminal icon in the task bar | 176 |
| Figure 106: Python - Console - PPLV2 directory | 177 |
| Figure 107: Python - PiXtend Python Library V2 demo program | 178 |
| Figure 108: Python - simple demo program | 179 |

List of Tables

| | |
|---|-----|
| Table 1: Raspberry Pi GPIO24 names | 20 |
| Table 2: Technical data – Retain system | 21 |
| Table 3: CODESYS - Add device - PiXtend V2 -S- | 124 |
| Table 4: CODESYS I/O Overview for PiXtend V2 -L- | 130 |
| Table 5: PiXtend V2 -S- SPI bus protocol overview | 187 |
| Table 6: Bits of the DigitalOut bytes | 189 |
| Table 7: Bits of the RelayOut bytes | 189 |
| Table 8: Bits of the GPIOOut bytes | 190 |
| Table 9: PWM0XL byte | 192 |
| Table 10: PWM0XH byte | 192 |
| Table 11: PWM0XL byte | 193 |
| Table 12: PWM0XH byte | 193 |
| Table 13: PWM0AL byte | 194 |
| Table 14: PWM0AH byte | 194 |
| Table 15: Bits of the PWM0AL bytes | 194 |
| Table 16: Bits of the PWM0AH bytes | 191 |
| Table 17: Bits of the PWM1XL bytes | 195 |
| Table 18: Bits of the PWM1XH bytes | 195 |
| Table 19: Bits of the PWM1XL bytes | 196 |
| Table 20: Bits of the PWM1XH bytes | 196 |
| Table 21: Bits of the PWM1AL bytes | 197 |
| Table 22: Bits of the PWM1AH bytes | 197 |
| Table 23: Bits of the PWM1AL bytes | 198 |
| Table 24: Bits of the PWM1AH bytes | 198 |
| Table 25: Bits of the AnalogOutL bytes | 199 |
| Table 26: Bits of the AnalogOutH bytes | 199 |
| Table 27: Bits of the DigitalIn bytes | 201 |
| Table 28: AnalogInXL | 202 |
| Table 29: Bits of the AnalogInXH bytes | 202 |
| Table 30: Bits of the GPIOIn bytes | 203 |
| Table 31: TempXL byte | 204 |
| Table 32: TempXH byte | 204 |
| Table 33: HumidXL byte | 205 |
| Table 34: HumidXH byte | 205 |
| Table 35: ModelOut byte | 208 |
| Table 36: Bits of the UCMODE bytes | 208 |
| Table 37: Bits of the UCCTRL0 bytes | 209 |
| Table 38: Watchdog setting | 209 |
| Table 39: Bits of the UCCTRL1 bytes | 211 |
| Table 40: Bits of the DigitalInDebounce01 bytes | 212 |
| Table 41: Bits of the DigitalInDebounce23 bytes | 212 |
| Table 42: Bits of the DigitalInDebounce45 bytes | 212 |
| Table 43: Bits of the DigitalInDebounce67 bytes | 212 |
| Table 44: Bits of the GPIOCTRL bytes | 214 |
| Table 45: Bits of the GPIODEBOUNCE01 bytes | 215 |
| Table 46: Bits of the GPIODEBOUNCE23 bytes | 215 |
| Table 47: Bits of the PWM0CTRL0 bytes | 216 |
| Table 48: Bits of the PWM0CTRL0 byte | 216 |
| Table 49: - Bits of the PWM0CTRL0 byte | 217 |
| Table 50: PWM mode | 217 |
| Table 51: PWM0CTRL1L byte | 218 |
| Table 52: PWM0CTRL1H byte | 218 |
| Table 53: Bits of the PWM1CTRL1L bytes | 221 |
| Table 54: Bits of the PWM1CTRL1H bytes | 221 |
| Table 55: Bits of the UCSTATE bytes | 224 |
| Table 56: Breakdown of the error bits in the UCSTATE byte | 224 |
| Table 57: Bits of the UCWARNINGS bytes | 225 |
| Table 58: PiXtend V2 -L- SPI protocol overview | 228 |
| Table 59: Bits of the DigitalOut0 bytes | 230 |
| Table 60: Bits of the DigitalOut1 bytes | 230 |
| Table 61: Bits of the RelayOut bytes | 231 |
| Table 62: Bits of the GPIOOut bytes | 232 |
| Table 63: PWMYXL byte | 233 |

| | |
|--|-----|
| Table 64: PWMYXH byte | 233 |
| Table 65: PWMYXL byte | 234 |
| Table 66: PWMYXH byte | 234 |
| Table 67: PWMYAL byte | 236 |
| Table 68: PWMYAH byte | 236 |
| Table 69: Bits of the PWMYXL bytes | 237 |
| Table 70: Bits of the PWMYXH bytes | 237 |
| Table 71: Bits of the AnalogOutXL bytes | 239 |
| Table 72: Bits of the AnalogOutXH bytes | 240 |
| Table 73: Bits of the DigitalIn0 bytes | 241 |
| Table 74: Bits of the DigitalIn1 bytes | 241 |
| Table 75: AnalogInXL | 242 |
| Table 76: Bits of the AnalogInXH bytes | 242 |
| Table 77: Bits of the GPIOIn bytes | 243 |
| Table 78: TempXL byte | 244 |
| Table 79: TempXH byte | 244 |
| Table 80: HumidXL byte | 245 |
| Table 81: HumidXH byte | 245 |
| Table 82: ModelOut byte | 248 |
| Table 83: Bits of the UCMODE bytes | 248 |
| Table 84: Bits of the UCCTRL0 bytes | 249 |
| Table 85: Watchdog setting | 249 |
| Table 86: Bits of the UCCTRL1 bytes | 251 |
| Table 87: Bits of the DigitalInDebounce01 bytes | 252 |
| Table 88: Bits of the DigitalInDebounce23 bytes | 252 |
| Table 89: Bits of the DigitalInDebounce45 bytes | 252 |
| Table 90: Bits of the DigitalInDebounce67 bytes | 252 |
| Table 91: Bits of the DigitalInDebounce89 bytes | 252 |
| Table 92: Bits of the DigitalInDebounce1011 bytes | 252 |
| Table 93: Bits of the DigitalInDebounce1213 bytes | 253 |
| Table 94: Bits of the DigitalInDebounce1415 bytes | 253 |
| Table 95: Bits of the GPIOCTRL bytes | 254 |
| Table 96: Bits of the GPIODEBOUNCE01 bytes | 255 |
| Table 97: Bits of the GPIODEBOUNCE23 bytes | 255 |
| Table 98: Bits of the PWMYCTRL0 bytes | 256 |
| Table 99: Bits of the PWMYCTRL0 byte | 256 |
| Table 100: - Bits of the PWMYCTRL0 byte | 257 |
| Table 101: PWM mode | 257 |
| Table 102: PWMYCTRL1L byte | 258 |
| Table 103: PWMYCTRL1H byte | 258 |
| Table 104: Bits of the UCSTATE bytes | 260 |
| Table 105: Breakdown of the error bits in the UCSTATE byte | 260 |
| Table 106: Bits of the UCWARNINGS bytes | 261 |
| Table 107: LED "L1" Signals | 262 |